Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

Tesis doctoral

Programa Oficial de Doctorado en
Tecnologías de la Información y la Comunicación

Técnicas avanzadas de optimización en ambientes dinámicos

Pavel Novoa Hernández

Directores:
Prof. Dr. David A. Pelta
Prof. Dr. Carlos A. Cruz Corona

Granada, 2013

Editor: Editorial de la Universidad de Granada
Autor: Pavel Novoa Hernández
D.L.: GR 716-2014
ISBN: 978-84-9028-877-1

El doctorando D. Pavel Novoa Hernández y los directores de la tesis Dr Carlos Cruz Corona y Dr. David A. Pelta garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Julio de 2013

Pavel Novoa Hernández

David A. Pelta

Carlos Cruz Corona

A mis padres y mi hermano...

Agradecimientos

Me gustaría agradecer a las siguientes personas e instituciones por su ayuda durante el desarrollo de esta investigación:

- Dr. David Alejandro Pelta y Dr. Carlos Cruz, mis tutores, por la motivación y orientaciones.
- Grupo de Trabajo en Modelos de Decisión y Optimización (MODO) del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, en especial a los profesores Dr. José Luis Verdegay y Dra. María Teresa Lamata, por el apoyo brindado.
- Al Programa de Doctorado Iberoamericano en Informática (Soft Computing), principalmente a sus patrocinadores y organizadores.
- Facultad de Informática y Matemática de la Universidad de Holguín por brindarme la oportunidad de formar parte de este programa doctoral, y apoyarme durante todo este tiempo. En particular a la Dra. Rosa Isabel Urquiza y el Dr. José Velázquez Codina.
- Programa de Becas del Grupo Coimbra para Profesores e Investigadores Jóvenes de Universidades Latinoamericanas.

En general, a todas aquellas personas que han enriquecido este trabajo con sus críticas y sugerencias. A todos muchas gracias.

Índice general

Índice de tablas

Tabla

Índice de figuras

Figura

Índice de algoritmos

Resumen

La presente investigación se desarrolla en el contexto de la optimización de problemas dinámicos mediante métodos meta-heurísticos multipoblacionales. Específicamente, las contribuciones se centran en la propuesta de mejoras a enfoques existentes aplicando técnicas de control de parámetros de tipo adaptativas y auto-adaptativas. En ese sentido, los objetivos trazados fueron los siguientes:

1. Proponer un herramienta para gestionar la experimentación en ambientes dinámicos, a partir de una organización adecuada de las medidas de rendimiento en ambientes dinámicos, y la inclusión de pruebas estadísticas no paramétricas.
2. Proponer estrategias adaptativas que mejoren el rendimiento del enfoque PSO multi-enjambre en ambientes dinámicos.
3. Investigar si la auto-adaptación, como técnica de control de parámetros, tiene sentido en ambientes dinámicos.
4. Analizar el rol de la auto-adaptación en ambientes dinámicos cuando es aplicada en distintas partes del algoritmo.

En relación al objetivo 1, se desarrolló la herramienta DynOptLab sobre la tecnología Java. Esta herramienta está compuesta por dos parte fundamentales: 1) un framework orientado a objetos que permite la inclusión de problemas, algoritmos y medidas de rendimiento, y 2) una interfaz gráfica de usuario orientada a la gestión y ejecución de experimentos computacionales. En este contexto, se propuso un marco de trabajo para organizar las medidas de rendimientos en ambientes dinámicos basado en cuatro categorías: información del problema, información del algoritmo, tiempo de medición, y función de agregación. Este marco de trabajo fue aplicado para organizar los progresos actuales, y en el diseño de las medidas de rendimiento en el framework de DynOptLab. En relación a la interfaz gráfica se diseñó de manera que el investigador pudiera establecer de manera intuitiva, los factores objeto de estudio (problemas y algoritmos) y las variables de respuesta (medidas de rendimiento). Asimismo, la interfaz gráfica de DynOptLab permite ejecutar los experimentos de manera paralela, y analizar los resultados a partir de: la visualización directa

de los resultados, comparaciones entre algoritmos, y la aplicación pruebas estadísticas no paramétricas. Con el objetivo de verificar las funcionalidades de la herramienta propuesta, se diseñó un caso de estudio basado en una investigación de la literatura. Los resultados de esta prueba fueron satisfactorios y permiten concluir que la herramienta propuesta permite una gestión eficiente de la experimentación en ambientes dinámicos.

Con respecto al objetivo 2, se desarrolló una revisión a fondo de las propuestas actuales que aplican el paradigma PSO a problemas dinámicos de optimización. A partir de este estudio se pudo comprobar que las variantes multi-poblaciones (multi-enjambre) se encuentran entre las más efectivas para lidiar con problemas dinámicos complejos. En ese sentido, se destaca el algoritmo mQSO, el cual incluye varias poblaciones y una estrategia para la generación de diversidad durante la ejecución denominada enfoque quantum. No obstante los éxitos demostrados por este algoritmo en escenarios dinámicos, en la presente investigación se propusieron dos estrategias adicionales para mejorar su rendimiento. Por un lado, se diseñó una estrategia de diversidad similar al enfoque quantum pero aplicada después de la ocurrencia de un cambio en el ambiente, y por otro, una regla de control difusa para decidir en tiempo de ejecución los enjambres que pueden contribuir al proceso de optimización. Los resultados de los experimentos computacionales demostraron los beneficios de las propuestas. En particular, la estrategia de diversidad resultó efectiva en problemas con funciones pico multimodales, mientras que la regla de control difusa en problemas con funciones pico unimodales.

Para cumplir con el objetivo 3 de la investigación, se desarrolló un estudio en profundidad de los progresos actuales relacionados con la aplicación de la auto-adaptación en ambientes dinámicos. A partir de esta revisión se organizaron los trabajos existentes de acuerdo al tipo y alcance de la investigación: trabajos básicos, trabajos teóricos, y trabajos avanzados. En el primer caso se encuentran principalmente las primeros trabajos, y se caracterizan por el estudio de paradigmas auto-adaptativos en ambientes dinámicos. En el segundo caso aparecen los trabajos que analizan teóricamente el comportamiento de modelos auto-adaptativos. Finalmente, en el tercer grupo se encuentran los trabajos más recientes, caracterizados por la inclusión de enfoques de adaptación dinámica en conjunto con algún modelo auto-adaptativo. De manera general, se observó que existen conclusiones divididas en relación al éxito o no de la auto-adaptación en ambientes dinámicos. Asimismo, que son prácticamente inexistentes las investigaciones que apliquen modelos auto-adaptativos directamente a los enfoques de adaptación dinámica. En este contexto, se propuso una estrategia auto-adaptativa para la generación de diversidad durante la ejecución, con el objetivo de investigar si en este caso, la auto-adaptación tiene sentido en ambientes dinámicos. En esencia, la estrategia permite la auto-adaptación de uno de los parámetros del enfoque quantum de mQSO, y fue aplicada en algoritmos multi-poblacionales basados en

el paradigma Evolución Diferencial (DE). Los experimentos computacionales desarrollados permitieron identificar: 1) la mejor combinación de parámetros que definen a la estrategia, y 2) el alto nivel de competitividad de los algoritmos propuestos (comparables y en ocasiones superiores a los de literatura). A partir de estos resultados, la principal conclusión a la que se arribó fue que la auto-adaptación si tiene sentido en ambientes dinámicos.

No obstante los resultados del objetivo anterior, resultó de interés analizar con más profundidad el rol de la auto-adaptación en ambientes dinámicos. Este aspecto fue desarrollado a través del objetivo 4. En este sentido, se diseñaron experimentos computacionales orientados a estudiar el rendimiento de 8 algoritmos multi-poblacionales con paradigmas computacionales basados en DE. Estos algoritmos se caracterizaron por la inclusión o no de distintos modelos auto-adaptativos tanto a nivel de paradigma como de enfoque de adaptación dinámica. Los resultados indicaron que la auto-adaptación posee un mayor impacto cuando es aplicada al enfoque de adaptación dinámica considerado. Asimismo, los algoritmos con presencia de modelos auto-adaptativos en ambos niveles resultaron superiores a los que solo aplican auto-adaptación solo a nivel de paradigma. Adicionalmente, con el objetivo de entender mucho mejor estas conclusiones, se propuso una medida que permite cuantificar el grado de adaptabilidad de los algoritmos. Finalmente, la competitividad de los mejores algoritmos estudiados fue analizada en relación a otros de la literatura. En relación a este aspecto, se pudo concluir que nuestros algoritmos alcanzan en general, niveles similares de rendimiento con respecto al resto.

Parte I

Preliminares

1 Introducción

Varios son los fenómenos de la vida real que pueden analizarse a través de un modelo matemático: el crecimiento de la población mundial, la selección de la mejor ruta en una ciudad, el comportamiento de las ventas en un período, etc. La mayoría de estos fenómenos, a través de los modelos apropiados, se convierten en problemas matemáticos que tienen que ser resueltos, dando lugar a soluciones que provean la ganancia máxima, el costo mínimo, la satisfacción de los usuarios, entre otros. Tales problemas se conocen como problemas de optimización y durante décadas han sido ampliamente tratados con resultados favorables, dando lugar al nacimiento de nuevas ramas dentro de las matemáticas como la Investigación Operativa.

En general, los modelos de optimización están compuestos por dos elementos fundamentales: el conjunto de posibles soluciones (espacio de búsqueda) donde las variables definidas tomarán valores específicos, y una función objetivo o de aptitud (FO), que asigna un valor numérico a cada combinación de valores en las variables (solución candidata). Un tercer elemento que es habitual son las denominadas restricciones, comúnmente éstas imponen límites al espacio de búsqueda o expresan relaciones entre las variables del problema. Resolver un problema de optimización (u optimizar) significa entonces, encontrar la solución que proporcione el mejor valor en la FO (solución óptima), satisfaciendo las restricciones. Cuando se busca el menor valor posible de la FO, entonces este recibe el nombre de mínimo global, y máximo global si por el contrario, la tarea es encontrar al mayor.

Aunque la mayor parte de los desarrollos obtenidos en el ámbito de la optimización matemática versan sobre problemas estacionarios, es decir, modelos cuyas variables, relaciones y objetivos permanecen constantes en el tiempo, existe un tipo especial de problema que presenta variabilidad en los elementos del modelo (ej. la función objetivo, conjunto de restricciones, dimensiones del espacio de búsqueda, etc.). A estos problemas se les conoce como problemas dinámicos de optimización (PDO). Los PDO se encuentran a

menudo en la vida, ejemplo de ello son la distribución óptima de recursos con clientes que aparecen/desaparecen con el tiempo, la planificación de tareas/eventos que varían con el tiempo, entre otros.

En las dos últimas décadas, estos problemas han sido tratados eficientemente mediante métodos de optimización conocidos como *meta-heurísticas* Melián et al (2003). Las meta-heurísticas son técnicas computacionales aproximadas pertenecientes al campo de la *Soft Computing* (Zadeh, 1994; Verdegay et al, 2008) (también conocida como Computación Inteligente). El éxito de tales técnicas en ambientes dinámicos se debe en gran medida a dos cuestiones fundamentales: la capacidad de las meta-heurísticas para optimizar escenarios complejos, y las adaptaciones incorporadas para lidiar con la dinámica del problema.

En particular, las meta-heurísticas *bioinspiradas*, tales como las pertenecientes a las áreas de los *Algoritmos Evolutivos* (EA), *Inteligencia de Enjambres*, entre otras, han resultado ser especialmente eficientes en este contexto. Sin embargo, al igual que otros tipos de meta-heurísticas, han requerido de adaptaciones adicionales para lidiar específicamente con lo que se conoce en ambientes dinámicos como el *problema de la convergencia* (Jin y Branke, 2005). En esencia, este fenómeno ocurre cuando el algoritmo converge en un óptimo obsoleto del problema, y es incapaz de seguir al nuevo óptimo tras la ocurrencia de un cambio en el ambiente. Por lo general, esta incapacidad del algoritmo se debe a una pérdida de significativa de la diversidad de la población (conjunto de de soluciones), o de algunos parámetros relacionados con su comportamiento durante la ejecución. De acuerdo con Jin y Branke (2005), los enfoques de adaptación existentes para lidiar con este tipo de problema, se clasifican en cuatro grupos: 1) diversidad durante la ejecución, 2) diversidad después de los cambios, 3) enfoques basados en memorias, y 4) enfoques multipoblacionales. Más recientemente, Nguyena et al (2012) proponen que otra variante podría ser aprovechar la *auto-adaptación* (Bäck y Schwefel, 1993; Eiben et al, 2007) (como técnica de control de parámetros), con la intención de que el algoritmo pudiera aprender de manera implícita la dinámica del problema.

No obstante los importantes progresos alcanzados en la actualidad dentro de este campo de investigación, es posible identificar las siguientes dificultades:

- No existe una herramienta que facilite el trabajo experimental del investigador, quién no solo debe preocuparse por la implementación de los problemas, algoritmos y medidas, sino que además debe prestar atención a la configuración de los experimentos y al posterior procesamiento de los resultados.
- Los principales enfoques de adaptación existentes, tales como los multipoblacionales, poseen un carácter invariable durante la ejecución del algoritmo, por lo que no aprovechan el estado actual de la búsqueda y la dinámica del problema. En conse-

cuencia, la *robustez* de los algoritmos queda afectada significativamente, cuando son aplicados a problemas con características diferentes.

- Dentro de las técnicas de control de parámetros, la auto-adaptación dota de un comportamiento inteligente al algoritmo en tiempo de ejecución. Sin embargo, aunque esta técnica de control ha sido extensivamente estudiada en problemas estacionarios, muy poco desarrollo existe en relación a su aplicación a escenarios dinámicos.

Con la intención de solucionar estos problemas aún abiertos, la presente investigación se plantea los siguientes objetivos:

1. Proponer un herramienta para gestionar la experimentación en ambientes dinámicos, a partir de una organización adecuada de las medidas de rendimiento en ambientes dinámicos, y la inclusión de pruebas estadísticas no paramétricas.
2. Proponer estrategias adaptativas que mejoren el rendimiento de enfoques PSO multi-enjambres en ambientes dinámicos.
3. Investigar si la auto-adaptación, como técnica de control de parámetros, tiene sentido en ambientes dinámicos.
4. Analizar el rol de la auto-adaptación en ambientes dinámicos cuando es aplicada en distintas partes del algoritmo, tanto a nivel de paradigma, como a nivel de enfoque de adaptación.

Las tareas desarrolladas para cumplir con estos objetivos se encuentran organizadas en cinco capítulos, los cuales se describen a continuación.

El Capítulo 2 brinda los fundamentos teóricos y metodológicos de la investigación. Inicialmente, se exponen los fundamentos relacionados con el campo de la *Soft Computing*, y dentro de este, el papel que juegan las meta-heurísticas. Se hace especial énfasis en los paradigmas PSO y Evolución diferencial (DE), por ser estos los empleados en las contribuciones de la tesis. A esto le sigue una amplia sección es dedicada a las técnicas de configuración de parámetros. Finalmente, los progresos actuales sobre la optimización en ambientes dinámicos son expuestos a través de problemas, métodos y medidas de rendimiento.

A partir del Capítulo 3 se exponen las contribuciones de la presente investigación. En particular, este capítulo describe principalmente a la herramienta propuesta DynOptLab, la cual está orientada a la gestión de experimentos en ambientes dinámicos. El capítulo comienza con la exposición de un marco de trabajo propuesto para organizar convenientemente las medidas de rendimiento en escenarios dinámicos. Este marco fue aplicado con éxito en la organización de las propuestas actuales, y posteriormente en el diseño de clases de DynOptLab. Más adelante, se revisan las metodologías estadísticas que existen para analizar los resultados en ambientes dinámicos y se establece la utilizada en el resto de la

tesis. En este apartado, se propone una nueva gráfica para visualizar de manera compacta e intuitiva, los resultados de las pruebas no parámetricas de Friedman y las *post hoc*, como la de Holm. En especial, la prueba de Friedman e Iman-Davenport fueron incluídas en el análisis estadístico que provee DynOptLab. Posteriormente, se describe en detalle la herramienta DynOptLab, sus principales funcionalidades y pantallas. El Capítulo termina con un caso de estudio de un experimento de la literatura, el cual fue desarrollado en DynOptLab.

El Capítulo 4 responde al objetivo 3 de la presente investigación. En este, se revisan los principales progresos relacionados con la optimización en ambientes dinámicos mediante el paradigma computacional PSO, poniéndose énfasis en el enfoque multipoblacional mQSO propuesto por Blackwell y Branke (2006). A raíz de esta exposición se identifican algunos de las posibles mejoras a este algoritmo y se proponen dos estrategias. La primera relacionada con la generación de diversidad después de los cambios, mientras que la segunda permite el control en tiempo de ejecución de los enjambres que intervienen en el proceso de optimización. El impacto de ambas propuestas es analizado a través de múltiples experimentos computacionales.

Por su parte, el Capítulo 5 se encuentra relacionado con los objetivos 4 y 5. Aquí, se realiza primeramente una revisión profunda sobre las investigaciones actuales que aplican algún tipo de modelo de auto-adaptación en ambientes dinámicos. Los trabajos revisados son organizados convenientemente con el objetivo de identificar los progresos alcanzados y las tendencias existentes en este campo de investigación. En este contexto, se propone una estrategia auto-adaptativa para controlar la diversidad de la población durante la ejecución. En esta ocasión, la estrategia propuesta es empleada en el enfoque multipoblacional similar al del Capítulo 4, pero con el paradigma DE en lugar de PSO. Con la intención de investigar detalladamente dicha estrategia, se desarrollaron varios experimentos computacionales, incluyendo comparaciones con algoritmos de la literatura.

El Capítulo 5 prosigue con el objetivo 5. En este caso, se diseñaron varios experimentos orientados a estudiar el rol de la auto-adaptación en ambientes dinámicos. En este diseño se tuvieron en cuenta un total de 8 algoritmos obtenidos a partir de la presencia o no de modelos auto-adaptativos distintos, aplicados en diferentes partes del algoritmo: a nivel de paradigma y a nivel de enfoque de adaptación. Los resultados fueron analizados estadísticamente y los mejores algoritmos fueron comparados con otros similares de la literatura.

Finalmente, en el Capítulo 6 se dan las conclusiones generales de la investigación, incluyendo los trabajos futuros.

2 Soft Computing y problemas dinámicos de optimización

Este capítulo tiene como propósito introducir los aspectos teóricos relacionados con la investigación. Primeramente, se describirá qué es Soft Computing a través de sus principales componentes. En este contexto, uno de los componentes fundamentales lo constituyen las meta-heurísticas, las cuales son descritas con profundidad. Se han expuesto además, las principales características de las técnicas de configuración de parámetros en el ámbito de la optimización evolutiva. Más adelante, se han descrito los fundamentos de la optimización en ambientes dinámicos mediante técnicas meta-heurísticas, a través de los principales enfoques de adaptación, problemas artificiales y medidas de rendimiento. Finalmente, el capítulo concluye con un resumen de los aspectos teóricos tratados.

2.1 Soft Computing

Concretamente, el principal objetivo de la Inteligencia Artificial (IA) es el desarrollo de sistemas capaces de aprender y comportarse inteligentemente. Desde los primeros tiempos, pioneros como von Neumann soñaban con sistemas con más de un procesador, incluso dotados con la capacidad de auto-reproducirse. En la historia de la IA varios enfoques han sido propuestos con el propósito de alcanzar este y otros objetivos igual de ambicioso. La mayoría de estos métodos pueden clasificarse en dos clases generales: enfoques *simbólicos* y *subsimbólicos*. Este último y también más reciente, es conocido por diversos nombres: Computación Inteligente (*Computational Intelligence*, CI) o natural, o intuitivamente *Soft computing* (SC).

El término Soft Computing, acuñado por primera vez por Zadeh (1994), se utiliza con toda intencionalidad para hacer notar la diferencia con lo que se conoce como *hard computing*, característica de las técnicas clásicas (simbólicas) de la IA. Específicamente,

Zadeh establece que:

> "*...soft computing no es un cuerpo homogéneo de conceptos y técnicas. Mas bien es una mezcla de distintos métodos que de una forma u otra cooperan desde sus fundamentos. En este sentido, el principal objetivo de la Soft Computing es aprovechar la tolerancia que conllevan la imprecisión y la incertidumbre, para conseguir manejabilidad, robustez y soluciones de bajo costo. Los principales ingredientes de la Soft Computing son la Lógica o Fuzzy, la Neuro-computación y el Razonamiento Probabilístico, incluyendo este ultimo a los Algoritmos Genéticos, las Redes de Creencia, los Sistemas Caóticos y algunas partes de la Teoría de Aprendizaje. En esa asociación de Lógica Fuzzy, Neurocomputación y Razonamiento Probabilístico, la Lógica Fuzzy se ocupa principalmente de la imprecisión y el Razonamiento Aproximado; la Neurocumputación del aprendizaje, y el Razonamiento Probabilístico de la incertidumbre y la propagación de las creencias.*".

De esta forma, Zadeh define Soft Computing mediante extensiones, esto es, mediante las diferentes técnicas y conceptos desarrollados para tratar problemas que incluyen impresición, incertidumbre o simplemente difíciles de categorizar. Sin embargo, como suele pasar en la mayoría de las ciencias, la definición de soft computing ha evolucionado desde el trabajo pionero de Zadeh, sobre todo por la madurez que han alcanzado precisamente esos conceptos y técnicas que la componen. En ese sentido, más recientemente Verdegay et al (2008) establecen que:

> "*Se trata de considerarla como antítesis de lo que se denomina Hard Computing, de manera que podría verse la Soft Computing como un conjunto de técnicas y métodos que permitan tratar las situaciones prácticas reales de la misma forma que suelen hacerlo los seres humanos, es decir, en base a inteligencia, sentido común, consideración de analogías, aproximaciones, etc. En este sentido Soft Computing es una familia de métodos de resolución de problemas cuyos primeros miembros serían el Razonamiento Aproximado y los Métodos de Aproximación Funcional y de Optimización, incluyendo los de búsqueda.* "

Desde este punto de vista queda claro que la Soft Computing es situada como base teórica del área de los Sistemas Inteligentes, y que la diferencia entre la Inteligencia Artificial clásica, y la de los Sistemas Inteligentes, es que la primera se apoya en la denominada Hard Computing, mientras que la segunda lo hace precisamente en la Soft Computing. A

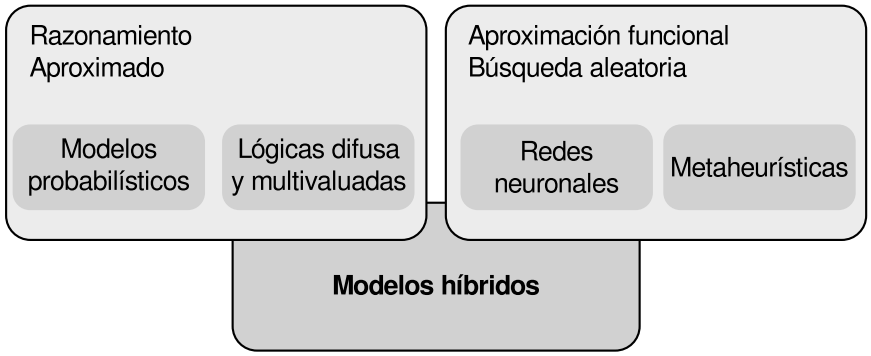

Figura 2.1 Soft Computing como mezcla de métodos y técnicas interrelacionados.

modo de ilustrar como la Soft Computing integra los mencionados métodos y técnicas véase la Figura 2.1.

Nótese en este sentido que uno de los constituyentes dentro de la rama de la Aproximación funcional y búsqueda aleatoria se encuentran las *meta-heurísticas*, las cuales representan métodos de optimización y serán abordadas más adelante.

2.2 Metaheurísticas

Antes de describir las principales características de las meta-heurísticas, conviene definir antes que son los problemas de optimización. En ese sentido, es importante recordar que las características del contexto o fenómeno que se pretende optimizar definen el modelo y por tanto, el tipo de problema de optimización. Por ejemplo, si al resolver el problema se deben tener en cuenta varios objetivos al mismo tiempo (ej. la mayor ganancia en el menor tiempo), entonces se estará en presencia de un problema de optimización *multiobjetivo* o *multicriterio*. Por otro lado, si el conjunto de soluciones debe satisfacer un conjunto de restricciones, entonces a este tipo de problema suele denominarse problema de optimización con restricciones.

Sin embargo, a menos que se establezca lo contrario, en el resto de la presente investigación se tratarán problemas de un solo objetivo, sin restricciones y con espacio de búsqueda continuo, esto es, las variables de decisión toman valores reales ($\mathbf{x} \in \mathbb{R}^n$). Dado que los problemas dinámicos constituyen una generalización de los problemas de optimización estacionarios, en lo que sigue se dará una definición formal de estos últimos para una mejor comprensión de los primeros.

Formalmente un problema de optimización estacionario se puede definir de la siguiente forma:

Definición 2.2.1 (Problema de optimización estacionario (P_E)). *Sea el espacio de búsqueda $\Omega \subseteq \mathbb{R}^D$, la función objetivo $f : \Omega \to \mathbb{R}$, y la relación de comparación $\succeq \in \{\leq, \geq\}$, encontrar el conjunto $\mathscr{X}$ de soluciones óptimas definido como:*

$$\mathscr{X} = \left\{ x^* \in \Omega \mid f(x^*) \succeq f(x),\ \forall x \in \Omega \right\} \tag{2.1}$$

Cuando $\succeq$ está representada por la relación $\leq$ entonces el problema es de minimización, en caso contrario de maximización.

Desde el punto de vista clásico, los problemas de optimización han sido tratados desde la antigüedad. Posteriormente, a partir del descubrimiento del Cálculo en el siglo XVIII por Newton y Leibnitz se inició un fructífero desarrollo que ha durado hasta nuestros días. Estos métodos, denominados clásicos, se clasifican principalmente en tres grupos *directos*, de *gradiente*, y de búsqueda Hessiana (*Hessian search methods*).

Los métodos directos determinan la dirección de la búsqueda sin el empleo de las derivadas (Schwefel, 1995), de manera similar a como lo hacen las *meta-heurísticas*, como se verá más adelante. Dentro de esta gran familia de métodos, se destacan los basados en *patrones* (Davidon, 1991) que realizan la búsqueda a través de la generación de puntos pertenecientes a una figura geométrica. En este caso se encuentra en el método *Simplex*[1] de Nelder y Mead (1964), el cual supone que el gradiente de la función puede ser estimado a través de un conjunto de $D+1$ puntos, siendo D la dimensión del espacio de búsqueda. Otros métodos directos clásicos son los de Rosenbrock (1960) y Powell (1964) que se caracterizan por recopilar información sobre la curvatura de la función objetivo durante la ejecución. Sobre este tipo de métodos el lector puede encontrar más información en la excelente revisión de Lewis et al (2000).

Si las derivadas de la función objetivo pueden ser calculadas, entonces los métodos basados en el gradiente y en la matriz Hessiana pueden ser aplicados. Los primeros tienen en cuenta la primera derivada, mientras que los segundos también la segunda derivada. Ejemplos de estos son los métodos de Newton y quasi-Newton (Broyden, 1970). Estas técnicas buscan puntos estacionarios de la función donde el gradiente se anula. En particular, los denominados quasi-Newton estiman la matriz Hessiana analizando los sucesivos vectores del gradiente. En la práctica, la primera derivada no siempre está disponible, y por tanto las de orden superior tampoco. Por esta razón, y por determinados aspectos de los problemas reales, los métodos clásicos resultan si no imposibles al menos difíciles de aplicar en estos escenarios. De ahí el éxito de los procedimientos meta-heurísticos, como se

[1] Aunque puede causar confusión con el algoritmo *Simplex* de G. Dantzig, de igual nombre, en este caso se trata de un método de búsqueda directa diferente, orientado a la solución de problemas de optimización no lineales.

verá más adelante.

Aunque la optimización de los problemas de optimización estacionarios no constituye directamente el objeto de estudio de la presente investigación, el lector puede encontrar más información acerca de posibles clasificaciones, métodos y herramientas computacionales en las siguientes referencias: (Beveridge y Schechter, 1977) para una visión clásica y (Griva et al, 2009; Rao, 2009) desde un punto de vista más actual.

2.2.1 Principales características

A diferencia de los métodos clásicos descritos brevemente en la sección anterior, las meta-heurísticas permiten tratar instancias de problemas de gran tamaño a través de la búsqueda de soluciones satisfactorias en un tiempo razonable (Glover, 1986). El término satisfactoria significa que al aplicar una meta-heurística no hay garantía de que se encuentre la solución óptima global del problema. Informalmente, las meta-heurísticas son "estrategias inteligentes para diseñar o mejorar procedimientos heurísticos generales con un alto rendimiento"(Melián et al, 2003).

En general, el uso cada vez más frecuente de este tipo de método en disímiles escenarios demuestra su eficiencia y efectividad en la solución de problemas grandes y complejos. Ejemplos de estos escenarios son (ghazali Talbi, 2009):

- Diseño ingenieril, optimización topológica y estructural en la electrónica, dinámica de fluidos, telecomunicaciones, automoción, aerodinámica, y robótica.
- Aprendizaje automatizado y minería de datos en bioinformática y finanzas.
- Modelización de sistemas, simulación e identificación en química, física, y biología; control, señales y procesamiento de imágenes.
- Planificación en problemas de rutas, problemas de producción, logística y transportación, gestión de cadenas de suministros, entre otros.

El creciente interés que ha despertado el mundo de las meta-heurísticas se puede comprobar en el aumento de eventos científicos internacionales (congresos, conferencias, sesiones) que se dedican anualmente a este tema.

A modo de resumen, la Figura 2.2 muestra la genealogía de las diversas meta-heurísticas existentes. Es importante aclarar que el concepto de heurística en la resolución de problemas de optimización fue introducido por primera vez en Polya (1945). En ese sentido, el método *Simplex*, creado por Dantzig en 1947, puede ser visto como algoritmo de búsqueda local (LS en la Figura 2.2)para problemas de programación lineal (Dantzig, 1951, 1963). Asimismo, Edmonds (1971) fue el primero en proponer heurísticas de tipo *greedy* en e área de la optimización combinatoria. Las siglas del resto de los métodos poseen el significado en

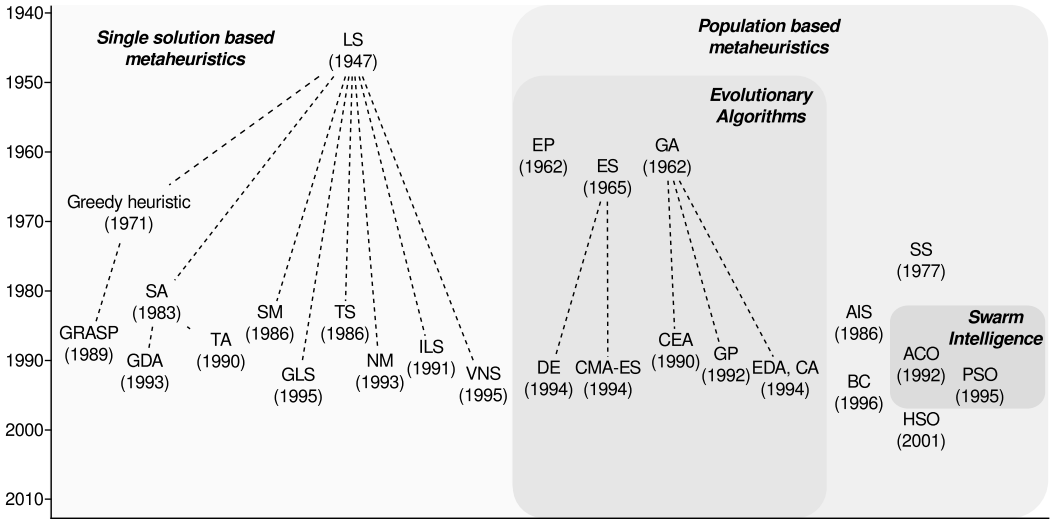

Figura 2.2 Genealogía de las meta-heurísticas desde sus inicios hasta la actualidad.

inglés que se muestra a continuación, en orden cronológico y agrupadas en dos grandes familias: meta-heurísticas de una sola solución, y poblacionales.

Metaheurísticas basadas en un sola solución:

- **SA**. Simulated Annealing (Kirkpatrick et al, 1983).
- **TS**. Tabu search (Glover, 1986).
- **SM**. Smoothing method (Glover y McMillan, 1986).
- **GRASP**. Greedy adaptive search procedure (Feo y Resende, 1989).
- **TA**. Threshold accepting (Duech y Scheuer, 1990).
- **ILS**. Iiterated local search (Martin et al, 1991).
- **GDA**. Great deluge (Dueck, 1993).
- **NM**. Noisy method (Charon y Hudry, 1993).
- **GLS**. Guided local search (Voudouris y Tsang, 1995).
- **VNS**. Variable neighborhood search (Mladenovic, 1995).

Metaheurísticas poblacionales:

- **EP**. Evolutive programming (Fogel, 1962).
- **GA**. Genetic algorithms (Holland, 1962).
- **ES**. Evolution strategy (Rechenberg, 1965).
- **SS**. Scatter search (Glover, 1977).
- **AIS**. Immune artificial system (Farmer et al, 1986).
- **CEA**. Coevolutionary algorithms (Hillis, 1990).

- **GP**. Genetic programming (Koza, 1992).
- **ACO**. Ant colony optimization (Dorigo, 1992).
- **DE**. Differential evolution (Price, 1994).
- **EDA**. Estimation of distribution algorithms.
- **CA**. Cultural algorithms (Reynolds, 1994).
- **PSO**. Particle swarm optimization (Kennedy y Eberhart, 1995).
- **BC**. Bee colony (Yonezawa y Kikuchi, 1996)
- **CMA-ES**. Covariance matrix evolutionary strategy (Hansen y Ostermeier, 1996).
- **HSO**. Harmony search optimization (Geem et al, 2001).

Al diseñar meta-heurísticas, siempre están presentes dos criterios opuestos que deben ser tomados en cuente: exploración del espacio de búsqueda (diversidad) y explotación de las mejores soluciones encontradas (intensificación). Las regiones de búsqueda son determinadas por las *buenas* soluciones obtenidas. Durante la fase de explotación estas regiones de búsqueda son exploradas con más profundidad con el objetivo de encontrar mejores soluciones. Por su parte, la exploración se dedica a visitar regiones no exploradas del espacio de búsqueda, lo cual es particularmente útil en problemas multimodales. En ese sentido, quizás los exponentes más extremos en cuanto a exploración y explotación sean la búsqueda aleatoria (*random search*) y la búsqueda local de mejoramiento iterativo (*iterative improvement local search*). En la primera, el algoritmo genera una solución aleatoria en el espacio de búsqueda sin emplear ninguna información de iteraciones anteriores. Muy diferente ocurre con la búsqueda local, en la que en cada iteración el algoritmo selecciona la mejor solución vecina que mejore a la actual.

En sentido general, se pueden utilizar diversos criterios para clasificar a las meta-heurísticas:

Inspiradas en la naturaleza vs. no inspiradas: Muchas meta-heurísticas están inspiradas en procesos naturales, por ejemplo los algoritmos evolutivos en la biología, PSO y ACO en la inteligencia de enjambres de seres sociales; mientras que *simulated annealing* por ejemplo lo está en procesos físicos.

Uso de memorias vs. sin memoria : Algunas meta-heurísticas no emplean memorias; esto es, ninguna información adicional es extraída dinámicamente durante la ejecución. Algunos representantes de esta clase son la búsqueda local, GRASP y *simulated annealing*. Mientras que otras meta-heurísticas usan memorias que explotan convenientemente durante el proceso de búsqueda. Por ejemplo, las memorias de corto y largo plazo en la búsqueda tabú.

Determinísticas versus estocásticas: Una meta-heurística determinística resuelve un problema de optimización tomando decisiones determinísticas (ej., búsqueda local, la búsqueda tabú). Sin embargo, en las meta-heurísticas estocásticas, algunas reglas aleatorias son aplicadas durante la ejecución (ej., *simulated annealing*, algoritmos evolutivos). Teóricamente, si los algoritmos determinísticos usan la misma solución inicial entonces obtienen la misma solución final, mientras que en los estocásticos no necesariamente ocurre. Esta característica es de vital importancia a la hora de evaluar el rendimiento de los meta-heurísticas.

Poblacionales vs. una sola solución: Los algoritmos basados en una sola solución (ej. grupo *Single-solution based meta-heuristics*, Figura 2.2) manipulan y transforman una sola solución durante la búsqueda, mientras que en los poblacionales (ej. grupo *Population based meta-heuristics*, Figura 2.2) todo un conjunto de soluciones evolucionan en cada iteración del algoritmo. Sin embargo, estas dos familias tienen características complementarias: las basadas en una sola solución están orientadas a la *explotación*, pues tienen el poder de intensificar la búsqueda en regiones locales. Por su parte las poblacionales están orientadas a la exploración, ya que permiten una mejor diversificación en todo el espacio de búsqueda. Precisamente en esta investigación las meta-heurísticas empleadas son de este último tipo, ya que como se verá más adelante, una de las dificultades más importantes en ambientes dinámicos es el mantenimiento de la diversidad.

Iterativa vs. *greedy*: En los algoritmos iterativos se comienza con una solución completa (o población de soluciones completas) que se transforman en cada iteración mediante operadores de búsqueda. Por su parte, los algoritmos greedy comienzan con una solución vacía, y en cada iteración se le asigna a esta una variable de decisión del problema hasta que esté completa. Sin embargo, como se puede apreciar en la Figura 2.2 la mayoría de las meta-heurísticas son iterativas.

2.2.2 Teoremas *No free lunch*

Uno de los aspectos que siempre preocupó a los investigadores y diseñadores de meta-heurísticas fue el asunto relacionado con la posibilidad de crear un algoritmo ideal, esto es, que tuviese un rendimiento superior a cualquier otro en todos los escenarios posibles. Intuitivamente parece imposible tal tarea, algo que fue dejándose ver en cada propuesta, congreso, o competición de algoritmos antes de 1995. En este mismo año, Wolpert y Macready (1995) sorprendieron a la comunidad científica con sus teoremas denominados

No free lunch (NFL), trabajo que fue extendido posteriormente en (Wolpert y Macready, 1997).

En esencia, los teoremas NFL prueban que, bajo ciertas condiciones, ningún algoritmo de optimización es superior a cualquier otro cuando ambos son aplicados a todos los posibles problemas de optimización, esto es, si el algoritmo A es mejor que B en un problema determinado, entonces existe otro problema donde B es mejor que A. En resumen, según Wolpert y Macready, ninguna meta-heurística puede superar uniformemente a cualquier otra, por lo que la pregunta relacionada con la superioridad de un determinado algoritmo solo tiene sentido cuando es aplicado a una clase específica de problemas.

En la actualidad, existen aún detractores de esta teoría, mientras que otros investigadores han profundizado en el tema. El lector interesado puede revisar por ejemplo los siguientes trabajos (Köppen et al, 2001; Igel y Toussaint, 2003; Oltean, 2004).

2.2.3 Algoritmos evolutivos

Los algoritmos que pertenecen a esta familia no solo comparten la característica de estar inspirados en la biología (específicamente en los principios evolutivos de Darwing), sino que además comparten un esquema general como se muestra en el Algoritmo 2.1. La diferencia entre los distintos algoritmos evolutivos (y sus extensiones) radica en como representan a los individuos (soluciones) y en como definen sus operadores evolutivos: variación (cruzamiento y mutación), y selección.

```
   // Inicialización
1  t ← 0;
2  Inicializar 𝒫(t) uniformemente en el espacio de búsqueda, Ω;
   // Ciclo evolutivo
3  mientras no condición de parada hacer
4  |  Variar 𝒫(t) → 𝒫'(t);
5  |  Evaluar 𝒫'(t);
6  |  Seleccionar 𝒫(t+1) a partir de 𝒫'(t);
7  |  t = t+1;
8  fin
```

Algoritmo 2.1: Esquema general de los algoritmos evolutivos.

Algoritmos genéticos (GA). Los algoritmos genéticos fueron desarrollados por J. Holland en los 70s (Universidad de Michigan, USA) con el objetivo de comprender los procesos adaptativos de los sistemas naturales (Holland, 1962). A partir de ahí, este paradigma fue aplicado en la optimización y en aprendizaje automatizado en los 80s, convirtiéndose así en uno de los algoritmos evolutivos más populares en la actualidad. Tradicionalmente, los GAs

están asociados con el uso de representación binaria, pero en la actualidad es posible encontrar GAs que usan otros tipos de representaciones (ej. más explícitas). Un GA comúnmente aplica el operador de cruzamiento a dos soluciones prometedoras, y el operador de mutación para modificar aleatoriamente las componentes del individuo obtenido. Asimismo, los GAs emplean una selección probabilística, la cual originalmente es la conocida selección proporcional. El reemplazamiento (selección de sobrevivientes) es generacional, esto es, los padres son reemplazados en cada iteración del algoritmo por los nuevos descendientes. Generalmente, el operador de cruzamiento es de tipo *n-punto* o *uniforme*, mientras que la mutación es de tipo *bit flipping*. Dos tasas de probabilidad (*pm* y *pc*), predefinidas antes de la ejecución del algoritmo, controlan la aplicación de los operadores de mutación y cruzamiento respectivamente.

Estrategias evolutivas (ES). Las estrategias evolutivas son otra subclase de los algoritmos evolutivos como los GAs. Este paradigma fue originalmente desarrollado por Rechenberg y Schewefel en 1964 en la Universidad Técnica de Berlin (Rechenberg, 1965; Schwefel, 1981). Las ESs son generalmente aplicadas a problemas de optimización continuos donde la representación de las soluciones viene dada por vectores de valores reales. Las primeras aplicaciones incluyeron la optimización de formas de parámetros reales. Este paradigma se caracteriza por emplear un reemplazamiento elitista y un operador de mutación basado en una distribución normal (Gaussiana). Por su parte, el cruzamiento es poco empleado. En ES existe una distinción importante entre la población de padres (de tamaño μ) y la población de hijos (de tamaño λ), ya que $\lambda \geq \mu$. Asimismo, un individuo está compuesto no solo por el correspondiente vector solución sino también por otros parámetros que guían la búsqueda. De esta manera las ES experimentan un tipo de auto-adaptación mediante la evolución de las soluciones y de los parámetros estratégicos como son los factores de mutación σ. El operador de selección es determinístico y está basado en el fitness de los individuos. La recombinación es discreta (similar al cruzamiento uniforme de los GAs) o intermediaria (cruzamiento aritmético). La principal ventaja de las ES es su eficiencia en términos de complejidad computacional. Desde el punto de vista teórico las ES han sido ampliamente estudiadas.

Programación evolutiva (EP). La Programación evolutiva se caracteriza por la mutación y por no usar recombinación. Tradicionalmente, las EP han sido desarrolladas para evolucionar autómatas de estado finito con la intención de resolver así problemas de predicción de series temporales, y de manera más general para resolver problemas del aprendizaje automatizado. Los EPs más contemporáneos han sido aplicados últimamente en la optimización de problemas continuos. Similar a las ES, EPs emplea mutación de distribución

normal y auto-adaptación en los parámetros que guían la búsqueda. La selección de los padres es determinística, mientras que la selección de los sobrevivientes (reemplazamiento) es probabilístico y está basado en torneos estocásticos $(\mu + \mu)$. El framework de las EP es menos empleado que el de otros paradigmas debido a su similitud con las ES.

Programación genética (GP). La programación genética es un enfoque evolutivo más reciente, el cual extiende el modelo genérico de aprendizaje al espacio de programas (Koza, 1992). Su principal diferencia, en relación al resto de los paradigmas evolutivos, es que los individuos en sí mismos son programas (representaciones no lineales basadas en árboles) y no cadenas de longitud fija a partir de un alfabeto limitado de símbolos (representación lineal). GP es una forma de inducción de programa que permite la generación automática de programas con el objetivo de resolver un determinado problema. En general, la selección de los padres es proporcional al fitness, mientras que la selección de los hijos es llevada a cabo mediante un reemplazamiento generacional. El operador de cruzamiento está basado en el intercambio de sub-árboles y la mutación se basa en un cambio aleatorio del árbol en cuestión. Una de las principales dificultades en la GP es el crecimiento incontrolado de los árboles; fenómeno llamado comúnmente *bloat*. Originalmente, J. Koza empleó expresiones en lenguaje Lisp para los programas. Más generalmente, las representaciones de las soluciones (programas) son S-expresiones (o árboles *parse*) donde las hojas son terminales y los nodos internos son operadores (funciones). La definición de las hojas y de los operadores depende del dominio del problema en cuestión. Normalmente, GP necesita de una población grande (e.g., miles de individuos), siendo por lo general costosos en términos computacionales. Desde el punto de vista teórico, GP ha sido menos desarrollada que las estrategias evolutivas y los algoritmos genéticos. En la actualidad las principales aplicaciones de los algoritmos basados en GP incluyen áreas como el aprendizaje automatizado y la minería de datos, particularmente en tareas como predicción y clasificación.

Con la intención de resumir las principales características de estos paradigmas evolutivos, en la Tabla 2.1 se muestra una comparación de los mismos teniendo en cuenta los aspectos descritos anteriormente.

Dentro de las meta-heurísticas bio-inspiradas existen otros exponentes que ha sido propuestos en los últimos años. Su similitud con los algoritmos evolutivos es algo clara en la mayoría de los casos, siendo la principal diferencia la definición de los operadores evolutivos que pueden estar inspirados en conceptos diferentes de la naturaleza. En este caso se encuentran los paradigmas basados en la inteligencia de enjambre (swarm intelligence) (Kennedy y Eberhart, 2001) caracterizados por la emergencia de un comportamiento inteligente a partir de la interacción de sus individuos. En esta familia se encuentran principalmente la *Optimización con Colonias de Hormigas* (*Ant Colony Optimization,*

Tabla 2.1 Principales características de los algoritmos evolutivos (ghazali Talbi, 2009).

Características	**Algoritmos genéticos**	**Estrategias evolutivas**	**Programación evolutiva**	**Programación genética**
Autores	J. Holland	I. Rechenberg y H-P. Schwefel	D. Fogel	J. Koza
Aplicaciones orginales	Optimización discreta	Optimización contínua	Aprendizaje automatizado	Aprendizaje automatizado
Atributos	No muy rápido	Optimización continua	–	Lento
Características especiales	Cruzamiento, diferentes variantes	Muy rápida, mucha teoría	Sin recombinación	–
Representación	Cadenas binarias	Vectores reales	Autómatas de estado finito	Árboles *parse*
Recombinación	n-point o uniforme	Discreta o intermediaria	No	Intercambio de sub-árboles
Mutación	Bit flipping con probabilidad fija	Perturbación gaussiana	Perturbación gaussiana	Cambios aleatorios en los árboles
Selección	Proporcional al fitness	Aleatoria uniforme	Determinística	Proporcional al fitness
Reemplazamiento (selección de sobrevivientes)	Todos los hijos reemplazan a los padres. (μ,μ)	(μ,λ) o $(\mu+\lambda)$	Probabilístico $(\mu+\mu)$	Generacional
Especialidad	Operador de cruzamiento	Auto-adaptación en el factor de mutación	Auto-adaptación	Necesita poblaciones grandes

ACO) (Dorigo et al, 1996) y la Optimización con Enjambre de Partículas (*Particle Swarm Optimization, PSO*) (Kennedy y Eberhart, 1995). Otros paradigmas evolutivos que han sido propuestos en las dos últimas dos décadas son: la Evolución Diferencial (Differential Evolution, DE) (Storn y Price, 1997), los Sistemas Inmunes Artificiales (Artificial Inmune Systems, AIS) (Farmer et al, 1986), y la optimización basada en la búsqueda de armonía (Harmony Search Optimization, HSO) (Geem et al, 2001).

Por la importancia que reviste para la presente investigación, en las secciones siguientes se describirán detalladamente los paradigmas Optimización con enjambre de partículas (PSO) y Evolución diferencial (DE).

2.2.4 Optimización con enjambre de partículas

PSO es una técnica de optimización estocástica basada en poblaciones, inspirada en el comportamiento social observado en animales o insectos (ej. cardúmenes de peces, bandadas de pájaros, etc) (Kennedy y Eberhart, 2001). Fue propuesta por primera vez por Eberhart y Kennedy (1995). Básicamente, los individuos en PSO reciben el nombre de partículas, y cada partícula i se compone por un vector de posición $\mathbf{x}_i$ (coordenadas en el espacio de búsqueda), un vector de velocidad $\mathbf{v}_i$ que define el desplazamiento de esa posición, y una memoria de la mejor solución encontrada hasta el momento por ella $\mathbf{p}_i$. En particular, la velocidad de una partícula está determinada tanto por $\mathbf{p}_i$ como por $\mathbf{g}_{best}$, la memoria global del enjambre $\mathscr{S}$ (ej. la mejor entre todas las partículas). Existen modelos que utilizan la mejor solución de una determinada vecindad, es decir, considerando solo una parte del enjambre. A estos modelos se les conocen como *lbest*, a diferencia del habitual *gbest* que es el considerado en esta investigación. La diferencia estriba en que en *lbest*, la memoria $\mathbf{g}_{best}$ es sustituida por el mejor localmente, $\mathbf{l}_{best}$.

Las ecuaciones originales de PSO para actualizar la velocidad y la posición de cada partícula i son las siguientes (Kennedy y Eberhart, 1995):

$$\mathbf{v}'_i = \mathbf{v}_i + c_1\eta_1 \circ (\mathbf{p}_i - \mathbf{x}_i) + c_2\eta_2 \circ (\mathbf{g}_{best} - \mathbf{x}_i) \tag{2.2}$$

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{v}'_i \tag{2.3}$$

Donde η_1 y η_2 son vectores n-dimensionales formados por números aleatorios en el rango $[0,1]$. Por otro lado, c_1 y c_2 son números reales denominados coeficientes de aceleración. El operador "$\circ$" especifica un producto Hadamard[2] entre las matrices formada por las coordenadas de los vectores, es decir, elemento a elemento. A modo de ilustrar

[2]Sean dos matrices $A, B \in \mathbb{R}^{m\times n}$, entonces "$\circ$" se define como el producto $(A \circ B) \in \mathbb{R}^{m\times n}$, donde $(A \cdot B)_{i,j} = A_{i,j} \cdot B_{i,j}$.

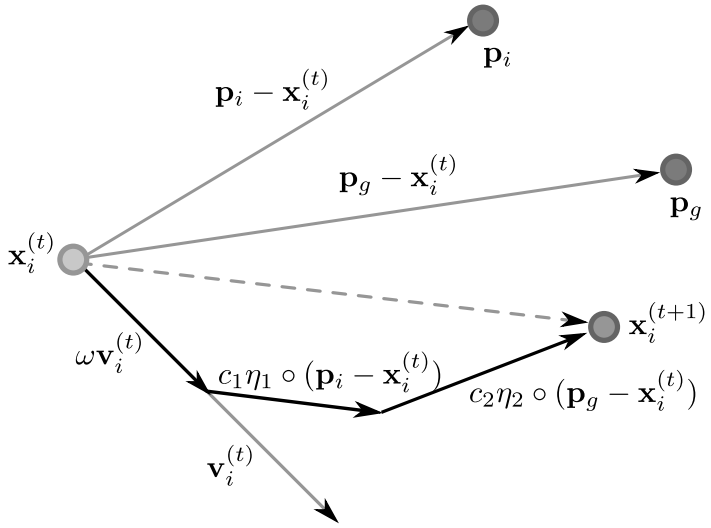

Figura 2.3 Movimiento que experimenta una partícula en PSO. La nueva posición ubica a la partícula en una zona intermedia entre las memorias personal y global.

geométricamente el movimiento de las partículas en PSO, la Figura 2.3 muestra a través de vectores una posible situación en un espacio bidimensional.

La expresión de la velocidad en sí, engloba el aporte principal de PSO que le permite clasificarse como un paradigma de la Inteligencia con Enjambres. Cada uno de sus componentes tiene un significado relacionado con el comportamiento social de los enjambres en la naturaleza, por ejemplo, $c_1\eta_1 \circ (\mathbf{p}_i - \mathbf{x}_i)$ representa el componente cognoscitivo del individuo, $c_2\eta_2 \circ (\mathbf{g}_{best} - \mathbf{x}_i)$ el social, y $\mathbf{v}_i$ el *momentum*: la velocidad previa que es usada para direccionar al individuo en su trayectoria (Blum y Li, 2008). Con el tiempo, las partículas deben converger a un peso promedio definido en función de sus dos atractores, a saber, $\mathbf{p}_i$ y $\mathbf{g}_{best}$. Los pasos principales de este paradigma se muestran en el algoritmo de la Figura 2.2.

Desde su surgimiento, PSO ha experimentado variaciones para controlar la propagación de la información entre partículas. En ese sentido se han propuesto diversas topologías de vecindarios, entre ellas se pueden mencionar las de tipo anillo, estrella, y von Neumann. Se ha podido comprobar que las topologías basadas en vecindarios restringidos, como las tipo von Neumann, son más apropiadas para problemas complejos, mientras que las topologías más grande (anillo, estrella) tienen mejor comportamiento en problemas sencillos (Mendes et al, 2004).

Un inconveniente que puede sufrir el PSO clásico es que las partículas pueden *explotar*: sus posiciones toman valores que las hacen salirse del espacio de búsqueda (ej. cuando están suficientemente lejos de $\mathbf{p}_i$ y $\mathbf{g}_{best}$). Para solucionarlo, se puede restringir las velocidades en un cierto valor V_{max}, por ejemplo, un 20 por ciento del espacio de búsqueda.

```
   // Initialization step
 1 para cada partícula i hacer
 2 |   Inicializar aleatoriamente v_i, x_i = p_i;
 3 |   Actualizar g_best;
 4 fin
 5 mientras not condición de parada hacer
 6 |   para cada partícula i hacer
 7 |   |   Actualizar i con las expresiones 2.2 y 2.3;
 8 |   |   Evaluar x_i;
 9 |   |   Actualizar memoria personal p_i;
10 |   |   Actualizar memoria global g_best;
11 |   fin
12 fin
```

Algoritmo 2.2: Pasos generales del paradigma Optimización con enjambre de partículas.

Un avance importante para asegurar la convergencia de PSO ha sido incluir un peso (denominado inercial) a la expresión de la velocidad. Considere por ejemplo, la siguiente expresión que modifica a la anterior (2.2), incluyendo ahora el peso ω.

$$\mathbf{v}_i' = \omega\mathbf{v}_i + c_1\eta_1 \circ (\mathbf{p}_i - \mathbf{x}_i + c_2\eta_2 \circ (\mathbf{g}_{best} - \mathbf{x}_i) \tag{2.4}$$

Una idea sugerente es utilizar ω <1.0, con esto se asegura un decrecimiento de la velocidad con el tiempo, comportamiento que no se logra por el contrario si ω >1.0. De hecho, se ha podido comprobar que con este peso inercial, PSO converge en determinados casos sin el uso de V_{max} (Clerc y Kennedy, 2002). Eberhart y Shi (2000) proponen el uso de este peso, de manera que disminuya con el tiempo desde 0.9 hasta 0.4.

Un análisis similar pero apoyado en fundamentos matemáticos sólidos, es llevado a cabo por Maurice Clerc en (Clerc y Kennedy, 2002; Clerc, 2006). Esta vez, se parte de un coeficiente que afecta a los tres componentes de la ecuación (2.2), denominado factor de constricción. PSO es analizado como un sistema dinámico simplificado unidimensional sin la presencia de los elementos estocásticos (η_1 y η_2), con el objetivo de determinar que pesos deben tener la velocidad y los componentes cognoscitivo y social, de manera que se asegure la convergencia de este sistema. El modelo simplificado puede ser escrito de la manera siguiente:

$$v_i' = \chi(v_i + c_1\eta_1 \circ (p_i - x_i) + c_2\eta_2 \circ (g_{best} - x_i)) \tag{2.5}$$

Donde $\chi = \frac{2}{|2-\phi-\sqrt{\phi^2-4\phi}|}$, $\phi = c_1 + c_2 > 4$.

Partiendo de $\phi = 4.1$ con $c_1 = c_2$ =2.05, implica que χ =0.7298. Sustituyendo χ en (2.5) se concluye que de la velocidad anterior solo será utilizada una 0.7298 parte, y que

los componentes cognoscitivo y social deben ser multiplicados, cuando más, por 1.4960. Precisamente estos datos (obtenidos analíticamente) han sido comprobados experimentalmente, incluso, se ha podido demostrar que el factor de constricción es equivalente al peso inercial (note que $0.4 < \chi < 0.9$) (Eberhart y Shi, 2000; Kennedy y Eberhart, 2001; Clerc y Kennedy, 2002). Resumiendo, la ecuación original (2.2) quedaría de la siguiente forma:

$$\mathbf{v}_i' = 0.7298\mathbf{v}_i + 1.4960\eta_1 \circ (\mathbf{p}_i - \mathbf{x}_i) + 1.4960\eta_2 \circ (\mathbf{g}_{best} - \mathbf{x}_i) \tag{2.6}$$

Al Algoritmo 2.2, con esta última ecuación, se le conoce comúnmente como PSO canónico (*Canonical PSO*). Sin embargo, aunque estos coeficientes permiten en muchos casos que PSO no *explote* se recomienda el uso de V_{max} (Eberhart y Shi, 2000), para un comportamiento más efectivo del algoritmo.

2.2.5 Evolución diferencial

La Evolución Diferencial (Storn y Price, 1997) es una meta-heurística muy popular en la última década y se basa en las leyes evolutivas de Darwing. En consecuencia, es comúnmente clasificado como un algoritmo evolutivo. Formalmente, DE posee una población de individuos (soluciones candidatas) representados por la tupla $\mathbf{y}_i = \langle \mathbf{x}_i, f_i \rangle$. Donde $\mathbf{x}_i \in \Omega$ es el vector posición y $f_i = f(\mathbf{x}_i)$ es el correspondiente valor de la función objetivo (*fitness*).

Como todo paradigma evolutivo, DE consta de dos etapas generales: inicialización y ciclo evolutivo (o principal). En la primera etapa, todos los individuos son ubicados en el espacio de búsqueda pseudo-aleatoriamente (e.g. según una distribución uniforme multivariada), y son posteriormente evaluados en la función objetivo. En el ciclo principal, la población de individuos es actualizada continuamente con nuevos individuos más aptos (de acuerdo a la función objetivo). El proceso de creación de de nuevos individuos es realizado mediante los operadores evolutivos *mutación* y *cruzamiento*. En particular, los principales parámetros que controlan a estos operadores evolutivos son: la tasa de cruzamiento CR, y el factor de escala F respectivamente. Por otro lado, la proceso de selección de nuevos individuos es simple y ocurre a nivel de individuo: si el nuevo individuo generado es mejor que su padre, entonces el primero toma el lugar del segundo en la población.

Los principales pasos del paradigma DE son mostrados por el Algoritmo 2.3. En particular, el parámetro μ define el tamaño de la población (cantidad de individuos), mientras que la expresión $\mathsf{alea}(0,1)$ es una función que retorna números pseudo-aleatorios siguiendo una distribución uniforme en el intervalo $[0,1]$.

Del pseudo-código anterior se puede notar que DE solo depende de unos pocos

```
   // Inicialización
1  Crear μ individuos;
2  para cada individuo i de la población hacer
3  |  Inicializar aleatoriamente x_i en el espacio de búsqueda;
4  |  Evaluar x_i;
5  fin
   // Ciclo principal
6  mientras no condición de parada hacer
   |  // Generación de nuevos individuos
7  |  para cada individuo i de la población hacer
   |  |  // Mutación
8  |  |  Crear un nuevo vector v empleando algún esquema de mutación (e.g. una de las
   |  |  expresiones 2.7-2.10);
   |  |  // Cruzamiento
9  |  |  Seleccionar aleatoriamente j ∈ [1,...,D];
10 |  |  Generar un vector de prueba u empleando cruzamiento binario:
11 |  |  para cada dimensión d ∈ [1,...,D] hacer
12 |  |  |  Asignar u^d ← { v^d  si j = d o alea(0,1) < CR ;
   |  |  |                { x_i^d por el contrario.
13 |  |  fin
   |  |  // Evaluar el vector de prueba
14 |  |  Asignar f_u ← f(u);
   |  |  // Selección
15 |  |  si f_u ≻ f_i entonces
16 |  |  |  Asignar y_i = ⟨u, f_u⟩;
17 |  |  fin
18 |  fin
19 fin
```

Algoritmo 2.3: Pasos generales del paradigma Evolución diferencial.

parámetros (μ, F y CR). No obstante, otros tipos de mutación y cruzamiento pueden ser usados. Dentro de los esquemas de mutación más conocidos para obtener el vector *donante* **v** se encuentran los siguientes:

- DE/Rand/1:

$$\mathbf{v} = \mathbf{x}_r + \mathsf{F}(\mathbf{x}_s - \mathbf{x}_t) \tag{2.7}$$

- DE/cur-to-best/1::

$$\mathbf{v} = \mathbf{x}_i + \mathsf{F}(\mathbf{x}_{best} - \mathbf{x}_i) + \mathsf{F}(\mathbf{x}_r - \mathbf{x}_s) \tag{2.8}$$

- DE/rand-to-best/2:

$$\mathbf{v} = \mathbf{x}_r + \mathsf{F}(\mathbf{x}_{best} - \mathbf{x}_i) + \mathsf{F}(\mathbf{x}_s - \mathbf{x}_t) + \mathsf{F}(\mathbf{x}_p - \mathbf{x}_q) \tag{2.9}$$

- DE/current-to-rand/1:

$$\mathbf{v} = \mathbf{x}_i + \mathsf{K}(\mathbf{x}_r - \mathbf{x}_i) + \mathsf{F}'(\mathbf{x}_s - \mathbf{x}_t) \tag{2.10}$$

Aquí, los vectores $\mathbf{x}_i$ y $\mathbf{x}_{best}$ corresponden al individuo i y al mejor de la población respectivamente. Asimismo, el resto de los vectores (aquellos marcados por los subíndices p,q,r,s,t) son seleccionados pseudo-aleatoriamente de la población cumpliéndose que $p \neq q \neq r \neq s \neq t$. Por otro lado, los escalares F, K y F' son factores de escala. Particularmente los dos últimos toman valores pseudo-aleatorios en tiempo de ejecución en el intervalo $[0,1]$.

En relación al operador cruzamiento, los dos más populares son el cruzamiento binario (empleado por el estándar DE, véase la Figura 2.3), y el cruzamiento *exponencial*. El lector interesado puede encontrar más información sobre DE en las excelentes revisiones de Neri y Tirronen (2010), y Das y Suganthan (2011).

2.3 Técnicas de configuración de parámetros

Como consecuencia de su propio diseño, todo algoritmo de optimización depende de determinados parámetros y estrategias, los cuales definen su comportamiento durante la ejecución en un problema particular. En ese sentido, se ha dicho que entre mejor sea el rendimiento de un algoritmo en un problema, así de buena es la representación del problema en los parámetros del algoritmo. En otras palabras, el sesgo de un algoritmo en un problema está en función de la información que puedan incluir sus parámetros sobre el problema en cuestión. De manera que es fácil convencerse de que debido a esto es imposible crear un algoritmo ideal capaz de tener un rendimiento superior a cualquier otro en todos los posibles problemas de optimización (tal y como establecen los Teoremas *No free lunch*, véase la Sección 2.2.2).

No obstante esta dificultad, es posible diseñar algoritmos eficientes para una determinada familia de problemas (e.g. seleccionando una configuración adecuada de los parámetros estratégicos). Sin embargo, elegir tal configuración no es una tarea fácil debido a las siguientes razones (Eiben et al, 2007; Smith, 2008):

1. **Los parámetros no son independientes**. Aunque diseñar experimentos multifactoriales pudiera ser una solución, en muchos casos combinar todos los posibles valores de los parámetros puede resultar en una explosión combinatoria.
2. **El ajuste puede llevar mucho tiempo**, incluso si los parámetros son ajustados individualmente (sin tener en cuenta sus interacciones).

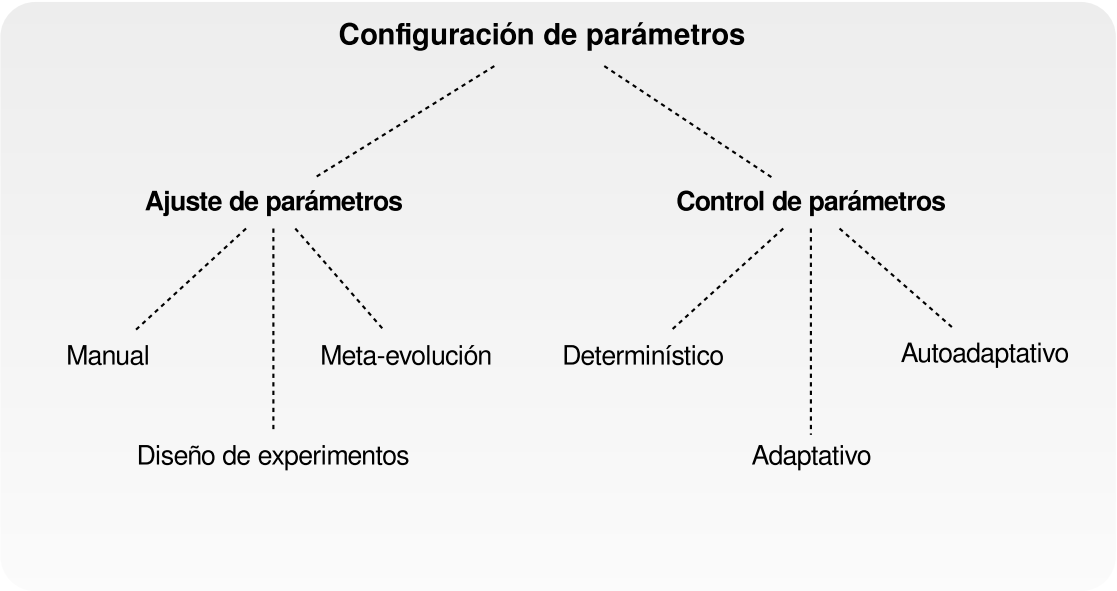

Figura 2.4 Taxonomía de los métodos de configuración de parámetros en la optimización (Kramer, 2010).

3. **Suboptimalidad de las configuraciones**. Dado que las meta-heurísticas son sistemas dinámicos, no existe garantía que una determinada configuración sea óptima para el estado actual de la búsqueda, sin mencionar para toda la ejecución.

Según Eiben et al (2007) y Kramer (2010), las técnicas actuales para la configuración de parámetros en algoritmos evolutivos se clasifican en dos clases generales: 1) *ajuste de parámetros* y 2) *control de parámetros* (véase la Figura 2.4). El rasgo distintivo de las técnicas pertenecientes a la primera clase es que se desarrollan antes de la ejecución (e.g. de manera *offline*), mientras que en la segunda las técnicas varían los parámetros durante la ejecución, en consecuencia el algoritmo puede explotar el estado actual de la búsqueda. En lo que sigue se describirán las principales características de estas categorías

2.3.1 Ajuste de parámetros

Como se aprecia en la Figura 2.4 el grupo de *Ajuste* queda dividido en tres categorías: el *ajuste manual*, *diseño de experimentos*, y *meta-evolución*. En lo que sigue se explicarán en que consisten estos enfoques.

Ajuste manual. En muchos casos los parámetros pueden ser ajustados manualmente. En este caso juega un papel esencial la experiencia del ser humano. Sin embargo, las configuraciones definidas por el usuario pueden no ser óptimas. Ejemplos de este tipo de

ajuste son el tamaño de la población, las tasas de cruzamiento y mutación, los cuales se predefinen antes de la ejecución del algoritmo. Por ejemplo, De Jong recomienda que la probabilidad de mutación en los algoritmos genéticos debe ser $pm = 0.001$ (De Jong, 1975), asimismo Schaffer et al (1989) recomiendan que $0.005 \leq pm \leq 0.01$. Otro ejemplo de este tipo de ajuste lo constituyen los parámetros ω, c_1, y c_2 en el paradigma PSO, y F y CR en Evolución diferencial.

Diseño de experimentos. En la actualidad herramientas estadísticas como el diseño de experimentos (*Design of experiments*, DoE) han sido aplicadas con éxito en el ajuste de parámetros. En ese sentido, Bartz-Beielstein (2006) brinda una excelente introducción a la investigación experimental en la computación evolutiva. Un diseño experimental es básicamente un plan experimental detallado que se establece antes de realizar los experimentos. Los DoE comienzan con la determinación de los objetivos de un experimento y la elección de los parámetros (factores) que serán objeto de estudio. De manera que es la calidad de los experimentos (respuesta) la que guía la búsqueda de configuraciones apropiadas. En un experimento, un DoE cambia deliberadamente uno o varios factores, con el objetivo de observar los cambios que se producen en una o varias variables de respuesta. La respuesta puede definirse como como la calidad de los resultados en términos de la medida de rendimiento empleada, e.g., la media del fitness en una generación o la tasa de convergencia. Un buen diseño de experimentos maximiza la información que puede ser obtenida a partir de un determinado esfuerzo experimental. Como ejemplos de este tipo de ajuste de parámetros se pueden mencionar los trabajos de Bartz-Beielstein et al (2005); Bartz-Beielstein y Preuss (2006) donde se propone a SPO (Sequential Parameter Optimization) un método de ajuste para algoritmos con salidas (respuestas) perturbadas estocásticamente. SPO combina métodos clásicos de regresión y otros enfoques estadísticos. De manera general, SPO ha sido aplicado con éxito en varios contextos, por ejemplo Preuss y Bartz-Beielstein (2007) emplean SPO para ajustar un algoritmo de codificación binaria.

Meta-evolución. En esta categoría aparecen los algoritmos *meta-evolutivos*, también conocidos como algoritmos evolutivos anidados. En esencia, la esencia de este tipo de ajuste de parámetros es que el proceso de optimización (evolución) es llevado a cabo en dos niveles (Kramer, 2010): un algoritmo externo ajusta los parámetros de otro que es el encargado de resolver el problema de optimización. Aquí juega un papel fundamental lo que se conoce como *isolation time*, el tiempo que tendrá el algoritmo interno para optimizar el problema, de manera que sus resultados guían la búsqueda del algoritmo externo. Es de notar que aunque estos enfoques anidados son capaces de ajustar adecuadamente los parámetros del algoritmo interno, son en sentido general algo ineficientes. Ejemplos de

meta-evolución son: el meta-GA de Grefenstette (1989), el cual optimiza un GA clásico; las estrategias meta-evolutivas de Herdy (1992) quien además analiza teóricamente una meta-(1, k)-ES en el problema de la Esfera usando resultados de la teoría de las tasas de progreso; las Estrategias evolutivas anidadas de Coello Coello (2000) que adaptan los factores de una función de penalización para problemas restringidos. Una propuesta más sofisticada es la de de Landgraaf et al (2007) donde se propone un método de estimación de relevancia y calibración de valores (REVAC) con el objetivo de estimar la sensibilidad de los parámetros y la elección de sus valores. En esencia, el método está basado en la teoría de la información y es considerado un Algoritmo de estimación de distribución (EDA).

2.3.2 Control de parámetros

Las técnicas dentro de este grupo de configuración de parámetros suponen que los cambios de los parámetros estratégicos son más influyentes durante la ejecución del algoritmo.

Control determinístico. El control determinístico de parámetros significa que los parámetros serán ajustados de acuerdo a un esquema fijo de tiempo, esto es, dependiente explícitamente del número de generaciones t. Por ejemplo, ha sido notado por varios investigadores que el factor de mutación en los algoritmos evolutivos debe reducirse durante el proceso de búsqueda con el objetivo de garantizar la convergencia de la población. En ese sentido, en Fogarty (1989) se propone que la probabilidad de mutación en los GA disminuya de la siguiente forma:

$$p_m(t) = \frac{1}{240} + \frac{11.375}{t^2}$$

Asimismo, Bäck y Schütz (1996) establece la siguiente regla para la probabilidad de mutación:

$$p_m(t) = \left(2 + \frac{l-2}{T-1} \cdot t\right)$$

donde T es el número máximo de generaciones. Otro campo de aplicación de este tipo de control es la optimización con restricciones. Aquí conviene que las penalizaciones aumenten conforme el proceso de búsqueda avanza, el motivo es simple: evitar obtener soluciones no factibles en las últimas generaciones. Basados en este análisis, Joines y Houck (1994) proponen la siguiente función de penalización dinámica:

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + (C \cdot t)^{\alpha} \cdot G(\mathbf{x})$$

donde $f(\mathbf{x})$ es la función objetivo del modelo, $G(\mathbf{x})$ es una medida de la violación de las restricciones, mientras que C y α son dos parámetros preestablecidos. Más recientemente,

Mezura-Montes y Palomeque-Ortiz (2009) aplican igualmente técnicas determinísticas en el paradigma Evolución diferencial para resolver problemas de optimización restringida.

Lo importante a notar en todos estos ejemplos es que las reglas de control son aplicadas independientemente de cómo se encuentre el proceso de búsqueda.

Control adaptativo. En este tipo de control el algoritmo se *retroalimenta* (toma información) del proceso de búsqueda con el objetivo de determinar la magnitud de los cambios en el parámetro. Quizás el ejemplo más popular es la regla de tasa de éxito de Rechenberg (1973), la cual está dedicada a adaptar el factor de mutación en el paradigma ES. En esencia, la regla establece que (Eiben et al, 2007): si la tasa de éxito en una ventana de tiempo determinada es mayor que la tasa de éxito considerada óptima (e.g. 0.2 ó 1/5) entonces el factor de mutación debe ser incrementado (se asume que el algoritmo está dando pasos muy pequeños); por otro lado, si la tasa de éxito de la ventana es menor que 0.2, entonces el factor de mutación debe decrementarse (el algoritmo está perdiendo al óptimo pues los pasos son muy grandes).

Otro ejemplo de control adaptativo en el contexto de la optimización mediante sistemas cooperativos multi-agentes, es la regla difusa propuesta por Pelta et al (2009) para actualizar las celdas del mundo donde los agentes exploran. Básicamente la regla es la siguiente:

> **Si** la calidad de la solución obtenida por el agente a_i es *baja* **entonces** a_i toma la mejor solución del *CIR* y reemplaza la que tiene en su celda.

En ese sentido es posible ver que la información del problema utilizada por el algoritmo es la calidad actual de la celda, la cual es evaluada teniendo en cuenta su grado de pertenencia a la etiqueta *baja*. Para tal propósito los autores proponen la siguiente función de pertenencia difusa:

$$\mu(f_i) = \begin{cases} 0.0 & \text{si } f_i > \beta; \\ \dfrac{(\beta - f_i)}{(\beta - \alpha)} & \text{si } \alpha \leq f_i \leq \beta; \\ 1.0 & \text{si } f_i < \alpha. \end{cases}$$

Note que, al igual que la tasa de éxito de Rechenberg, los parámetros que definen dicha función son constantes predefinidas de antemano y permanecen fijos en el tiempo.

Control auto-adaptativo. Finalmente, si el algoritmo es capaz de ajustar los parámetros sin la intervención de reglas de control externas, entonces se dice que el algoritmo presenta un control auto-adaptativo (Meyer-Nieberg y Beyer, 2007; Eiben et al, 2007).

En ese sentido, el ejemplo más común es el de las Estrategias evolutivas auto-adaptativas, donde el factor de mutación σ_i es codificado por sus individuos (e.g. en conjunto al vector

solución). Formalmente un individuo viene dado por $\mathbf{y}_i = \langle \mathbf{x}_i, \sigma_i \rangle$. Antes de cada variación del vector $\mathbf{x}_i$, los parámetros σ_i son actualizados de la siguiente forma:

$$\sigma_i' = \sigma_i \cdot \exp\Big(\tau_1 \cdot N(0,1) + \tau_2 \cdot N_i(0,1)\Big)$$
$$\mathbf{x}_i' = \mathbf{x}_i + \sigma_i' \circ N_i(0,1)$$

donde $N(0,1)$ es un número aleatorio proveniente de una distribución normal, y los parámetros τ_1 y τ_2 son constantes predefinidas.

Otro ejemplo de auto-adaptación es el esquema propuesto por Brest et al (2006) para actualizar los parámetros F y CR de cada individuo *i* en el paradigma Evolución diferencial:

$$\tilde{\mathsf{F}}_i = \begin{cases} 0.1 + 0.9 \cdot \mathsf{alea}(0,1) & \text{si } \mathsf{alea}(0,1) < \tau_1; \\ \mathsf{F}_i & \text{por el contrario.} \end{cases} \tag{2.11}$$

$$\tilde{\mathsf{CR}}_i = \begin{cases} \mathsf{alea}(0,1) & \text{si } \mathsf{alea}(0,1) < \tau_2; \\ \mathsf{CR}_i & \text{por el contrario.} \end{cases} \tag{2.12}$$

aquí igualmente los parámetros τ_1 y τ_2 son constantes predefinidas antes de la ejecución. Por otro lado, $\mathsf{alea}(0,1)$ es una función que devuelve números aleatorios reales en el intervalo $[0,1]$ de manera uniforme.

Dos cuestiones deben ser observadas en este tipo de control de parámetros. Primero, su diferencia en relación al control adaptativo es que la regla de actualización de los parámetros no es externa, esto es, depende del estado actual de la búsqueda. En segundo lugar, aunque la auto-adaptación es uno de los caminos para obtener un algoritmo más inteligente, esto no implica que se eliminen parámetros. Por ejemplo, note que los dos ejemplos descritos anteriormente dependen de dos parámetros τ_1 y τ_2, los cuales permanecen fijos durante la ejecución. Desde este punto de vista, la auto-adaptación puede verse como un tipo de meta-evolución implícita, esto es, sin la presencia de un algoritmo externo.

Intuitivamente, de todos los tipos de técnicas de control de parámetros, las auto-adaptativas son las más adecuadas para un algoritmo, sobre todo si el objetivo es favorecer el rendimiento del algoritmo en una amplia variedad de problemas. A esta capacidad de tener un rendimiento aceptable en problemas con características diferentes se le conoce habitualmente como *robustez*. Como se verá más adelante este es uno de los objetivos de la presente investigación: desarrollar algoritmos robustos en el contexto de la optimización dinámica.

2.4 Optimización en ambientes dinámicos

Una vez introducidos los elementos básicos anteriores, en este apartado nos centraremos en la optimización en ambientes dinámicos. Primero, se dará una definición formal de problema dinámico de optimización, y se describirá el rol que juega la función que rige la dinámica de los cambios a través de algunos ejemplos sencillos. Más adelante, los principales problemas dinámicos artificiales son descritos detalladamente. En una segunda parte, se analizan los principales enfoques de adaptación dinámica existentes y su rol dentro de las meta-heurísticas poblacionales. Finalmente, se revisan las medidas de rendimiento más importantes en este contexto.

2.4.1 Problemas dinámicos

Intuitivamente se puede asumir que un problema dinámico es un conjunto de problemas estacionarios aproximados que deben ser resueltos (optimizados) de manera secuencial, esto es, uno a continuación del otro. De manera específica, cada problema estacionario debe ser optimizado en un determinado tiempo, esto es, antes de que ocurra el próximo cambio en el ambiente y así la aparición del siguiente problema. Resulta también de de interés definir a los problemas dinámicos en función de la dinámica que presentan los cambios. Esta dinámica puede considerarse como una función Φ_{dyn} con dominio e imagen en el espacio de los problemas estacionarios $\mathscr{E}$. Formalmente, $\Phi_{dyn} : \mathscr{E} \rightarrow \mathscr{E}$, de manera que $\varepsilon^{(t+1)} = \Phi_{dyn}(\varepsilon^{(t)})$. En general, se espera que $\varepsilon^{(t+1)} \sim \varepsilon^{(t)}$, esto es, que la transición entre problemas estacionarios se realice mediante cambios no *muy caóticos*, y por tanto ambos problemas sean similares entre sí.

Teniendo en cuenta estas consideraciones y para propósitos de este trabajo, un problema dinámico de optimización se define de la siguiente forma:

Definición 2.4.1 (**Problema dinámico de optimización** (PDO))**.** Sea el conjunto de problemas estacionarios $\mathbf{E} \subseteq \mathscr{E}$ y la función $\Phi_{dyn} : \mathscr{E} \rightarrow \mathscr{E}$. El objetivo es optimizar secuencialmente cada problema estacionario $\varepsilon^{(t)} \in \mathbf{E}$, tal que $\varepsilon^{(t+1)} = \Phi_{dyn}(\varepsilon^{(t)})$.

A partir de la definición dada anteriormente, se puede ver que la función Φ_{dyn} es un tipo especial de función de transición entre los elementos de la sucesión de problemas estacionarios que conforman a un problema dinámico determinado. Dado que los problema estacionarios son del tipo aproximado, estos pueden verse como una tupla $\widetilde{\varepsilon}^{(t)} = \langle \Omega^{(t)}, f^{(t)}, \succeq^{(t)}, T^{(t)}, \varepsilon^{(t)} \rangle$. En consecuencia, la función Φ_{dyn} establece como será la

transición de los elementos de un problema a otro:

$$\begin{pmatrix} \Omega^{(t)} \\ F^{(t)} \\ \succeq^{(t)} \\ T^{(t)} \\ \varepsilon^{(t)} \end{pmatrix} \xrightarrow{\Phi_{dyn}} \begin{pmatrix} \Omega^{(t+1)} \\ F^{(t+1)} \\ \succeq^{(t+1)} \\ T^{(t+1)} \\ \varepsilon^{(t+1)} \end{pmatrix}$$

Sin embargo, se puede apreciar que no todos los elementos son atómicos, es decir, los mismos están compuestos por otros. Por ejemplo, el espacio de búsqueda Ω es definido habitualmente por un conjunto de restricciones (eg. del tipo igual y menor igual) y por la dimensión del mismo. Asimismo, la función objetivo pudiera depender de la posición de los óptimos globales (y locales) del problema y de algunos parámetros que definen su forma geométrica (ej. anchura o altura). Por el contrario, el resto de los elementos suelen estar definidos de manera más precisa, ya que $\succeq \in \{\leq, \geq\}$, $T \in \mathbb{N}_0$ y $\varepsilon \in \mathbb{R}$, esto es, representan elementos de conjuntos bien definidos.

A modo de ejemplo en lo que sigue se describirán algunos casos donde la función objetivo es el elemento que varía en el tiempo.

Ejemplo 2.4.1. Cambio de forma en la función objetivo. Considere la secuencia de problemas estacionarios definidos por las funciones f_i siguientes (note que se trata de distintas variaciones de la función $y = -x^2$): $\{f^{(0)}(x) = -x^2, f^{(1)}(x) = -5x^2, f^{(2)}(x) = -0.3x^2, f^{(3)}(x) = -x^2 + 1, f^{(4)}(x) = -x^2 - 2, ...\}$

Con ayuda de las Figuras 2.5 a) y 2.5 b), es posible ver que, con respecto a la función objetivo inicial $f^{(0)}$, en $f^{(1)}$ y $f^{(2)}$ la parábola se ha contraído y expandido respectivamente (cambio en la anchura), mientras que en $f^{(3)}$ y $f^{(4)}$, se ha retornado a la forma inicial, pero con un aumento y luego una disminución del valor del óptimo (altura).$\diamondsuit$

Evidentemente, estos cambios de forma se deben a un cambio en la fórmula matemática de la función objetivo. Para los algoritmos que basen su búsqueda en la propia estructura de la función objetivo (ej. algoritmos basados en gradiente) esto presupone tener en cuenta las distintas expresiones en cada instante de tiempo, ya que se corre el riesgo de utilizar representaciones falsas de la realidad actual del problema (ej. derivadas de funciones que no existen en un determinado instante de tiempo). En aquellos algoritmos que utilicen la mejor solución encontrada en la generación de soluciones, un cambio en la altura, puede provocar la toma de decisiones erróneas. Por otro lado, un cambio en la forma puede hacer más difícil (o más fácil) la búsqueda, ya que disminuye (aumenta) la probabilidad de encontrar al óptimo en el espacio de búsqueda. Por ejemplo, para el espacio de búsqueda

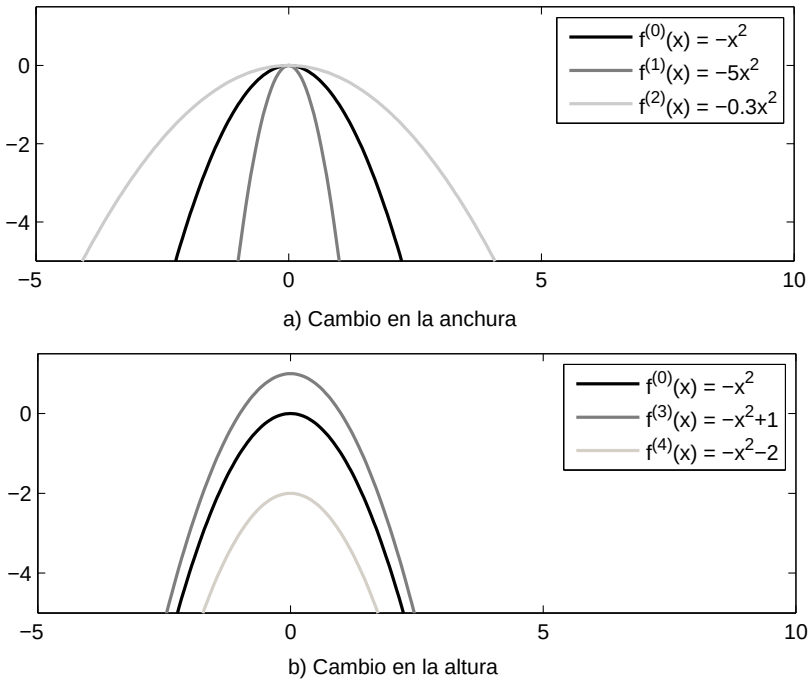

Figura 2.5 Cambio en la forma de un óptimo. Las gráficas representan variaciones de la función de la parábola invertida $y = -x^2$.

$\Omega = [-5.0; 5.0]$ la función $f^{(1)}$ pudiera ser más difícil de optimizar que $f^{(2)}$ ya que esta última ofrece un seno de atracción mayor que la primera.

Sin embargo, este tipo de cambio no suele traer graves dificultades para un algoritmo iterativo, ya que puede ser resuelto con la evaluación de las soluciones en una nueva iteración.

Ejemplo 2.4.2. Cambio en la posición del óptimo. Además de su valor en la función objetivo, un óptimo puede cambiar de posición, es decir, su valor puede desplazarse de las coordenadas originales hacia otras nuevas. Note que esto puede ocurrir sin que necesariamente se altere su valor en la función objetivo. Matemáticamente, esto se conoce como una transformación en el sistema de coordenadas. Note que en el ejemplo anterior, las soluciones óptimas para las distintas funciones son las mismas ($x^* = 0$, el óptimo no se ha movido). Sin embargo, considere ahora la siguiente sucesión como continuación del ejemplo anterior: $\{..., f^{(5)}(x) = -(x+2)^2, f^{(6)}(x) = -3(x+3)^2 - 1, ...\}$

En este caso ocurren desplazamientos hacia la izquierda de $f^{(0)}$ (Figura 2.6). Asimismo, con la intención de ilustrar que efecto tendría si se combinan los cambios del ejemplo anterior y los de la posición del óptimo, obsérvese lo que sucede para el caso de $f^{(6)}$.

Este elemento del modelo posee un efecto importante en el algoritmo si se tiene en

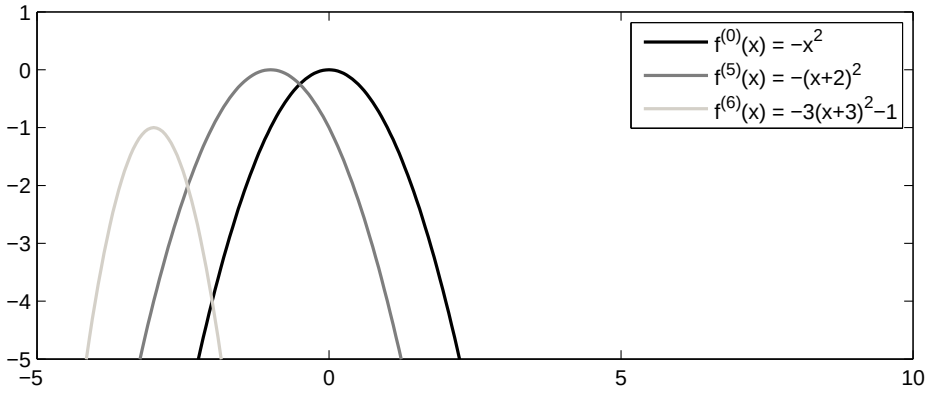

Figura 2.6 Cambio en la posición de un óptimo. Las gráficas son variantes de la función de la parábola invertida $y = -x^2$.

cuenta que un cambio de posición del óptimo del problema puede provocar que el algoritmo se enfoque en un *falso* óptimo. En general, se trata de un problema similar al de optimizar funciones *multimodales* en las que los algoritmos pueden converger a óptimos locales.

Tipos de cambio

En la sección anterior se analizaron que elementos del modelo pueden ser objeto de cambio a través de la función Φ_{dyn}, sin embargo es de interés también conocer de que forma pueden ocurrir estos cambios. Desde un punto de vista formal, la naturaleza de los cambios pueden expresarse matemáticamente mediante la función Φ_{dyn}.

En la literatura se han reportado diversos tipos de cambios, siendo los más populares los siguientes (Branke, 1999b; Weicker, 2002; Li y Yang, 2008a):

- Uniforme aleatorio.
- De paso fijo.
- Con correlación con el cambio anterior.
- Caótico.
- Recurrente.
- Recurrente con ruido.

En los siguientes ejemplos se pueden observar con más detalle dos de estos tipos de cambios.

Ejemplo 2.4.3 (**Tipo de cambio de paso fijo**)**.** Este es sin lugar a dudas uno de los tipos de cambios más simples. En sentido, considere por ejemplo que el problema varía la frecuencia

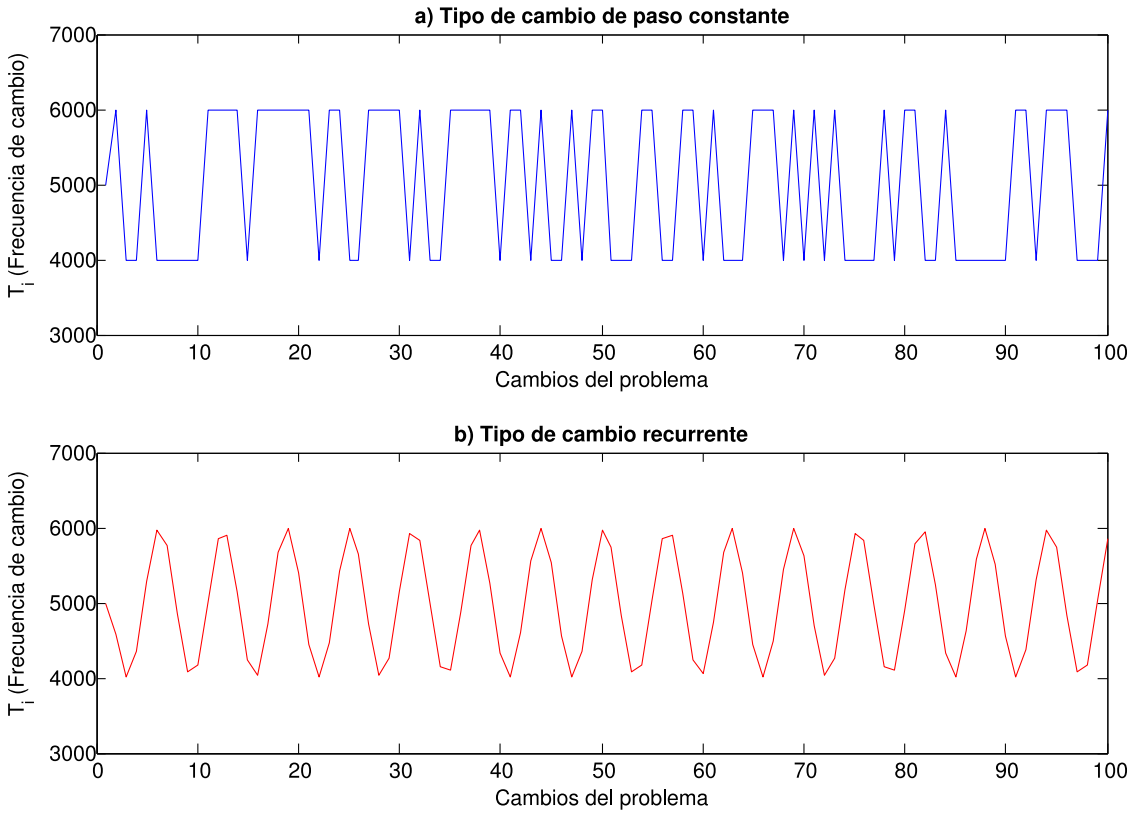

Figura 2.7 Tipos de cambio de paso constante a), y de tipo recurrente b) para el elemento T_i (frecuencia de cambio).

de los cambios, esto es, el elemento T_i asociado a cada problema estacionario i. Una función de cambio con paso fijo para este elemento pudiera ser la siguiente:

$$T^{(t+1)} = \begin{cases} T^{(0)} + \sigma_{step} & \text{si alea}(0,1) < 0.5 \\ T^{(0)} - \sigma_{step} & \text{en otro caso.} \end{cases}$$

donde $T^{(0)} \in \mathbb{N}_0$ es un valor fijo inicial, $\text{alea}(0,1)$ es una función que devuelve valores aleatorios uniformemente distribuidos en el intervalo $[0,1]$, y $\sigma_{step} \in \mathbb{R}_+$ es el paso.

Para ilustrar mejor lo que sucede con esta magnitud conforme pasa el tiempo, obsérvese la Figura 2.7 a), que muestra la evolución de T_i en función de los cambios del problema. Note que en esta gráfica, $T^{(0)} = 5000$ y $\sigma_{step} = 1000$, lo que significa que el problema dinámico alternará entre problemas estacionarios con frecuencia de cambio 4000 ó 6000.

Otro tipo de cambio que puede ocurrir en problemas reales es el recurrente. Su principal características es la repetición de elementos de manera cíclica.

Ejemplo 2.4.4 (**Tipo de cambio recurrente**)**.** Una manera fácil de definir un cambio recurrente es a través de una función trigonométrica periódica (ej. seno o coseno).

$$T^{(t)} = \lceil 5000 + 1000 \cdot \cos(i) \rceil$$

donde $\lceil \rceil$ es un operador que devuelve la parte entera del argumento. Si observa la Figura 2.7 que al igual que el ejemplo anterior, el elemento $T^{(t)}$ comienza en 5000 y varía en el rango $[4000, 6000]$, pero en esta ocasión la tendencia que sigue este elemento es cíclica y puede tomar valores distintos de 4000, 5000 y 6000. Finalmente, es importante señalar que una de las formas de afrontar este tipo de cambio es mediante el uso de memorias, esto es, empleando información de etapas anteriores que puedan ser útiles en el presente. Precisamente este es uno de los enfoques más importantes en ambientes dinámicos como se verá en el próximo capítulo.

2.4.2 Problemas dinámicos artificiales

Un problema artificial es, en general, un entorno de simulación que sirve para estudiar el comportamiento de un algoritmo. En este sentido, la importancia que reviste un problema de este tipo se soporta en dos aspectos fundamentales: 1) se evitan las consecuencias negativas que pueden ocurrir en entornos reales (a través de la simulación), y 2) favorecen la comparación de propuestas en la comunidad científica.

Hasta ahora el número de problemas artificiales en ambientes dinámicos es relativamente pequeño, aunque en principio, cualquier problema estacionario puede ser adaptado para convertirlo en un PDO (ej. incluyendo la dimensión temporal). Entre los más conocidos figuran el Problema de la Mochila Variable en el Tiempo (Goldberg y Smith, 1987; Dasgupta y Mcgregor, 1992; Mori et al, 1996), el de la Parábola Móvil (Angeline, 1997), Bit-matching Dinámico (Pettit y Swigger, 1983; Dorigo et al, 2000; Stanhope y Daida, 1999), el generador DF1 introducido por Morrison y De Jong (1999), Planificación de trabajos que arriban con el tiempo (Mattfeld y Bierwirth, 2004), entre otros. No obstante la utilidad de estos problemas, en la actualidad se han hecho muy populares los generadores de problemas dinámicos. Estos generadores suelen aparecer en código fuente de algún lenguaje de alto nivel y permiten, a través de la configuración de sus parámetros, la obtención de diferentes familias de problemas dinámicos. Tres de los generadores más importantes se muestran en la Tabla 2.2 junto a los principales trabajos que los emplearon en sus análisis experimentales. En lo que sigue se describirán brevemente los dos últimos generadores, por ser estos los que se utilizarán en el resto de la tesis.

Tabla 2.2 Distribución de trabajos según el generador de problemas artificiales utilizado.

Generador de problemas	Referencias
DF1 generator (cones) (Morrison y De Jong, 1999)	(Morrison y De Jong, 1999; Saleem y Reynolds, 2000; Blackwell y Bentley, 2002a,b; Hu y Eberhart, 2002; Eriksson y Olsson, 2002; Li y Dam, 2003; Morrison, 2003; Parrott y Li, 2004; Esquivel y Coello Coello, 2004; Morrison, 2004; Peng y Reynolds, 2004)
Moving peaks benchmark (Branke, 1999b)	(Branke, 1999a, 2002; Janson y Middendorf, 2003; Blackwell y Branke, 2004; Parrott y Li, 2004; Eriksson y Olsson, 2004; Zou et al, 2004; Janson y Middendorf, 2004; Blackwell y Branke, 2006; Du y Li, 2008; Novoa y Cruz, 2009; Novoa et al, 2009; Pelta et al, 2009; Trojanowski y Wierzchon, 2009; García Del Amo et al, 2010; Moser y Chiong, 2010; Novoa-Hernández et al, 2010, 2011; du Plessis y Engelbrecht, 2011, 2012; Duhain y Engelbrecht, 2012; Sharifi et al, 2012; Wan et al, 2012; Xiao y Zuo, 2012; Yazdani et al, 2012)
Generalized dynamic benchmark generator (Li y Yang, 2008a).	**Competición CEC'2009:** (Brest et al, 2009; Korosec y Silc, 2009; Li y Yang, 2009; Olivetti de França y Zuben, 2009; Yu y Suganthan, 2009; du Plessis y Engelbrecht, 2011, 2012; Novoa-Hernández et al, 2013a)
	Competición IEEE WCCI-2012: (Hui y Suganthan, 2012; Korosec y Silc, 2012; Lepagnot et al, 2012; Li et al, 2012)

Problema de *Movimiento de Picos*

El problema de *Movimiento de Picos* más conocido como MPB por sus siglas en inglés (*Moving Peaks Benchmark*) fue propuesto por Branke (1999b). Es un problema que provee una función multimodal altamente parametrizable que varía con el tiempo. Se encuentra implementado en los lenguajes de programación ANSI C y Java. En este problema se emplea la composición de funciones para definir la función objetivo:

$$F_i(\mathbf{x}) = \max\left\{B(\mathbf{x}), \max_{i=1,\ldots,m} P\left(\mathbf{x}, h_i, w_i, \mathbf{p}_i\right)\right\}$$

donde $\mathbf{x} \in \Omega$, $B(\mathbf{x})$ es una función base constante (invariable en el tiempo), y P es una función que define la forma general de los picos. Cada pico i tiene sus propios parámetros variables en el tiempo, a saber, altura (h), anchura (w), y posición ($\mathbf{p}$). Precisamente estos parámetros son la base de la naturaleza cambiante del problema. Estos se inicializan aleatoriamente en el espacio según un generador de números aleatorios, y cada Δe evaluaciones de la función objetivo, son afectados mediante la adición de una variable gaussiana $(\cdot)$. A su

vez, las coordenadas son modificadas por un vector **v** de una longitud fija s en una dirección aleatoria (ej. si $\gamma = 0$), o en una dirección correlacionada a la anterior (si $\gamma > 0$).

En resumen, cada cambio está definido por su severidad (s), frecuencia (Δe), y tendencia (γ). Una definición formal de como varían los parámetros de este problema es la que sigue:

$$\begin{aligned} \sigma &\in N(0,1) \\ h_i^{(t+1)} &= h_i^{(t)} + s_{height} \cdot \sigma \\ w_i^{(t+1)} &= w_i^{(t)} + s_{width} \cdot \sigma \\ \mathbf{p}_i^{(t+1)} &= \mathbf{p}_i^{(t)} + \mathbf{v}_i^{(t+1)} \end{aligned}$$

El vector $\mathbf{v}_i^{(t+1)}$ es una combinación lineal de un vector **r** y el vector previo $\mathbf{v}_i^{(t)}$, y al mismo tiempo normalizado mediante la longitud s:

$$\mathbf{v}_i^{(t+1)} = \frac{s}{\left|\mathbf{r} + \mathbf{v}_i^{(t)}\right|}\left((1-\gamma)\cdot\mathbf{r} + \gamma\cdot\mathbf{v}_i^{(t)}\right)$$

Para ayudar en la comparación y utilización de este problema, su autor ha definido tres escenarios que a su vez, simulan instancias independientes de problemas reales (Branke, 1999b). Estos escenarios se muestran en la Tabla 2.3, en donde se especifican los posibles valores de los parámetros explicados anteriormente. En particular, la función que define a los picos en el Escenario 2 es la siguiente:

$$P(\mathbf{x}, h_i, w_i, \mathbf{p}_i) = h_i - w_i \cdot f(\mathbf{p}_i - \mathbf{x}) \tag{2.13}$$

donde f es la función que le da forma a los picos.

Cada uno de estos escenarios pueden poseer características distintas. A modo de ilustrar este aspecto, las Figuras 2.8 y 2.9 muestran el escenario 2 con distintas funciones para representar a los picos: unimodales y multimodales. Note sin embargo que en todos los casos, la función que se obtiene producto a la composición de los picos, es multimodal.

GDBG: un generador de problemas dinámicos

Un representante más moderno dentro de los generadores de problemas dinámicos dinámicos lo constituye el (*Generalized Dynamic Benchmark Generator*, GDBG). Este generador inicialmente propuesto por Li y Yang (2008b), fue utilizado en la competición *Evolutionary Computation in Dynamic and Uncertain Environments Competition* perteneciente al Congreso de Computación Evolutiva del 2009 (Li et al, 2008).

Tabla 2.3 Escenarios definidos para el Problema de Movimiento de Picos (Branke, 1999b).

Parámetros	Escenario 1	Escenario 2	Escenario 3
Número de picos (m)	5	10-200	50
Dimensiones (n)	5	5	5
Altura (h)	30.0-70.0	30.0-70.0	30.0-70.0
Anchura (w)	0.0001-0.2	1.0-12.0	1.0-12.0
Coordenadas	0.0-100.0	0.0-100.0	0.0-100.0
Severidad (s)	0.0-2.0	0.0-3.0	1.0
Severidad altura (s_{height})	7.0	7.0	7.0
Severidad anchura (s_{width})	0.01	1.0	0.5
Factor de correlación (γ)	0.0	0.0-1.0	0.5
Frecuencia de cambio (Δe)	5000	1000-5000	1000

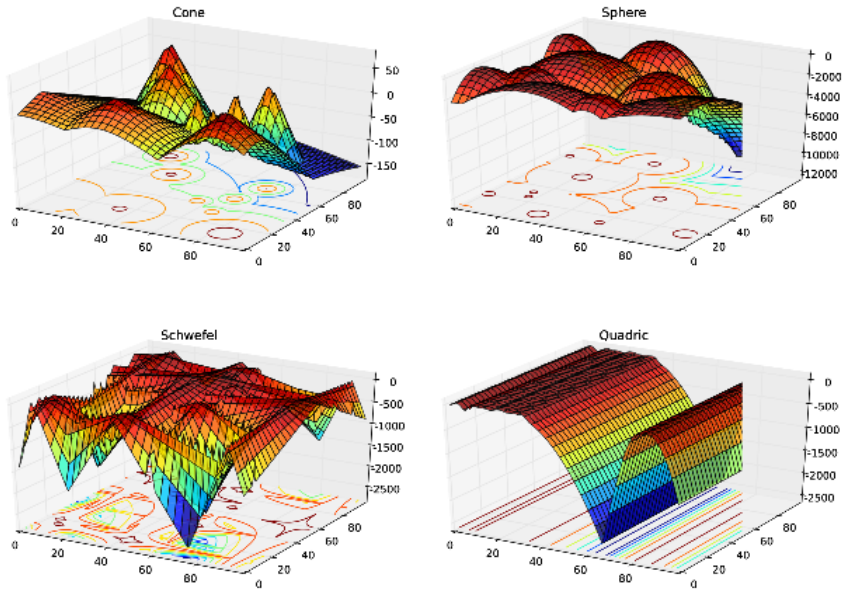

Figura 2.8 Problema de Movimiento de Picos con 10 picos representados por distintas funciones unimodales: *Cone, Sphere, Schwefel*, y *Quadric*. Las gráficas corresponden a un instante de tiempo t específico.

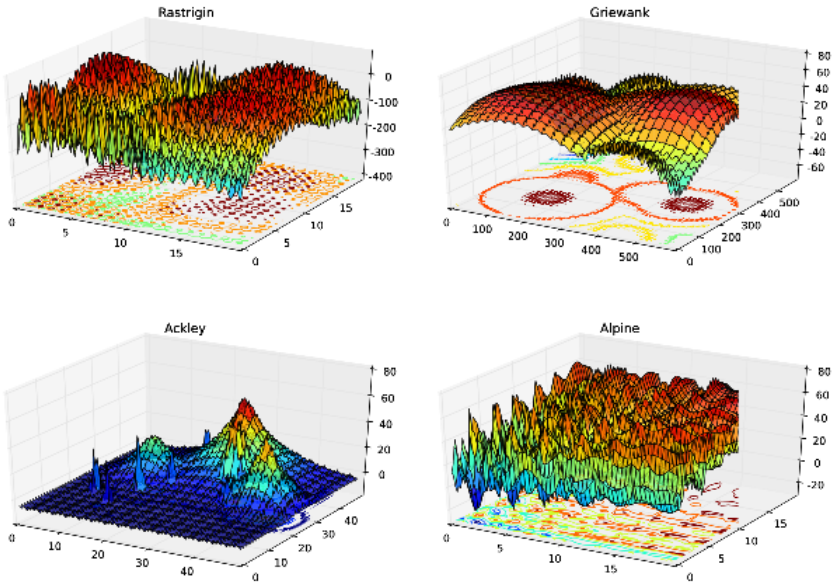

Figura 2.9 Problema de Movimiento de Picos con 10 picos representados por distintas funciones multimodales: *Rastrigin, Griewank, Ackley*, y *Alpine*. Las gráficas corresponden a un instante de tiempo t específico.

Para los autores, un problema dinámico se define como:

$$F = f(\mathbf{x}, \phi, t)$$

donde F es el problema de optimización, f es la función de costo, $\mathbf{x}$ es una solución dentro del conjunto de soluciones factibles, t es el tiempo del mundo real, y ϕ es el parámetro de control del sistema, el cual determina la distribución de soluciones en el espacio de búsqueda.

En el generador GDBG, el dinamismo proviene a partir de la desviación de la distribución de soluciones en el ambiente, mediante el ajuste de los parámetros de control del sistema:

$$\phi(t+1) = \phi(t) \oplus \Delta\phi$$

donde $\Delta\phi$ es la desviación de los parámetros de control del sistema actual. De manera que se puede obtener el siguiente ambiente en el tiempo $t+1$ como sigue:

$$f(x, \phi, t+1) = f(x, \phi(t) \oplus \Delta\phi, t) \tag{2.14}$$

En relación a los tipos de cambios que pueden presentar los parámetros de control, los autores definieron los siguientes: paso pequeño (*small step change*), paso grande (*large step change*), aleatorio (*random change*), caótico (*chaotic change*), recurrente (*recurrent*

change), y recurrente con ruido (*recurrent change with noise*). La manera en que se definen matemáticamente estos tipos de cambios es como se muestra a continuación:

- Paso pequeño (T1):

$$\Delta\phi = \alpha \cdot \|\phi\| \cdot r \cdot \phi_{severity}$$

- Paso grande (T2):

$$\Delta\phi = \|\phi\| \cdot (\alpha \cdot sign(r) + (\alpha_{max} - \alpha) \cdot r) \cdot \phi_{severity}$$

- Aleatorio (T3):

$$\Delta\phi = N(0,1) \cdot \phi_{severity}$$

- Caótico (T4):

$$\phi(t+1) = A \cdot \left(\phi(t) - \phi_{min}\right) \cdot \left(1 - \frac{(\phi(t) - \phi_{min})}{\|\phi\|}\right)$$

- Recurrente (T5):

$$\phi(t+1) = \phi_{min} + \|\phi\| \cdot \frac{\sin\left(2\pi t P^{-1} + \varphi\right) + 1}{2}$$

- Recurrente con ruido (T6):

$$\phi(t+1) = \phi_{min} + \|\phi\| \cdot \frac{\sin\left(2\pi t P^{-1} + \varphi\right) + 1}{2} + N(0,1) \cdot noisy_{severity}$$

donde $\|\phi\|$ es el rango de cambio de ϕ, $\phi_{severity}$ es una constante que indica la severidad del cambio en ϕ, ϕ_{min} es el mínimo valor de ϕ, $noisy_{severity} \in [0,1]$ es la severidad del ruido utilizado. Por otro lado, $\alpha \in [0,1]$ y $\alpha_{max} \in [0,1]$ son valores constantes iguales a 0.04 y 0.1 respectivamente. En el tipo de cambio caótico, se emplea una función logística donde A es una constante positiva en en el intervalo $[1.0, 4.0]$, de tal forma que si ϕ es un vector, los valores iniciales de los elementos en ϕ deberían ser diferentes en el rango $\|\phi\|$. Además, P es el período para los cambios recurrentes, siendo φ la fase inicial, y r es un número aleatorio en el intervalo $[-1,1]$, la función $sign(x)$ devuelve 1 si x es positiva, -1 si es negativa y 0 en caso contrario. $N(0,1)$ denota una distribución normal unidimensional con media en 0 y desviación estándar 1.

Con el objetivo de simular problemas reales donde el número de variables (dimensiones del espacio de búsqueda) varían con el tiempo (ej.problemas de planificación o de distribución), los autores proponen también un tipo de cambio para las dimensiones del

problema:

- Dimensional (T7):

$$D(t+1) = D(t) + \Delta D(t)$$

donde $\Delta D(t) \in \{-1, 1\}$ es una variable inicializada en -1 y cambia de valor de la manera siguiente:

$$\Delta D(t+1) := \begin{cases} +1 & \text{si } D(t) = D_{min}, \\ -1 & \text{si } D(t) = D_{max}, \\ \Delta D(t) & \text{por el contrario} \end{cases} \tag{2.15}$$

aquí, D_{min} y D_{max} el número mínimo y máximo de dimensiones, respectivamente.

En relación al tipo de función empleada, el generador define dos grupos *Dynamic rotation peak benchmark generator (DRPBG)* y el *Dynamic composition benchmark generator (DCBG)*. En el primero se extiende a los generadores de Branke (1999b) y Morrison y De Jong (1999). En efecto, dado el problema $f(\mathbf{x}, \phi, t), \phi = (\mathbf{H}, \mathbf{W}, \mathbf{X})$, donde $\mathbf{H}, \mathbf{W}$ y $\mathbf{X}$ denotan la altura de los picos, su anchura y posición respectivamente. La función para este grupo de problemas se define como:

$$f(\mathbf{x}, \phi, t) = \max_{i=1}^{m} \frac{\mathbf{H}_i(t)}{1 + \mathbf{W}_i(t) \cdot \sqrt{\frac{1}{D} \sum_{j=1}^{D} \left(x_j - \mathbf{X}_j^i(t) \right)^2}}$$

donde m es el número de picos y D las dimensiones del espacio. Cuando ocurre un cambio en el ambiente, $\mathbf{H}$ y $\mathbf{W}$ varían según uno de los tipos de cambios definidos anteriormente. Sin embargo, la principal diferencia de este grupo de problemas es la utilización de una matriz de rotación que varía las coordenadas de los picos.

Con respecto al segundo grupo de problemas, la función queda definida de la siguiente forma:

$$F(\mathbf{x}, \phi, t) = \sum_{i=1}^{m} \left\{ w_i \cdot \left[f_i' \left(\frac{\mathbf{x} - \mathbf{O}_i(t) + O_{i,old}}{\lambda_i} \cdot \mathbf{M}_i \right) + \mathbf{H}_i(t) \right] \right\}$$

donde $\phi = (\mathbf{O}, \mathbf{M}, \mathbf{H})$, $F(\mathbf{x})$ es la función de composición, $f_i(\mathbf{x})$ es la i-ésima función básica que compone a $F(\mathbf{x})$. Además, m es el número de estas funciones, mientras que $\mathbf{M}_i$ es una matriz de rotación ortogonal empleada para rotar a las funciones $f_i(\mathbf{x})$. En relación a $O_{i,old}$, se trata del óptimo original de $f_i(\mathbf{x})$, esto es, sin tener en cuenta los cambios del problema.

2.4.3 Meta-heurísticas en ambientes dinámicos

Hasta ahora se han visto diversos aspectos relacionados con la optimización mediante procedimientos meta-heurísticos, y en todos los casos en el contexto de la optimización

estacionaria. Estos elementos son esenciales para comprender el área de la optimización dinámica. De hecho, la tendencia actual en las propuestas de nuevos métodos en este contexto es la adaptación (en términos de diseño) de paradigmas eficientes en el campo de la optimización estacionaria (Cruz et al, 2011). Estas adaptaciones son necesarias debido a los retos que imponen la dinámica de los problemas. Para el caso específico de las meta-heurísticas poblacionales estos retos son, en sentido general, los siguientes:

1. La detección de los cambios en el ambiente para poder reaccionar adecuadamente.
2. El mantenimiento de la diversidad necesaria para poder enfrentar nuevos escenarios.

Si la meta-heurística desarrolla la búsqueda basada en determinadas soluciones (individuos) que fueron seleccionados por su calidad (fitness), entonces la detección de los cambios resulta imprescindible para actualizar a esas soluciones. En caso contrario la búsqueda se guiará por soluciones obsoletas, y la convergencia de la población podría ser hacia *falsos* óptimos.

En cuanto al mantenimiento de la diversidad, este resulta necesario si la meta-heurística ha alcanzado un nivel importante de convergencia, y es incapaz por tanto de seguir al nuevo óptimo del problema. Esta situación ocurre por lo general en escenarios con alta frecuencia y baja severidad de los cambios. En la literatura esta dificultad ha sido identificada por varios autores. Por ejemplo, Blackwell la denomina *pérdida de diversidad* cuando se refiere al paradigma PSO (Blackwell, 2003, 2005). Por su parte, Jin y Branke (2005) lo denominan el *problema de la convergencia* al referirse a los algoritmos evolutivos.

Dentro de los enfoques existentes para la detección de cambios en ambientes dinámicos, quizás la manera más sencilla es la reevaluación de una o varias memorias (soluciones) del algoritmo con el objetivo de detectar inconsistencias en relación al fitness, véase por ejemplo el Algoritmo 2.4.

```
   // Guardar el fitness anterior de mem
1  Asignar f_temp ← f_mem;

   // Reevaluar para obtener el fitness actual
2  Asignar f_mem ← f(x_mem);

   // Chequear inconsistencia
3  si f_temp ≠ f_mem entonces
   |  // Ha ocurrido un cambio
4  |  retornar verdadero;
5  fin
   // No ha ocurrido un cambio
6  retornar falso;
```

Algoritmo 2.4: Ejemplo de detección de cambios mediante reevaluación de memorias.

Debido a su fácil implementación y alta efectividad en la mayoría de los escenarios este enfoque es el más difundido en la actualidad. Entre los trabajos que emplean este tipo de enfoque se encuentran: (Parrott y Li, 2004; Blackwell y Branke, 2006; Pelta et al, 2009; Novoa-Hernández et al, 2010, 2011, 2013a; du Plessis y Engelbrecht, 2012).

Otra forma de detectar los cambios en el ambiente puede lograrse mediante el monitoreo del comportamiento del algoritmo. Por ejemplo, en (Cobb, 1990) los cambios son detectados mediante el control del fitness promedio de las mejores soluciones en varias generaciones. Por su parte, Janson y Middendorf (2004) propone en un enfoque basado en el paradigma PSO, que cada enjambre se divida en tres subenjambres organizados jerárquicamente. De manera que los cambios son detectados cuando existe un cambio en la estructura de dicha jerarquía. En (Morrison, 2004) se estudian varios modelos, entre los que se encuentran: la detección de cambios a través de la diversidad de la población, y mediante la tasa de éxito del algoritmo. Una técnica más sofisticada fue aplicada por Richter (2009), donde los cambios son detectados mediante pruebas de hipótesis, cuyo objetivo es encontrar diferencias entre las distribuciones de las poblaciones en dos generaciones consecutivas.

Por otro lado, según Jin y Branke (2005) los enfoques para lidiar con el problema de la convergencia (mantenimiento de la diversidad) se pueden clasificar en cuatro grupos:

Diversidad durante la ejecución. La idea básica de este enfoque es aplicar sistemáticamente algún operador de variación que genere diversidad en la población. El reto es identificar cuánta diversidad sería necesaria de manera que no se afecte la convergencia del algoritmo. Ejemplos de este tipo de enfoque existen varios: los *random inmigrants* de Grefenstette (1992), los cuales son nuevos individuos generados aleatoriamente e insertados en la población cada cierto tiempo. Más reciente, las variantes Quantum y Charged PSO (Blackwell y Bentley, 2002b; Blackwell y Branke, 2004, 2006) en las que en cada iteración del algoritmo partículas quantum o cargadas se mueven alrededor de las clásicas, sin afectar la explotación de éstas últimas e intercambiando información mediante la actualización del mejor individuo de la población.

Diversidad después del cambio. Los esquemas agrupados en este enfoque asumen que el algoritmo cuenta con la diversidad suficiente para optimizar el problema en las ventanas de tiempo en el que el problema no varía, pero que se queda sin diversidad al final de cada etapa. En consecuencia el algoritmo es incapaz de seguir a los nuevos óptimos del problema. En este sentido, quizás la manera más simple (y más robusta) es reiniciar alguna parte de la población en el espacio de búsqueda (ej. PSO con reinicio (Hu y Eberhart, 2002)). Sin embargo, existen problemas en los que la severidad de los cambios no provoca grandes diferencias entre escenarios. Por tanto, conviene generar diversidad inteligentemente (e.g.

realizar algún tipo de reinicio local). Ejemplo de este tipo de diversidad son la hipermutación de Cobb (1990) donde la tasa de mutación es incrementada unas generaciones después del cambio. Otro ejemplo es el reinicio en cierto entorno a la mejor solución aplicada por (Novoa-Hernández et al, 2010) en un algoritmo basado en PSO.

Enfoques basados en memorias. En (Jin y Branke, 2005) se ha dicho que el reto de un EA en ambientes dinámicos es "reusar la información de los ambientes anteriores para acelerar la optimización después de los cambios". Motivados por este reto varios investigadores han desarrollado diferentes enfoques que emplean algún tipo de memoria como información que fue útil en tiempos anteriores (ej. determinados individuos, estrategias o parámetros). En particular, dos tipos de enfoques de memorias son identificados, *memoria implícita* y *memoria explícita*. En la primera, se incluye algún tipo de redundancia en la población con el objetivo de influir indirectamente en el ciclo evolutivo del EA. Algunos de los trabajos que emplean este tipo de memoria son Goldberg y Smith (1987); Dasgupta y Mcgregor (1992); Yang (2003). En la *memoria explícita* se almacena algún tipo de información para ser utilizada convenientemente en generaciones posteriores (Mori et al, 1996; Branke, 1999b; Pelta et al, 2009).

Enfoques multipoblacionales. En algunos problemas dinámicos multimodales existe una alternancia entre los óptimos locales con respecto al óptimo global. Con lo cual, el desplazamiento del óptimo global es mayor que la severidad con que se desplazan los óptimos locales. Por ese motivo, en varios trabajos se ha propuesto incluir múltiples poblaciones en el algoritmo para vigilar (explorar) de manera paralela el espacio de búsqueda. En este caso, la interacción entre las poblaciones es importante y la experiencia de algunas pudiera ser aprovechada por otras. Ejemplo de trabajos que utilizan este enfoque son (Branke et al, 2000; Parrott y Li, 2004; Mendes y Mohais, 2005; Blackwell y Branke, 2006; Brest et al, 2009; Novoa-Hernández et al, 2010, 2013a).

Si se analiza con detenimiento la clasificación anterior, es fácil ver que estas cuatro categorías no son excluyentes entre sí y por tanto es posible encontrar esquemas que pueden ser clasificados en más de una de estas categorías. Esto quizás se deba a que las dos primeras dicen cuando se aplica el esquema, mientras que las otras son dos maneras de realizar el cómo. Por ejemplo, el enfoque de memoria explícita propuesto por Branke (1999b) puede ser visto como una forma de generar diversidad después del cambio, mientras que las memoria implícita de (Yang, 2003) puede ser clasificada como una técnica de generación de diversidad durante la ejecución. En lo que sigue denominaremos a estas categorías como enfoques de adaptación dinámica.

Recientemente, Nguyena et al (2012) advierten que otro enfoque de adaptación "*es*

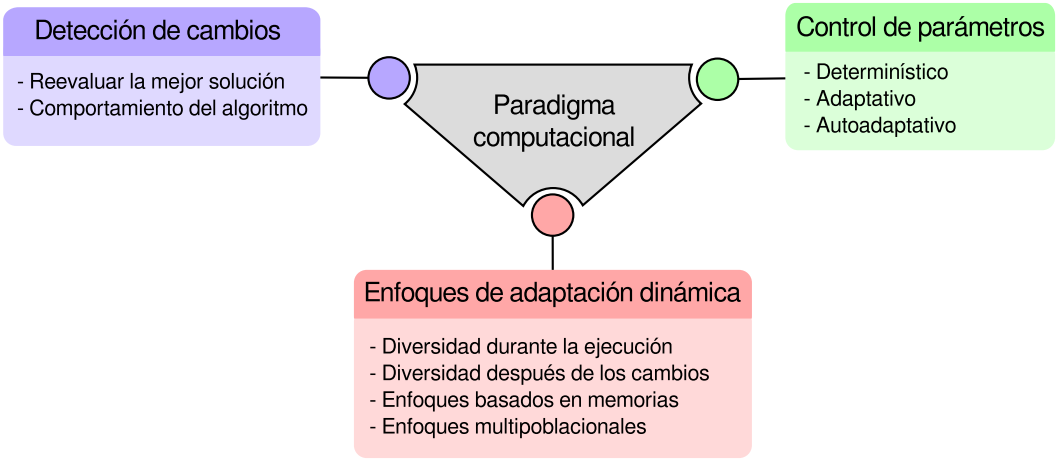

Figura 2.10 Meta-heurísticas *poblacionales* en ambientes dinámicos, como composición de cuatro elementos: paradigma computacional, técnica de detección de cambios, técnica de control de parámetros, y enfoques de adaptación dinámica.

hacer uso de mecanismos auto-adaptativos de los EAs y otras meta-heurísticas para lidiar con los cambios". En este caso se encuentran diversos trabajos como (Angeline et al, 1996; Weicker y Weicker, 1999; Boumaza, 2005; Schönemann, 2007; Brest et al, 2009; Novoa-Hernández et al, 2013a), los que se verán con más detalles en el Capítulo 5. Sin embargo, en nuestra opinión, estos enfoques auto-adaptativos forman parte del paradigma computacional empleado. Como se verá más adelante, las técnicas de control de parámetro (donde se encuentra precisamente la auto-adaptación) pueden aplicarse a otras partes de la meta-heurística, como por ejemplo, en los enfoques de adaptación dinámica. Es por eso que no consideramos a este tipo de auto-adaptación (presente en el paradigma computacional) como un enfoque de adaptación dinámica.

Con el objetivo de resumir esta sección, la Figura 2.10 muestra gráficamente los elementos que componen a las meta-heurísticas poblacionales en ambientes dinámicos. En primer lugar aparece el paradigma computacional el cual está dedicado a optimizar el problema en el período de tiempo en que no existen cambios en el ambiente. Ejemplos de estos paradigmas son los explicados en la Sección 2.2. Por otra parte, se encuentran las técnicas de detección de cambios que son necesarias en problemas donde no se asume conocidos tiempos en que ocurren los cambios. Adicionalmente, aparecen los enfoques de adaptación dinámica que constituyen una parte esencial de las meta-heurísticas y están orientados a tratar las consecuencias que provocan los cambios en el algoritmo. Por último, hemos considerado necesario incluir también las técnicas de control de parámetros, las cuales provienen del propio diseño de algunos paradigmas computacionales, como las Estrategias evolutivas.

2.4.4 Medidas de rendimiento en ambientes dinámicos

Un aspecto que siempre ha sido importante en el análisis de las meta-heurísticas en general lo constituye el establecimiento de medidas de rendimiento. En este contexto de la optimización en ambientes dinámicos, en la actualidad existen diversas propuestas de acuerdo al interés del investigador.

Una de las primeras medidas propuestas fue la *exactitud ponderada* de Mori et al (1996), la cual está dedicada caracterizar la *exactitud* del algoritmo de manera diferente en cada generación g de la ejecución:

$$perfAcc = \frac{1}{G}\sum_{g=1}^{G} \alpha_g \cdot accuracy^{(g)} \tag{2.16}$$

donde $\alpha^{(g)} \in \mathbb{R}$ es el peso de la exactitud en la generación g, y *accuracy* está definida por la siguiente expresión:

$$accuracy^{(g)} = \frac{\hat{f}^{(g)} - f_{min}}{f_{max} - f_{min}} \tag{2.17}$$

aquí, $\hat{f}$ es el fitness de la mejor solución del algoritmo en la generación g. Por otra parte, f_{max} y f_{min} son el fitness de la mejor y la peor solución en el espacio de búsqueda en la generación g. Es importante notar aquí que si $\alpha_g = 1\ \forall G$ entonces esta expresión se convierte en un promedio usual.

Por otro lado, dentro de las medidas más utilizadas se encuentran el error y el rendimiento fuera de línea (Branke, 1999b). Estas medidas están orientadas a medir el error y el costo (fitness) de la mejor solución del algoritmo a lo largo de la ejecución. Formalmente estas medidas vienen dadas por las siguientes expresiones:

$$offlineError = \frac{1}{T}\sum_{t=1}^{T} \left| f_{optim}^{(t)} - \hat{f}^{(t)} \right| \tag{2.18}$$

$$offlinePerf = \frac{1}{T}\sum_{t=1}^{T} \hat{f}^{(t)} \tag{2.19}$$

donde t es la evaluación de la función objetivo (tiempo) actual en el ambiente, de manera que T es el máximo de estas evaluaciones.

Otras medidas basadas en el fitness son la *adaptabilidad* y la *exactitud de la etapa* (Trojanowski y Michalewicz, 1999). Estas medidas se definen matemáticamente a través de

las siguientes expresiones:

$$adaptability = \frac{1}{\mathsf{K}} \sum_{k=1}^{\mathsf{K}} \left(\frac{1}{G_k} \sum_{g=1}^{G_k} \left| f_{optim}^{(g)} - \hat{f}^{(g)} \right| \right) \tag{2.20}$$

$$stageAccuracy = \frac{1}{\mathsf{K}} \sum_{k=1}^{\mathsf{K}} \left| f_{optim}^{(G_k)} - \hat{f}^{(G_k)} \right| \tag{2.21}$$

$$\tag{2.22}$$

donde K es el número de cambios en el ambiente, y G_k es el número de generaciones dentro del cambio (o etapa) k.

Por su parte, Weicker (2002) propone las medidas *estabilidad* y *reacción* con el objetivo de medir otros objetivos en ambientes dinámicos. Estas medidas se definen de la siguiente forma:

$$stability^{(G)} = \max\left\{0, acc^{(g-1)} - acc^{(g)}\right\} \tag{2.23}$$

$$\varepsilon - reactivity^{(G)} = \min\left\{g' - g \ : \ g < g' \leq G, \ \frac{acc(g')}{acc(g)} \geq (1-\varepsilon)\right\} \tag{2.24}$$

donde *acc* es una medida de exactitud (e.g. expresión 2.17). Además $\varepsilon \in \mathbb{R}_+$ y $g' \in \mathbb{N}$. Note que *stability* tomará valores en el intervalo $[0,1]$ si se emplea la expresión 2.17 para la *acc*, aunque en cualquier caso un valor cercano a 0 significa que el algoritmo es estable. En contraste, $\varepsilon - reactivity$ no expresa un valor relativo como la anterior y está dada en unidades de tiempo, específicamente en generaciones. El significado de su valor es el tiempo que necesitó el algoritmo para alcanzar una exactitud cercana a la exactitud de una generación previa (e.g. antes del cambio). En particular, un valor bajo implica una reacción alta.

Adicionalmente, Weicker (2002) investigó otras variantes de la *estabilidad* y *reacción*, pero empleando diferentes expresiones para la *exactitud* (*acc*). En particular, el autor consideró expresiones de la *distancia* de la mejor solución con respecto al óptimo del problema, así como una estimación de la exactitud definida en la expresión 2.17. Estas variantes serán analizadas con más detalle en el próximo capítulo.

Otro trabajo interesante es el de (Morrison, 2003), donde se propone una variante generacional del rendimiento fuera de línea de Branke (1999b). Esta medida denominada por su autor como *Collective fitness* viene dada por la expresión siguiente:

$$collectiveFitness = \frac{1}{G} \sum_{g=1}^{G} \hat{f}^{(g)} \tag{2.25}$$

Cuando se conoce que el problema es multimodal y se tiene información sobre la posición y valor de la función objetivo de los óptimos (picos) entonces se pueden aplicar medidas como las propuestas por (Branke, 2002; Bird y Li, 2007). La primera está orientada a medir cuantos picos son cubiertos por el algoritmo, mientras que la segunda mide el error en función del pico correspondiente al óptimo.

Más recientemente, algunos autores han propuesto medidas más sofisticadas. En este caso se incluyen la *degradación del fitness* (Alba y Sarasola, 2010), la cual se basa en la regresión lineal para estimar la exactitud total del algoritmo. Otro enfoque novedoso es el de (Shilane et al, 2009), en el que se aplican múltiples pruebas de hipótesis a las series temporales de varios algoritmos.

Finalmente, es importante destacar que otras medidas son posibles si se emplean otros elementos para la definición de estas. Por ejemplo, hasta el momento se han visto expresiones que promedian los resultados de la serie de mediciones realizadas en una ejecución. Sin embargo, aquí pudiera emplearse otra función de agregación menos susceptible a valores dispersos como es el caso de la *mediana*. En el próximo capítulo se analizarán esta y otras cuestiones.

2.5 Resumen

En este capítulo se han descrito los fundamentos teóricos de la presente investigación, los cuales quedan resumidos de la siguiente forma:

- La Soft Computing es un conjunto de técnicas y métodos que permiten tratar las situaciones prácticas reales de la misma forma que suelen hacerlo los seres humanos, es decir, en base a inteligencia, sentido común, consideración de analogías, aproximaciones, etc..
- Dentro de la Soft Computing las meta-heurísticas juegan un papel esencial como métodos de optimización aproximados que obtiene soluciones de alta calidad con un gasto bajo de recursos.
- Según los Teoremas *No Free Lunch* no existe una meta-heurística mejor que otra si se consideran sus rendimientos en todos los posibles problemas de optimización.
- Las meta-heurísticas *Optimización con enjambre de partículas* y *Evolución diferencial* se destacan por su efectividad y fácil implementación.
- Las técnicas de configuración de parámetros se clasifican en técnicas de ajuste y de control. En este último caso, las técnicas adaptativas y auto-adaptativas permiten un comportamiento inteligente del algoritmo en tiempo de ejecución.

- Los problemas dinámicos pueden ser definidos como un conjunto de problemas estacionarios que deben ser resueltos secuencialmente. En este caso, la función que rige la dinámica de los cambios define la transición entre problemas estacionarios.
- Varios tipos de cambios pueden presentarse en un problema dinámico, lo cual es un aspecto importante para caracterizar la complejidad del problema en cuestión.
- Dentro del estudio y experimentación con problemas dinámicos artificiales, los generadores de problemas juegan un papel importante por la posibilidad de obtener múltiples escenarios con diferentes dinámicas. Entre los generadores más empleados en la literatura se destacan el Movimiento de Picos(Branke, 1999b), y el generador GDBG (Li y Yang, 2008b).

Parte II

Contribuciones

3 Análisis experimental en ambientes dinámicos

En este capítulo se describen tres de las contribuciones de la presente investigación en el contexto del análisis experimental en ambientes dinámicos. Primero, se propone un marco de trabajo para organizar las medidas de rendimiento en ambientes dinámicos. Dicho marco de trabajo no solo tiene por objetivo resumir adecuadamente los principales progresos e identificar posibles desarrollos futuros, sino también estructurar intuitivamente las medidas para su implementación en tecnologías de desarrollo existentes. En una segunda parte, se revisan las principales metodologías estadísticas para analizar el rendimiento de los métodos meta-heurísticos en ambientes dinámicos. En este contexto se propone una nueva gráfica para visualizar de manera compacta los resultados de dos pruebas no paramétricas. Finalmente, se describe una herramienta para apoyar el proceso de experimentación en ambientes dinámicos. Dicha herramienta ha sido desarrollada con la tecnología Java y provee una interfaz gráfica que facilita la gestión de experimentos simples y multifactoriales. Adicionalmente, la herramienta está compuesta por un framework de aplicación que permite la incorporación de nuevos problemas, algoritmos, y medidas de rendimiento. Para validar la herramienta se consideró un caso de estudio basado en un algoritmo de la literatura.

3.1 Un marco de trabajo para las medidas de rendimiento en ambientes dinámicos

Sin lugar a dudas, la evaluación del rendimiento de un algoritmo en ambientes dinámicos ha sido siempre un tema de gran interés. En este contexto, las medidas de rendimiento juegan un papel importante, pues caracterizan de manera cuantitativa el rendimiento de los algoritmos. En la Sección 2.4.4 del pasado capítulo se mencionaron algunas de las medidas utilizadas en la actualidad. No obstante los progresos alcanzados en este campo de

investigación, actualmente es posible encontrar algunas dificultades. En ese sentido cabe mencionar la ausencia de un marco de trabajo actualizado que permita, al mismo tiempo: 1) organizar adecuadamente las medidas existentes, 2) identificar los progresos alcanzados y posibles trabajos futuros, y 3) favorecer la implementación de las medidas en tecnologías de desarrollo actuales.

Aunque pocos, existen algunos autores que han tratado esta dificultad en el pasado. Por ejemplo, Weicker (2002) analiza las medidas propuestas por aquellos días, y propone una clasificación utilizando dos categorías: información del algoritmo e información del problema. En el primer caso existen dos posibilidades:

- basadas en el fitness,
- basadas en el genotipo (o fenotipo).

En cuanto a la información del problema, las medidas pueden basarse en:

- la posición del óptimo (que es el caso de algunos escenarios artificiales)
- el fitness del óptimo,
- ninguna información.

Una de los beneficios de esta clasificación es la posibilidad de identificar el tipo de escenario en que puede aplicarse una determinada medida, esto es, si se tiene en cuenta la categoría *información del problema*. Otro aspecto interesante es que esta clasificación puede ser aplicada en el contexto de la optimización estacionaria, ya que sus categorías no hacen alusión a detalles como el tiempo o la naturaleza dinámica del problema.

Weicker también organiza las medidas de acuerdo al objetivo de un algoritmo en ambientes dinámicos. En ese sentido, el autor establece que un algoritmo debe cumplir con los objetivos siguientes:

- *exactitud*: obtener soluciones precisas,
- *estabilidad*: mantener un nivel de exactitud similar entre cambios,
- *reactividad*: recuperar rápidamente la precisión obtenida antes del último cambio.

Más recientemente, en una revisión sobre optimización evolutiva en ambientes dinámicos, Nguyena et al (2012) organiza las medidas en dos tipos: *basadas en la optimalidad*, y *basadas en el comportamiento*. En el primer caso se encuentran las medidas dedicadas a evaluar la capacidad de los algoritmos en la búsqueda de soluciones con el mejor valor de fitness, o aquellas cercanas al óptimo global. En particular, este tipo de medidas está compuesto por las siguientes categorías:

- basadas en la mejor solución de la generación,

- error y rendimiento fuera de línea modificados,
- mejor error antes del cambio,
- exactitud de la optimización,
- marcas normalizadas,
- basadas en la distancia (sin tener en cuenta el fitness).

Por otro lado, las *medidas basadas en el comportamiento* son aquellas que evalúan determinados comportamientos útiles del algoritmo en ambientes dinámicos. Ejemplo de tales comportamientos son el mantenimiento de la diversidad; recuperarse rápidamente después de una *caída*, entre otros. Como bien observan los autores, estas medidas son empleadas por lo general, como complementos de las basadas en la optimalidad para estudiar el comportamiento particular de los algoritmos. Específicamente, las medidas basadas en el comportamiento son agrupadas a través de las siguientes categorías:

- basadas en la diversidad,
- rendimiento después de los cambios,
- velocidad de convergencia después de los cambios,
- degradación del fitness a lo largo del tiempo.

Como puede verse, este marco de trabajo permite describir a las medidas de manera más específica que Weicker (2002), y permite una fácil exposición de los progresos existentes (posteriores en su gran mayoría a Weicker (2002)). Al mismo tiempo, este marco de trabajo resulta más general que el primero pues incluye de manera explícita las medidas basadas en el comportamiento del algoritmo.

Teniendo en cuenta los tres requerimientos mencionados al comienzo de la presente sección, se puede ver que estos marcos de trabajo cumplen parcialmente con dichos requerimientos. Por ejemplo, el de Weicker (2002) organiza las medidas de manera adecuada pero sin tener en cuenta otros elementos distintivos como el tiempo de medición.

No obstante los beneficios derivados de estas investigaciones, en nuestra opinión ambas poseen algunas limitaciones importantes, pues no permiten identificar:

1. el tiempo de medición, esto es, en que momentos relevantes se realizan las mediciones de los datos (ej. evaluaciones, generaciones, generación antes del cambio, etc.),
2. tipo de función de agregación empleada para caracterizar una ejecución del algoritmo (ej. media, mediana, etc.),
3. si la medida asume conocida (o no) algunos elementos de la dinámica del problema (ej. cuando ocurren los cambios, la severidad y frecuencia de los mismos, etc.).

Estos elementos resultan importantes si se tienen en cuenta los tres requerimientos mencionados al principio de la presente sección. Con el objetivo de superar estas dificultades, en

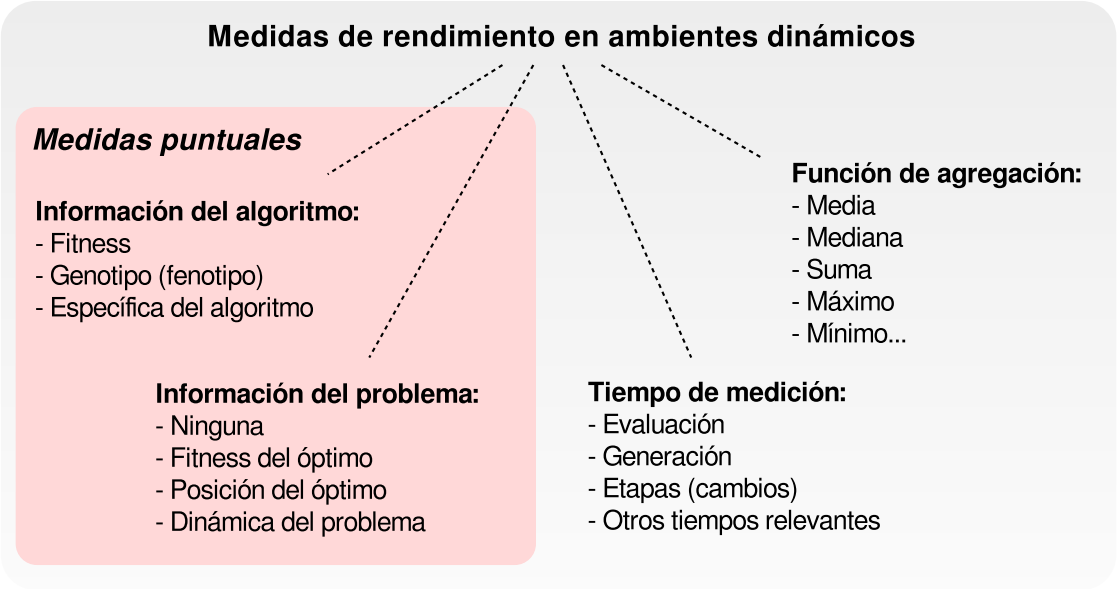

Figura 3.1 Marco de trabajo propuesto para las medidas de rendimiento en ambientes dinámicos.

la presente investigación se propone el marco de trabajo de la Figura 3.1. Como se aprecia, en esta organización se incluyen las dimensiones propuestas por Weicker (información del problema y del algoritmo), las cuales hemos agrupado en lo que hemos denominado *medidas puntuales*. Por medidas puntuales asumiremos la expresión matemática que se emplea para realizar la medición, en un tiempo determinado dentro de una ejecución específica del algoritmo. Más adelante, se mencionarán algunas de las medidas puntuales más empleadas.

Volviendo al marco de trabajo propuesto, se puede ver también que se han agregado las dimensiones *tiempo de medición* y la *función de agregación*. En particular, a la dimensión *información del problema* se le ha añadido la *dinámica del problema* con la intención de incluir aquellos problemas en los que se asuman conocido este tipo de información (e.g. cuando ocurren los cambios, cuáles y cómo varían los elementos del modelo, etc.). Note que esto último no implica necesariamente que se supongan conocidas otras informaciones como la posición o el fitness del óptimo global del problema, lo cual puede ser el caso de algunos escenarios reales.

Para ilustrar como utilizar el marco de trabajo propuesto, en lo que sigue se realizará una revisión sobre las medidas puntuales y de rendimiento empleadas en la actualidad.

3.1.1 Medidas puntuales

Según Weicker, cada medida de rendimiento responde a lo que sus autores consideran como el objetivo de un algoritmo en ambientes dinámicos, los cuales se pueden resumir en tres

objetivos fundamentales:

1. La exactitud o precisión (accuracy) del algoritmo (e.g. en función de las soluciones encontradas).
2. La estabilidad o capacidad del algoritmo de mantener un grado de exactitud aceptable a pesar de los cambios.
3. La reacción o tiempo que necesita el algoritmo para recuperar, de manera aproximada, la exactitud de etapas anteriores.

Aunque no de manera explícita, el trabajo de Weicker permite concluir que todas las medidas existentes en la literatura, y que intentan caracterizar a los algoritmos en función de los objetivos anteriores, se basan en lo que el investigador asuma como exactitud de manera puntual, esto es, por unidad de tiempo. Estas medidas puntuales brindan *fotogramas* valiosos del rendimiento del algoritmo en un determinado momento, y en su conjunto permiten caracterizar el rendimiento a nivel de ejecución. En consecuencia, las diferencias entre las distintas medidas propuestas no solo está en la elección de una determinada medida puntual, sino también en el tiempo de muestreo (medición) y en la función de agregación empleada para resumir estos datos puntuales.

Para una mejor comprensión de la notación utilizada en esta sección, asumiremos que un individuo (solución) de la población $\mathscr{P}^{(t)}$ del algoritmo se define formalmente de la siguiente forma:

$$\mathbf{y}_i = \langle \mathbf{x}_i, f_i \rangle \tag{3.1}$$

donde $\mathbf{x}_i \in \mathbb{R}^D$ es el vector solución y $f_i = f(\mathbf{x}_i)$ es el valor del fitness (función objetivo) correspondiente. De manera similar, el óptimo global del problema estaría compuesto por el vector solución $\mathbf{x}^*$ y el fitness f_{optim}.

Ejemplos de estas medidas, que en lo adelante llamaremos *puntuales*, son el *mejor fitness*, el *fitness promedio* (de la población), y la *exactitud* (Feng et al, 1997), expresiones 3.2, 3.3, 3.4, respectivamente. Note que todas están marcadas intencionalmente por un tiempo t.

$$bestFitness^{(t)} = \max_{i=1,\ldots,|\mathscr{P}^{(t)}|} f_i \tag{3.2}$$

$$avgFitness^{(t)} = \frac{1}{|\mathscr{P}^{(t)}|} \sum_{i=1}^{|\mathscr{P}^{(t)}|} f_i \tag{3.3}$$

$$accuracy^{(t)} = \frac{bestFitness^{(t)} - f_{min}}{f_{max} - f_{min}} \tag{3.4}$$

donde $\mathbf{y}_i$ es el i-ésimo individuo de la población del algoritmo $\mathscr{P}^{(t)}$, mientras que f_{max} y f_{min} representan los valores máximo y mínimo de la función objetivo del problema, respectivamente. En particular f_{max} es el valor de la función objetivo en el óptimo global del problema. Note que en esta última medida la información sobre el máximo y el mínimo del problema puede no estar disponible en determinados escenarios, tales como los reales. Para resolver esta dificultad, Weicker propone como estimación de esta medida, la exactitud dentro de una ventana de tiempo w:

$$windowAcc^{(t)} = \max\left\{ \frac{f(\mathbf{x}_i) - f_{worst}^{(w)}}{f_{best}^{(w)} - f_{worst}^{(w)}} \quad : i = 1, ..., |\mathscr{P}^{(t)}| \right\} \tag{3.5}$$

donde w es una unidad de tiempo (evaluación o generación) del algoritmo (e.g. después de ocurrir un cambio), además:

$$\begin{aligned} f_{best}^{(w)} &= \max\Big\{ f(\mathbf{x}_i) \quad : \quad i = 1, ..., |\mathscr{P}^{(t')}|,\ t - w \leq t' \leq t \Big\} \\ f_{worst}^{(w)} &= \min\Big\{ f(\mathbf{x}_i) \quad : \quad i = 1, ..., |\mathscr{P}^{(t')}|,\ t - w \leq t' \leq t \Big\} \end{aligned}$$

Otras medidas puntuales basadas en el valor de la función objetivo y no analizadas en los trabajos de Weicker (Weicker, 2002, 2003) son el *mejor error*, y el *error promedio*.

$$\begin{aligned} bestError^{(t)} &= \left| f(\mathbf{x}^*) - bestFitness^{(t)} \right| \\ avgError^{(t)} &= \frac{1}{|\mathscr{P}^{(t)}|} \sum_{i=1}^{|\mathscr{P}^{(t)}|} \left| f(\mathbf{x}^*) - f(\mathbf{x}_i) \right| \end{aligned}$$

De manera similar a las anteriores, existen otras medidas puntuales basadas en el vector posición de las soluciones, esto es, a diferencia de las anteriores que emplean el valor de la función objetivo. Obviamente, las basadas en la posición suelen ser más exactas que las basadas en el fitness, pero suponen conocida la solución óptima del problema. En este grupo se encuentran la mejor distancia (Weicker, 2003) y la distancia del *centroide* de la población (Salomon y Eggenberger, 1997), las cuales vienen dadas por las expresiones siguientes:

$$bestDist^{(t)} = \max\left\{ \frac{\delta_{max} - \delta(\mathbf{x}^*, \mathbf{x}_i)}{\delta_{max}} \ : \ i = 1, ..., \left|\mathscr{P}^{(t)}\right| \right\} \tag{3.6}$$

$$centerDist^{(t)} = \frac{\delta_{max} - \delta(\mathbf{x}^*, \overline{\mathbf{y}})}{\delta_{max}} \quad \text{con} \quad \overline{\mathbf{x}} = \frac{1}{|\mathscr{P}^{(t)}|} \sum_{i=1}^{|\mathscr{P}^{(t)}|} \mathbf{x}_i \tag{3.7}$$

aquí $\delta(\mathbf{x}_i, \mathbf{x}_j)$ es una función que devuelve la distancia euclidiana entre los vectores $\mathbf{x}_i$ y $\mathbf{x}_j$, mientras que δ_{max} representa la máxima distancia (extensión) del espacio de búsqueda.

3.1.2 Medidas de rendimiento

A partir de las medidas dadas anteriormente, diversos autores han propuesto otras más sofisticadas para caracterizar el rendimiento de los algoritmos en una ejecución. Dos aspectos son fundamentales en la definición de estas últimas: el tiempo de medición y la función de agregación para obtener el valor final de cada ejecución. Por ejemplo, Mori et al (1996) emplea una versión ponderada de la exactitud que promedia esta medida puntual teniendo en cuenta las generaciones G (iteraciones) del algoritmo:

$$perfAcc = \frac{1}{G} \sum_{g=1}^{G} \alpha_g \cdot accuracy^{(g)} \tag{3.8}$$

donde $\alpha_g \in \mathbb{R}$ es el peso de la exactitud acc_t en la generación g. Note que si $\alpha_g = 1 \; \forall t$ entonces la expresión 3.8 es un promedio exacto. Básicamente, la intención de los autores al incluir estos pesos fue ponderar determinadas generaciones durante la ejecución (ej. después de detectado el cambio o aquellas donde el óptimo se alcanza de manera exacta: $accuracy^{(g)} = 1$).

Con el objetivo de obtener el rendimiento del algoritmo a lo largo de la ejecución, Branke (1999b) propone el *error* y el *rendimiento fuera de línea*, los cuales son promedios (considerando evaluaciones) de las medidas puntuales $bestError$ y $bestFitness$, respectivamente. Estas medidas están definidas formalmente a través de las expresiones 3.9 y 3.10.

$$offlineError = \frac{1}{T} \sum_{t=1}^{T} bestError^{(t)} \tag{3.9}$$

$$offlinePerf = \frac{1}{T} \sum_{t=1}^{T} bestFitness^{(t)} \tag{3.10}$$

Siguiendo un objetivo ligeramente diferente e interesado en la adaptación de la población durante la ejecución, (De Jong, 1975) propone el *rendimiento en línea* el cual se define según la expresión 3.11.

$$onlinePerf^{(G)} = \frac{1}{G} \sum_{g=1}^{G} avgFitness^{(g)} \tag{3.11}$$

aquí g representa una generación (iteración) puntual durante la ejecución y G es la cantidad

máxima de estas generaciones.

Otras medidas similares son propuestas por Trojanowski y Michalewicz (1999). La diferencia radica en que los autores consideran el rendimiento en función de las generaciones y no de las evaluaciones. Las medidas propuestas son denominadas por los autores como *adaptabilidad* y el *exactitud*, ambas definidas por las ecuaciones 3.12 y 3.13.

$$adaptability^{(K)} = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{G_k}\sum_{g=1}^{G_k} bestError^{(g)}\right) \tag{3.12}$$

$$stageAccuracy^{(K)} = \frac{1}{K}\sum_{k=1}^{K} bestError^{(G_k)} \tag{3.13}$$

donde K es la cantidad de etapas (número de cambios) del problema, y G_k es el número de generaciones dentro de la etapa k. Quizás el aspecto más interesante de estas propuestas es que asumen un enfoque diferente al de Branke y DeJong, que consideran o bien todas las evaluaciones o generaciones, aquí la ejecución del algoritmo es dividida en etapas. En particular, *stageAccuracy* establece que el objetivo no es encontrar la mejor solución en todo momento, sino antes de cada cambio, específicamente en la última generación.

Por su parte, las medidas de *estabilidad* y *reacción* de Weicker estarían definidas de la siguiente forma:

$$stability^{(G)} = \max\left\{0, acc^{(g-1)} - acc^{(g)}\right\} \tag{3.14}$$

$$\varepsilon - reactivity^{(G)} = \min\left\{g' - g \;:\; g < g' \leq G,\ \frac{acc(g')}{acc(g)} \geq (1-\varepsilon)\right\} \tag{3.15}$$

donde $acc(g-1)$ y $acc(g)$ son dos medidas puntuales (e.g. cualquiera de las definidas en la sección anterior) correspondientes a las generaciones $g-1$ y g respectivamente. Además $\varepsilon \in \mathbb{R}_+$ y $g' \in \mathbb{N}$. Note que *stab* tomará valores en el intervalo $[0,1]$ de usarse la expresión 3.4 para la exactitud, en cualquier caso un valor cercano a 0 significa que el algoritmo es estable. En contraste, $\varepsilon - reactivity$ no expresa un valor relativo como la anterior y está dada en unidades de tiempo, específicamente en generaciones. El significado de su valor es el tiempo que necesitó el algoritmo para alcanzar una exactitud cercana a la exactitud de una generación previa (e.g. antes del cambio). En particular, un valor bajo implica una reacción alta.

Estas medidas son particularmente especiales pues al depender de lo que se considere como exactitud (*acc*) el investigador puede obtener diferentes variantes de las mismas. Esto fue aprovechado convenientemente por Weicker, el cual condujo un análisis experimental en (Weicker, 2002, 2003) para determinar cuan informativas resultan estas medidas variando la medida puntual que define a la exactitud. En ese sentido, consideramos que el estudio de

Weicker es un progreso importante en el área de las medidas de rendimiento en ambientes dinámicos por tres razones fundamentales: 1) por primera vez se derivan medidas en función de los objetivos de un algoritmo en ambientes dinámicos, 2) la estabilidad y la reacción quedan definidas en función de lo que el investigador defina como exactitud, por lo que pueden ser utilizadas en diversos contextos, y 3) se brinda una metodología para estudiar estadísticamente cuán informativas pueden ser las medidas y que nivel de correlación existe entre ellas. El trabajo de Weicker ha sido empleado en diversas investigaciones posteriores (e.g. Alba et al, 2009; Alba y Sarasola, 2010), pero aún consideramos que es insuficiente su presencia en investigaciones actuales. En ese sentido, la tendencia es utilizar otras medidas quizás menos informativas como el error fuera de línea, y el error antes del cambio.

A modo de resumen de la revisión anterior, en la Tabla 3.1 se organizan las medidas de rendimiento en ambientes dinámicos a través del marco de trabajo propuesto.

Como se aprecia en esta tabla, existen escenarios en los que, al asumirse conocido el fitness del óptimo del problema, es posible aplicar medidas como el error y rendimiento fuera de línea. En particular, estas medidas también necesitan conocer, al menos según la definición de Branke (2002), cuando ocurren los cambios en el ambiente, para actualizar convenientemente el fitness de la mejor solución del algoritmo. De lo contrario, tanto el error como el rendimiento fuera de línea pueden esta considerando falsos valores. Esto ocurre en escenarios donde los cambios afecten directamente a la función objetivo (e.g. la altura de los picos). Lo ideal sería que la medida no empleara información alguna del problema, sin embargo esto implica al menos para el caso de la optimización en ambientes dinámicos, una disminución de la precisión en la medición. Incluso las estimaciones como la exactitud en una determinada ventana de tiempo, propuesta por Weicker (2002) (expresión 3.5), pueden resultar poco confiables si se tiene en cuenta que esta medida no utiliza información alguna del problema.

Por otro lado, note que existen casos en que la información que necesita las medidas depende de la requerida por para de la medida puntual utilizada. En este caso se encuentran la *estabilidad* y la *reacción* que dependen de la definición de la medida puntual *acc*. En cualquier caso, ambas necesitan conocer cuando ocurren los cambios para cada etapa del problema. Otro aspecto destacable en cuanto a la información utilizada por las medidas es el predominio de medidas basadas en el fitness en relación con las basadas en la posición. En nuestra opinión, esta tendencia se justifica por una razón fundamental: la posición del óptimo del problema solo está disponible en escenarios artificiales, por lo que la aplicación de estas medidas se verá limitada a este contexto.

En cuanto al resto de las dimensiones de la clasificación propuesta, estas nos brindan otros detalles técnicos importantes. Por ejemplo, la dimensión *agregación* expresa el tipo de función empleada para resumir los datos de las mediciones. Por ejemplo, existe una

Tabla 3.1 Organización de las medidas de rendimiento según el marco de trabajo propuesto.

Referencia	Medida de rendimiento	Información del algoritmo	Información del problema	Tiempo de medición	Función de agregación
(De Jong, 1975)	$onlinePerf^{(G)} = \frac{1}{G}\sum_{g=1}^{G}\left(\frac{1}{\mu}\sum_{i=1}^{\mu} f_i^{(g)}\right)$	Fitness de la población	–	Generaciones	Media
(Branke, 2002)	$offlineError^{(T)} = \frac{1}{T}\sum_{t=1}^{T} bestError^{(t)}$	Fitness	Fitness, Dinámica del problema	Evaluaciones	Media
(Branke, 2002)	$offlinePerf^{(T)} = \frac{1}{T}\sum_{t=1}^{T} bestFitness^{(t)}$	Fitness	Dinámica del problema	Evaluaciones	Media
(Trojanowski y Michalewicz, 1999)	$stageAccuracy^{(K)} = \frac{1}{K}\sum_{k=1}^{K} bestError^{(Gk)}$	Fitness	Fitness, Dinámica del problema	Etapa, Generaciones	Media
(Trojanowski y Michalewicz, 1999)	$adaptability^{(K)} = \frac{1}{K}\sum_{k=1}^{K}\left[\frac{1}{G_k}\sum_{g=1}^{G_k} bestError^{(k)}\right]$	Fitness	Fitness, Dinámica del problema	Etapa, generaciones	Media
(Weicker, 2002)	$windowAcc = \frac{f_{best}-f_{W,worst}}{f_{W,best}-f_{W,worst}}$	Fitness	–	Etapa, Generaciones	Media
(Weicker, 2002)	$bestDistAccuracy^{(G)} = \frac{1}{G}\sum_{g=1}^{G} bestDist^{(g)}$	Genotipo	Posición del óptimo	Generaciones	Media
(Weicker, 2002)	$stability^{(G)} = \max\left\{0, acc^{(g-1)} - acc^{(g)}\right\}$	Depende de *acc*	Depende de *acc*	Etapa, Generaciones	Media
(Weicker, 2002)	$reactivity_{\varepsilon}^{(G)} = \min\left\{g' - g \ : \ \frac{acc(g')}{acc(g)} \geq (1-\varepsilon)\right\}$	Depende de *acc*	Depende de *acc*	Etapa, Generaciones	Media
(Morrison, 2003)	$collectiveFitness = \frac{1}{G'}\sum_{i=1}^{G'} bestFitness^{(g)}$	Fitness	–	Generaciones	Media
(Schönemann, 2007)	$bestPerf^{(G)} = \{bestFitness^{(g)} \ : \ g = 1, ..., G\}$	Fitness	–	Generaciones	Mediana
(Li y Yang, 2008a)	$meanAbsoluteError^{(T)} = \frac{1}{K}\sum_{k=1}^{K} bestError^{(Tk)}$	Fitness	Fitness, Dinámica del problema	Etapa, Evaluaciones	Media
(Shilane et al, 2009)	t-estadígrafos basados en $bestFitness^{(g)}$	Fitness	–	Generaciones	Media
(Alba y Sarasola, 2010)	Regresión lineal basada en $bestFitness^{(g)}$	Fitness	–	Generaciones	Media

tendencia a emplear la media para caracterizar una ejecución (De Jong, 1975; Trojanowski y Michalewicz, 1999; Branke, 2002; Weicker, 2002), sin embargo, otros autores argumentan que este estadígrafo es sensible a valores dispersos. En contraste, emplean otros más robustos como la mediana, tal es el caso de (Schönemann, 2007) que aplica esta función de agregación no para caracterizar una ejecución en particular, sino para obtener el una serie temporal partir de varias ejecuciones.

La dimensión *tiempo de medición* nos permite definir el nivel de la medición. En ese sentido, como se aprecia en la clasificación propuesta existen medidas que se toman los datos provenientes de todas las evaluaciones o generaciones de la ejecución, como es el caso del error y el rendimiento fuera de línea de Branke (2002), mientras que otras medidas como *stageAccuracy* (Trojanowski y Michalewicz, 1999) y *meanAbsoluteError* (Li y Yang, 2008b) solo emplean la última generación (evaluación) antes de la ocurrencia de cada cambio. Note que ambas medidas utilizan expresiones similares, pero con tiempos de medición distintos. A diferencia de las fuera de línea de Branke, estas medidas asumen que el objetivo del algoritmo es encontrar la mejor solución en cada etapa (problema estacionario) k de manera secuencial.

Otras medidas más sofisticadas quedan organizadas fácilmente por la clasificación propuesta, como se aprecia en los casos de las *hipótesis múltiples* de Shilane et al (2009) y la *degradación del fitness* de Alba y Sarasola (2010) que se basan en el $bestFitness^{(g)}$ de cada generación.

Es de notar también que la clasificación permite identificar posibles medidas aún no propuestas y que pudieran ser quizás más representativas en determinados escenarios. Por ejemplo, la mayoría de las medidas analizadas emplean la *media* como función de agregación, ¿qué sucedería si se empleara la mediana? Asimismo, la tendencia es realizar las mediciones en cada evaluación o generación de la ejecución, en ese sentido diferentes medidas pueden obtenerse si se realizan en las evaluaciones o generaciones antes del cambio. Nuestros trabajos futuros estarán orientados al análisis de estas posibles combinaciones.

3.2 Análisis estadístico en ambientes dinámicos

Dada la naturaleza estocástica de los problemas dinámicos y las meta-heurísticas empleadas para su solución, el tratamiento estadístico de la información que generan las medidas de rendimiento juega un papel esencial.

En ese sentido sobresalen, aunque pocos, algunos trabajos pertenecientes al ámbito de la optimización estacionaria. Recientemente (García et al, 2009) analizan la aplicación de pruebas no paramétricas para la comparación de los algoritmos participantes en la sesión

especial sobre optimización continua del congreso CEC'2005. Es importante notar que esta investigación se puede ver como una extensión a la que realizó (Demsar, 2006) en el contexto de la minería de datos para la comparación de clasificadores en múltiples *datasets*. En el caso de la optimización en ambientes estacionarios (incluso dinámicos), este análisis puede extrapolarse sin problemas. Por ejemplo, asumiendo que los clasificadores son los algoritmos, mientras que los *datasets* son los problemas a optimizar.

En ambas investigaciones los autores concuerdan en que la aplicación de pruebas no paramétricas se justifica debido a dos razones fundamentales: 1) los supuestos de normalidad se violan con frecuencia en los datos, y 2) la cantidad de datos es insuficiente para aplicar pruebas paramétricas. Asimismo, en estos trabajos se distinguen dos tipos de análisis: a nivel de problema, y a nivel global (varios problemas). Estos análisis utilizan los resultados de una medida de rendimiento en particular. La diferencia entre estos tipos de análisis radica en que el primero emplea como datos los valores de las ejecuciones de los algoritmos en un problema determinado, mientras que en el segundo se usan los valores agregados de varios problemas. Obviamente el primer tipo de análisis está orientado a analizar el rendimiento de los algoritmos en un problema en específico, mientras que el segundo se puede ver como una análisis de robustez, ya que se emplean los resultados de los algoritmos en varios problemas, que por lo general, poseen características diferentes.

En el contexto de la optimización en ambientes dinámicos, el empleo de pruebas estadísticas para el análisis de los resultados es aún pobre. La técnica más común es la comparación directa de los resultados (Weicker y Weicker, 1999; Carlisle y Dozier, 2002; Boumaza, 2005; Blackwell y Branke, 2006; Schönemann, 2007; Wang et al, 2007; Li y Yang, 2008a, 2009; Brest et al, 2009; Chun-Kit y Ho-Fung, 2012). Asimismo persiste el uso (por lo general incorrecto) de pruebas paramétricas (Karaman et al, 2005; Yang, 2005, 2006; Fernandes et al, 2008; Du y Li, 2008; Tinos y Yang, 2008; Yang et al, 2010).

En la presente investigación se adoptará la metodología propuesta por (García et al, 2009). En particular el segundo tipo de análisis, el cual está orientado al estudio de los algoritmos en varios tipos de problemas. Esta metodología propone realizar primeramente las pruebas de Friedman e Iman-Davenport con la intención de detectar diferencias en general. En caso de que exista tales diferencias, se aplicarán en consecuencia pruebas post-hoc (eg. test de Holm) para determinar las diferencias entre cada par de algoritmos. Esta metodología es mostrada por la Figura 3.2. Note que la prueba de Friedman asigna como resultado adicional el rango (posición relativa) de los algoritmos. Estos rangos medios son indicativos de la calidad de los resultados de los algoritmos. Precisamente, tanto estos rangos medios como los resultados de las comparaciones múltiples a partir de pruebas post-hoc son empleados para analizar el rendimiento de los algoritmos. En ese sentido cabe indicar que una de las cuestiones más importantes dentro de este ámbito es como representar

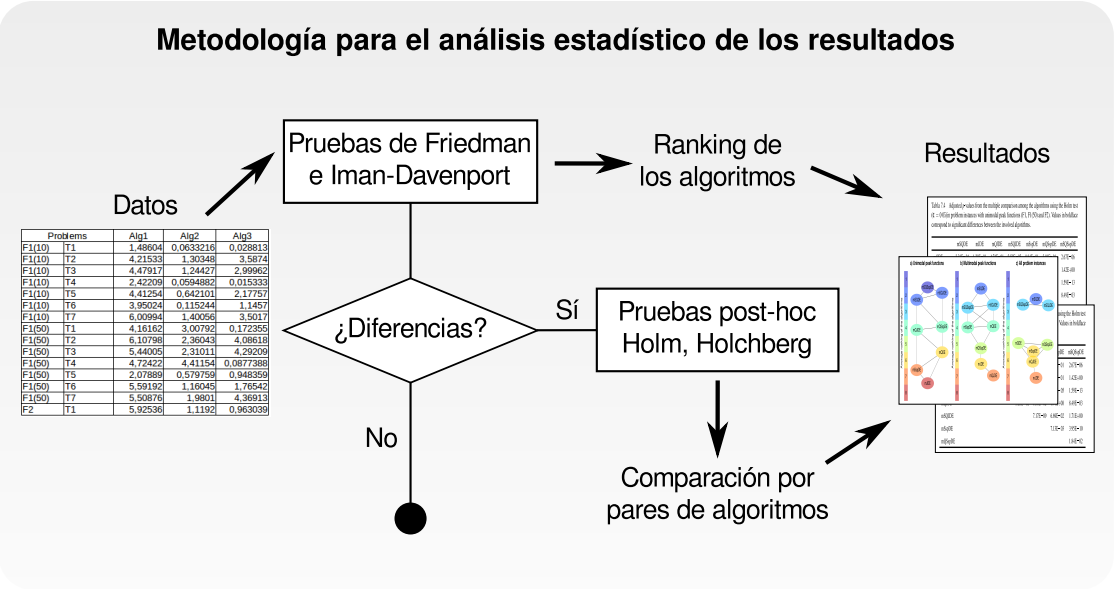

Figura 3.2 Esquema de la metodología seguida en la investigación para el análisis de los resultados de los experimentos.

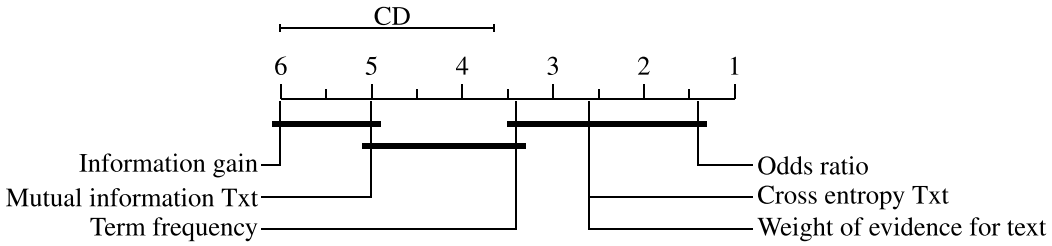

Figura 3.3 Gráfica propuesta por (Demsar, 2006) para representar los resultados de los rangos medios según el test de Friedman y las comparaciones múltiples entre métodos según test.

gráficamente los resultados de este análisis comparativo. Por ejemplo, (Demsar, 2006) propone utilizar la gráfica de la Figura 3.3, que muestra la relación entre los algoritmos basados en los rangos medios y en un estadígrafo conocido como *diferencia crítica*. En ese sentido, las líneas que conectan a los métodos indica que estos no son significativamente diferentes entre sí.

Similarmente, (García et al, 2009) proponen la gráfica de la Figura 3.4, la cual representa en barras, los rangos medios según Friedman, y en líneas discontinuas, la *diferencia crítica* de acuerdo al test post-hoc de Bonferroni-Dun. En este caso, a diferencia del análisis de (Demsar, 2006), la gráfica se centra en la relación del mejor algoritmo (algoritmo control) y el resto.

No obstante los avances logrados por estas investigaciones en esta dirección, en este trabajo proponemos la gráfica mostrada por la Figura 3.5. Como se aprecia, de esta forma

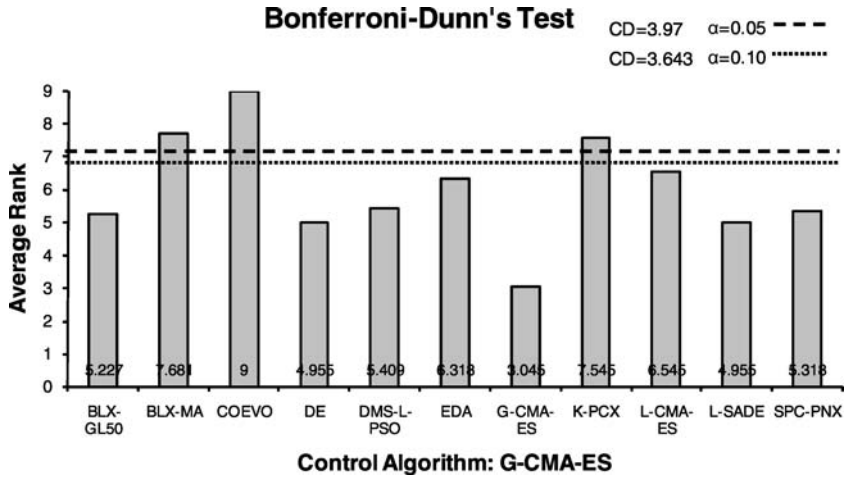

Figura 3.4 Gráfica propuesta por (García et al, 2009) para representar los resultados de los rangos medios según el test de Friedman y la diferencia crítica en relación a un algoritmo control (G-CMA-ES).

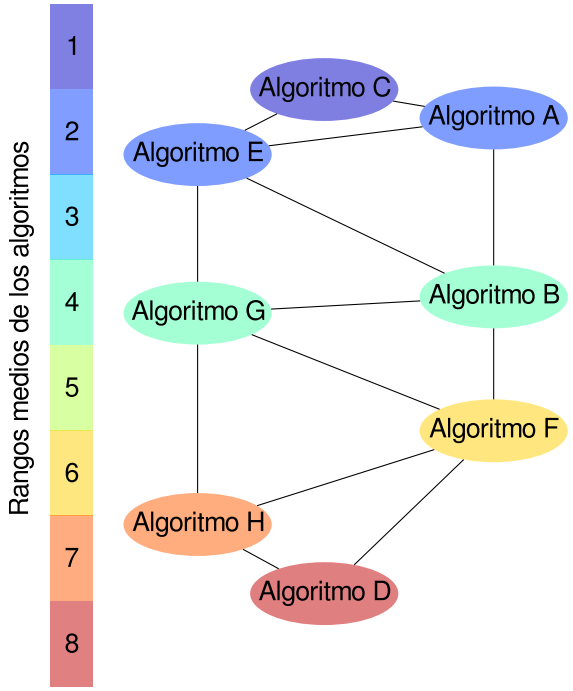

Figura 3.5 Gráfica propuesta para representar los rangos medios según la prueba de Friedman y las comparaciones múltiples a partir de la aplicación de pruebas post-hoc.

se ilustra de manera intuitiva y compacta, los resultados de ambos análisis: la prueba de Friedman e Iman Davenport, y las múltiples comparaciones entre los algoritmos según pruebas post-hoc. En particular, las comparaciones múltiples se aprecian a partir de los arcos que relacionan a los algoritmos. En este sentido, la presencia de los arcos indica que no existen diferencias significativas entre los algoritmos involucrados.

3.3 DynOptLAB: una herramienta para el análisis experimental en ambientes dinámicos

Experimentar significa variar deliberadamente las condiciones habituales de trabajo para encontrar mejores maneras de proceder y ganar al mismo tiempo un conocimiento más profundo sobre el comportamiento de productos y/o procesos (Prat Bartés y Tort-Martorell Llabrés, 2004). En muchos campos del saber los investigadores realizan experimentos, por lo general para descubrir algo acerca de un proceso o sistema en particular. Desafortunadamente, en muchas ocasiones estos experimentos se realizan sin un diseño previo, lo que impide obtener todos los resultados potenciales. Es por eso que los diseños de experimentos resultan más útiles que un simple experimento.

Por Diseño de Experimentos se entiende aquella "*estrategia de planificación de experimentos tal que las conclusiones válidas y relevantes puedan ser eficiente y económicamente enriquecidas*" (Fernández, 2004). En este sentido se aprecia que el objetivo es obtener conclusiones válidas y relevantes con eficiencia y economía, utilizando como vía para ello la realización de varios experimentos de forma iterativa, pero que hayan sido previamente planificados en la medida de lo posible. También se le define habitualmente como una prueba o serie de pruebas donde se inducen cambios deliberados en los factores para observar y analizar los cambios en la respuesta de salida. A continuación se relacionan algunos de los conceptos relacionados con el Diseño de Experimentos:

- **Respuestas.** Es el nombre genérico que se da a la característica estudiada. Representa la salida de un experimento.
- **Factores.** Se designa de esta forma a las variables que se considera puede afectar a la respuesta, y por tanto, se incluyen en el plan de experimentación.
- **Niveles.** Son los valores que toma un factor en un determinado experimento.
- **Tratamiento.** Combinación de variantes y/o niveles de cada factor que se utiliza en una determinada prueba o experimento.

La metodología del diseño de experimentos estudia como realizar comparaciones lo más homogéneas posibles, para aumentar en consecuencia, la probabilidad de detectar cambios o identificar variables influyentes, o lo que es lo mismo, estudia el efecto que sobre una variable respuesta tienen un conjunto de otras variables que se pueden llamar variables experimentales, factores o tratamientos. En la literatura es posible encontrar numerosas metodologías para llevar a cabo diseños de experimentos, como por ejemplo las que se plantean en los trabajos de Montgomery (2000); Bartz-Beielstein y Preuss (2006). Sin embargo, por su fácil adaptación a la experimentación en ambientes dinámicos se decidió seleccionar para esta investigación la propuesta por Fernández (2004), que se muestra

seguidamente:

1. **Comprensión y formulación del problema.** Consiste en la determinación del problema a resolver y su formulación. Es necesario desarrollar todas las ideas sobre los objetivos del experimento.
2. **Elección de factores y niveles.** Se eligen los factores que variarán en el experimento, los intervalos de dicha variación y los niveles específicos a los cuales se hará el experimento. Debe considerarse la forma en que se controlarán los factores en los valores deseados y como se les medirá.
3. **Selección de la variable respuesta.** Se debe estar seguro que la respuesta que se va a medir realmente suministre información útil acerca del proceso de estudio.
4. **Elección del diseño experimental.** Aquí es muy importante tener en cuenta los principios básicos del diseño de experimento y el objetivo experimental. Estos elementos definen el método estadístico a seleccionar.
5. **Realización del experimento.** Debe hacerse de acuerdo a lo planificado, con las responsabilidades asignadas según las necesidades de tiempo y recursos establecidas.
6. **Análisis de los datos.** Se utiliza el método estadístico escogido. Generalmente se utilizan paquetes estadísticos incluidos en algún software.
7. **Conclusiones y recomendaciones.** A menudo es útil usar métodos gráficos para la presentación de resultados y las ejecuciones de seguimiento y pruebas de confirmación para validar las conclusiones del experimento.

El sistema informático que se propone en esta investigación está orientado a asistir al investigador en los pasos del procedimiento antes mencionado. En esencia, su principal objetivo es la gestión de la experimentación en ambientes dinámicos. Para ello, deberá brindar la posibilidad de incluir y seleccionar problemas, algoritmos y medidas de rendimiento. Además, permitir la elección y variación de parámetros relacionados, los cuales constituyen los factores y niveles del experimento. En relación a la variable de respuesta, el software deberá facilitar la selección de una medida de rendimiento (como las explicadas al comienzo del capítulo). Una vez definido el diseño del experimento, el software deberá efectuar el desarrollo del mismo (simulación). En consecuencia, los resultados serán visualizados con posibilidades de algún tipo de análisis estadístico acorde a estos.

DynOptLAB[1] fue programada sobre la tecnología Java, la cual es un lenguaje de alto nivel y multi-plataforma desarrollado por la compañía Sun Microsystems[2]. Como se mencionaba en la introducción, *DynOptLAB* está formada por dos componentes generales: un *framework* para la creación e inclusión de nuevos problemas y algoritmos, y una *interfaz*

[1] Dynamic Optimization Laboratory
[2] http://www.sun.com/

gráfica de usuario[3] para controlar la ejecución de los experimentos. Por lo que el usuario (investigador) solo debe preocuparse por la programación de sus propuestas, y si logra una buena generalización de las mismas, entonces será posible configurarlas a través de la aplicación visual. En las secciones siguientes se explican los detalles de estos dos componentes.

3.3.1 Framework

Un *Framework* dirigido a objetos (FDO) es el diseño reusable de un sistema que describe como debe descomponerse el mismo en un conjunto de objetos que interactúan. A diferencia de una simple arquitectura de software, un FDO está expresado en un lenguaje de programación y está basado en el dominio de un problema específico (Fayad et al, 1999). Básicamente, un FDO está compuesto por dos tipos de elementos principales: *puntos calientes* y *puntos congelados*. Los calientes representan código extensible a través de clases abstractas e interfaces, mientras que los congelados son funcionalidades que no pueden ser cambiadas por el usuario final y que definen la lógica del dominio del problema (en este caso, la experimentación en ambientes dinámicos). En ese sentido, con la idea de proveer al investigador de las facilidades necesarias en la programación de propuestas y problemas se implementó un FDO, cuyas clases se muestran en la Figura 3.6.

Las clases con una tonalidad más intensa representan los *puntos calientes* (ej. *Algorithm*, *DynamicOptimizationProblem* y *Measure*), mientras que las más claras son los *puntos congelados*. En este diagrama se han excluido otros métodos y atributos para una mejor comprensión. Note que a un experimento, definido por la clase *Experiment*, se le pueden asignar un problema, un algoritmo y un listado de medidas de rendimiento. Esta clase realiza varias ejecuciones de ese par problema-algoritmo, teniendo como variables de respuesta a las medidas consideradas. En particular, las medidas de rendimiento han sido diseñadas teniendo en cuenta el marco de trabajo propuesto en la sección anterior. Note en este sentido la presencia de los métodos *setMeasurementTime* y *setAggregationFunction* en la interfaz *Measure*, los cuales permiten establecer el tiempo de medición y la función de agregación respectivamente. Adicionalmente, se ha incluido el tipo de reporte de la medida, que puede ser *RUN* o *SERIES*. El primero indica que la medida caracteriza cada ejecución mediante un solo valor, obtenido a partir de la función de agregación definida. En el segundo caso, la medida no agrega los valores que va registrando, por lo que el resultado por cada ejecución es una serie temporal. Esta última forma de registrar los valores resulta en extremo útil en el análisis del comportamiento del algoritmo durante el tiempo.

[3]GUI, por sus siglas en inglés

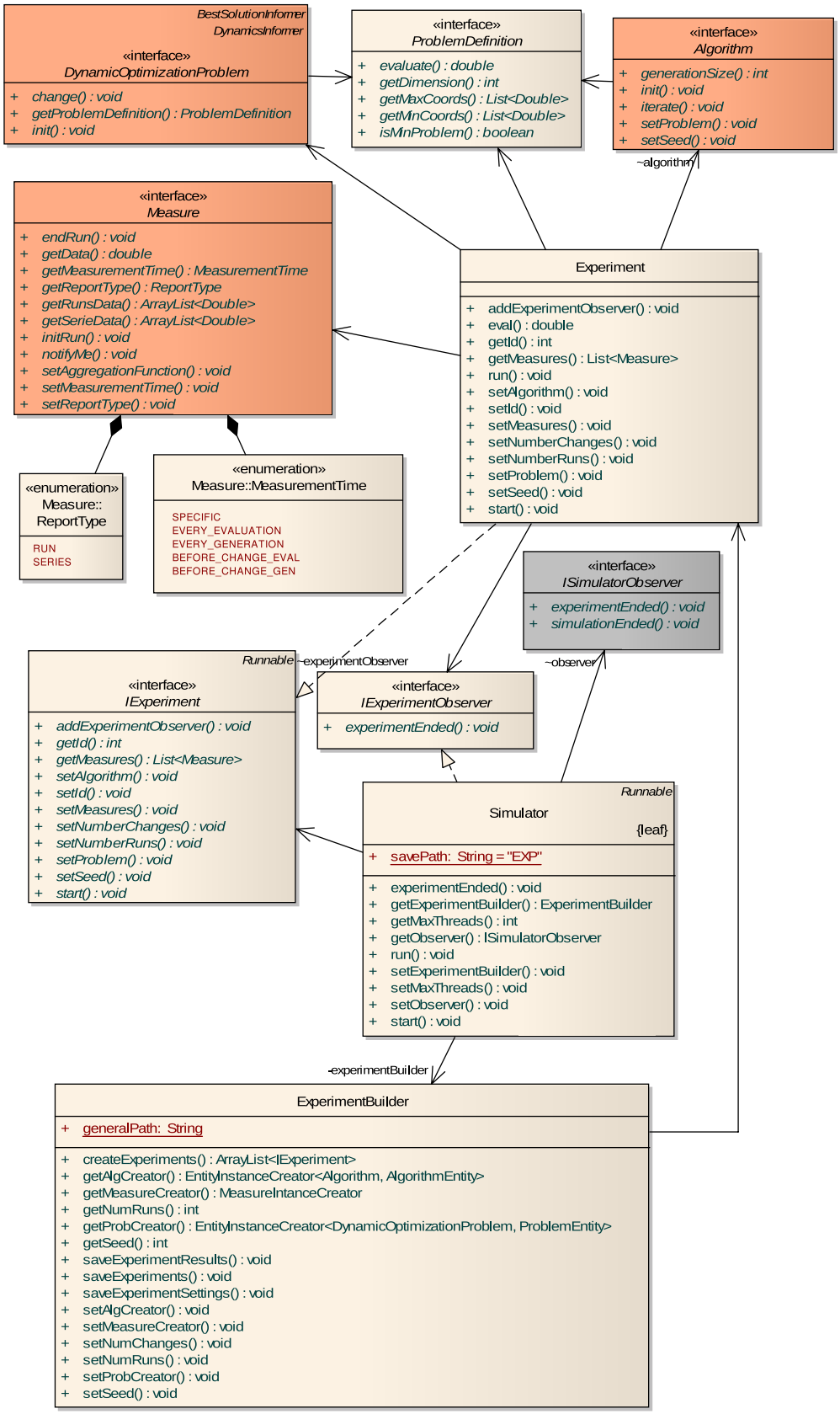

Figura 3.6 Diagrama de clases del *framework* incluido en *DynOptLAB*. Algunos métodos y atributos son excluidos para una mejor comprensión.

Resulta importante resaltar además, la presencia de la clase *Simulator*, que controla la ejecución de un número de experimentos de manera paralela. De esta forma, la aplicación aprovecha los núcleos (subprocesos) presentes en el procesador del ordenador donde se ejecuten los experimentos. Los experimentos se relacionan con la clase *Simulator* a través del patrón de diseño *Observer*, el cual es típico en los modelos basados en eventos de las tecnologías de desarrollo actuales. De esta forma, cuando un experimento termina su ejecución, este le informa a la instancia de la clase *Simulator*, y esta a su vez crea un nuevo *hilo* para ejecutar un nuevo experimento de los que se encuentren en la cola.

La clase *Simulator* se relaciona de dos maneras con la interfaz visual de usuario de *DynOptLAB*. Por un lado, mediante la clase *ExperimentBuilder*, la cual está dedicada a la creación de la cola de experimentos, y a guardar los resultados de las medidas, esto es, conforme van terminando los experimentos. La otra forma es a través del patrón de diseño *Observer*, que está presente en la implementación de la interfaz *ISimulationObserver*. En este sentido, una de las clases de la interfaz de usuario implementa la interfaz *ISimulationObserver* y de esta forma se mantiene informada del progreso de la simulación.

En sentido general, el investigador debe interactuar con las interfaces correspondientes a los problemas, medidas, y algoritmos. Los parámetros que desee definir como factores del experimento deberán ser declarados de tipo *public* con el objetivo de facilitar la asignación de los valores en tiempo de ejecución. Esta asignación se lleva a cabo mediante la tecnología incluída en el paquete `java.lang.reflect` del propio SDK de Java.

3.3.2 Aplicación visual

La interfaz gráfica de *DynOptLAB* es simple e intuitiva. Fue desarrollada sobre la biblioteca *SWT*[4] del proyecto *Eclipse*[5], la cual provee un conjunto de componentes (*widgets*) para construir interfaces gráficas en Java. El objetivo de la elección de esta tecnología es lograr que la aplicación propuesta mostrara un estilo visual consistente en todas las plataformas, esto es, idéntico a la forma en que se visualizan las aplicaciones nativas.

A través de cinco pestañas de una misma ventana es posible acceder a las principales funcionalidades del sistema. Estas funcionalidades están agrupadas en los módulos: *Problems*, *Algorithms*, *Measures*, *Experiments*, y *Results*. La Figura 3.7 muestra estos módulos en pestañas de una misma interfaz y la pantalla de bienvenida (splash).

Esta interfaz gestiona los experimentos mediante la información persistente de ficheros correspondientes a problemas, algoritmos y medidas de rendimiento. Con el objetivo de lograr un formato intuitivo y organizado de dichos ficheros se consideró el estándar XML.

[4] Standard Widget Toolkit
[5] www.eclipse.org

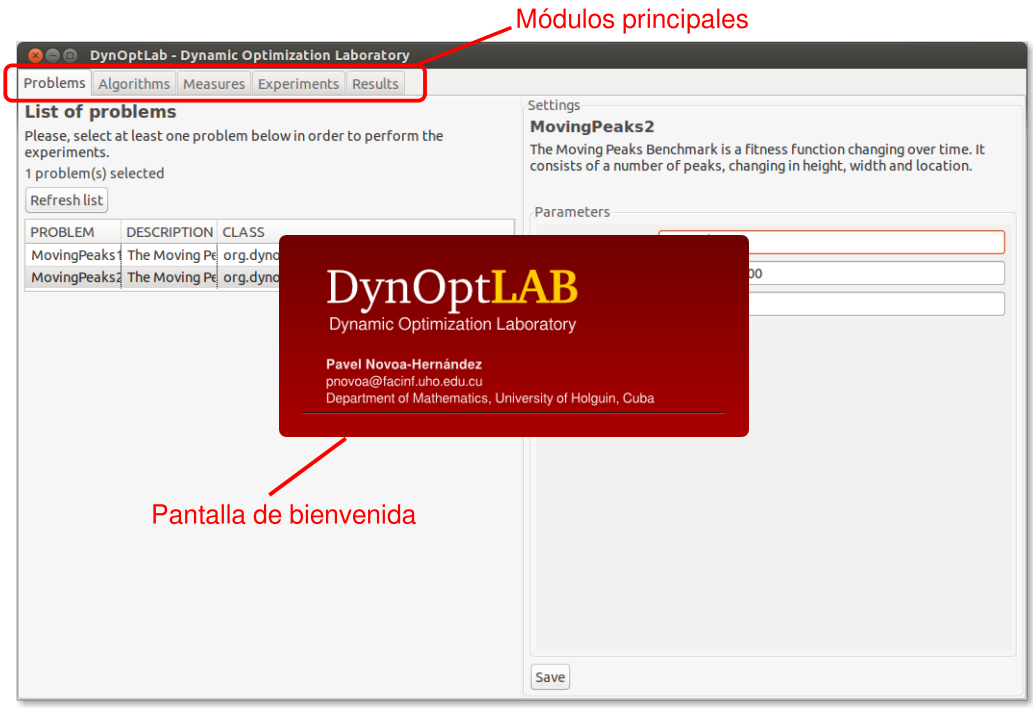

Figura 3.7 Interfaz de la aplicación *DynOptLAB*. La aplicación está organizada en los módulos *Problems*, *Algorithms*, *Experiment*, y *Results*

Estos ficheros se gestionaron con el uso del framework *Simple XML*[6], el cual permite realizar la persistencia de objetos en Java mediante anotaciones directas en el código fuente (en este caso, los atributos considerados como factores). En la Figura 3.8 se muestran ejemplos de la estructura de estos ficheros.

Como se aprecia, los tres elementos posee una estructura similar: nombre, descripción, clase de la que se deriva, nombre del fichero, y parámetros (los cuales son listas). En el caso particular de las medidas de rendimiento, la estructura del fichero también incluye el tiempo de medición, la función de agregación, y el tipo de reporte.

En lo que sigue, se explicarán las principales funcionalidades de la interfaz gráfica con más detalle.

Configuración de problemas, algoritmos y medidas

Como se mencionaba antes, uno de los requerimientos más importantes en *DynOptLAB* es la gestión de los factores y variables de respuesta de los experimentos. En este caso, los factores corresponden a los parámetros que definen a los problemas y algoritmos, mientras que las variables de respuesta a las medidas de rendimiento consideradas.

[6]www.simplexml.sourceforge.net

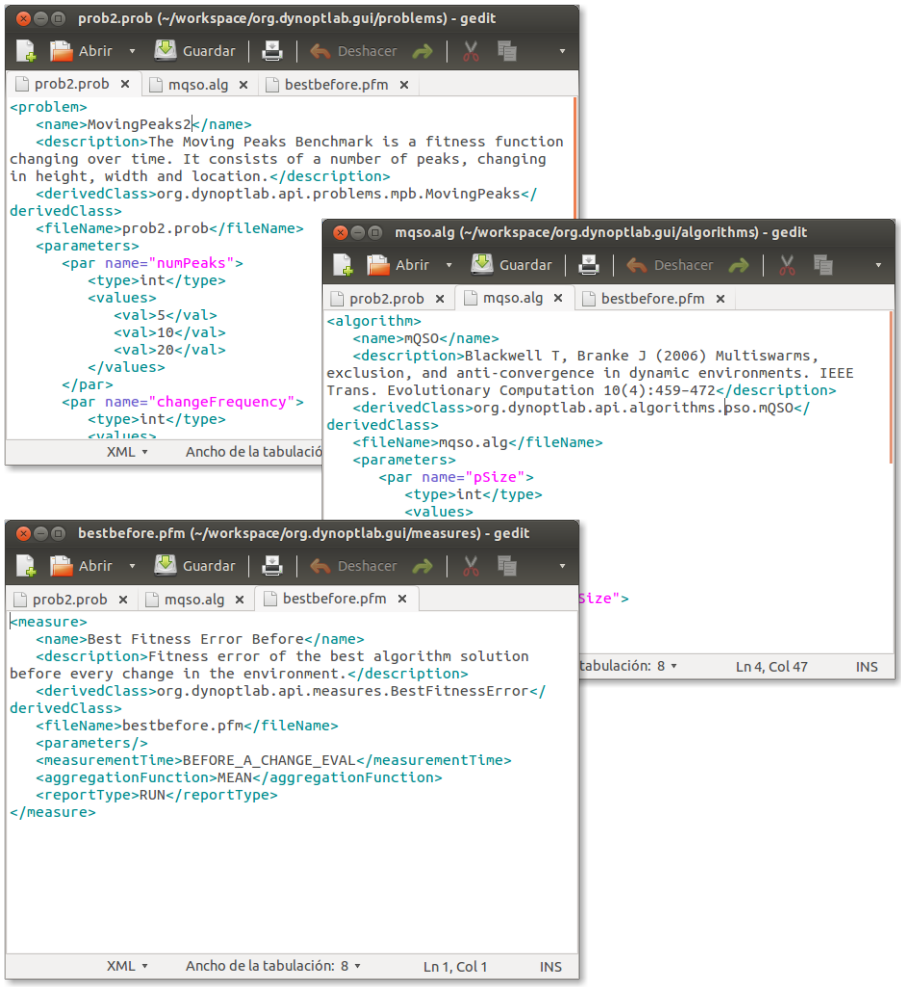

Figura 3.8 Fichero de configuración en *DynOptLAB*. Cada clase está asociada a un fichero similar, el cual permite establecer valores a los atributos de la clase sin modificar el código fuente.

En efecto, *DynOptLAB* provee la configuración de los problemas, algoritmos y medidas a utilizar en los experimentos. En la Figura 3.9 se muestra la pantalla relacionada con la configuración de los problemas. Esta interfaz consiste básicamente de dos zonas principales: en la parte izquierda se encuentra el listado de los problemas disponibles, y en la parte derecha los detalles y parámetros del problema seleccionado. La lista de problemas se llena a partir de los ficheros de configuración existentes en la carpeta *problems*, la cual se encuentra en la misma ruta que la aplicación. Cuando el usuario selecciona uno de los problemas que aparecen a su izquierda, en el panel de la derecha se muestran los parámetros definidos en el fichero de configuración. A estos parámetros se le pueden asignar valores separados por comas para indicar los factores que serán objeto de estudio en los experimentos. En este caso la figura muestra el ejemplo en el que al parámetro *vlength* se le asignan tres valores (factores): 1.0, 5.0, y 10.0. El usuario también podrá elegir otros parámetros y establecer los

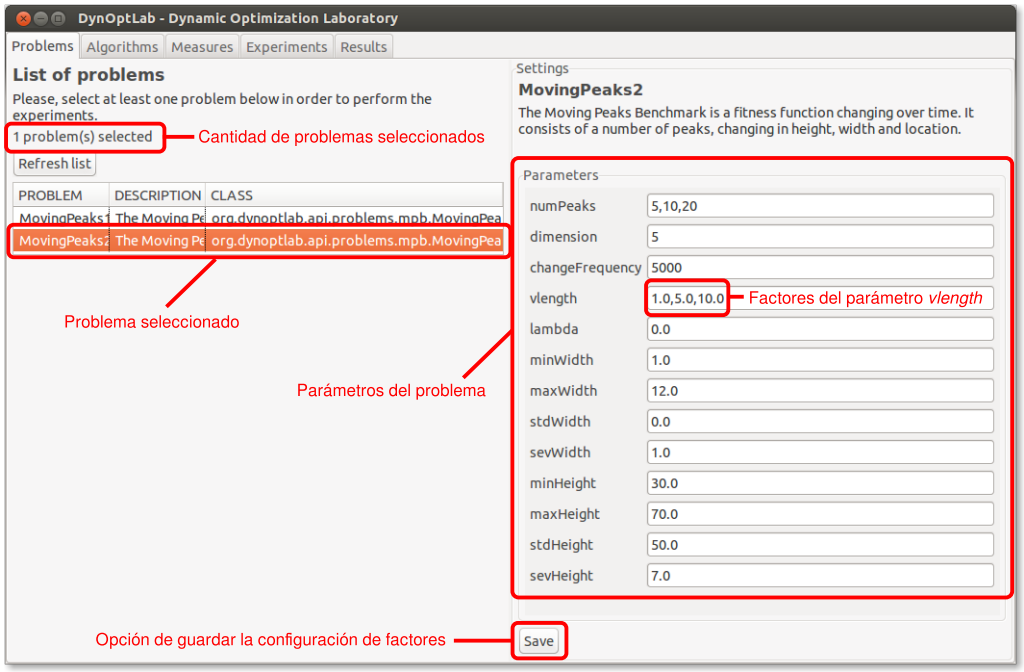

Figura 3.9 Pantalla correspondiente a la configuración de los problemas.

valores correspondientes. En cualquier caso, podrá guardar estas configuraciones mediante el botón *Save*.

La posibilidad de establecer varios factores por cada parámetro conlleva a la obtención de múltiples instancias de un mismo problema, esto es, una por cada combinación de factores. Por ejemplo, en el caso que se muestra en la Figura 3.9, existen dos parámetros, *vlength* y *numPeaks*, con tres valores cada uno. De manera que a partir de esta configuración se generarán nueve instancias distintas del problema *MovingPeaks2*.

De manera similar ocurre en las pestañas dedicadas a los algoritmos y medidas de rendimiento. Estas se verán más detalladamente en el caso de estudio al final de esta sección.

Gestión de los experimentos. Una vez seleccionados los problemas, algoritmos, y medidas, el usuario deberá seleccionar la pestaña *Experiments*. Como se apreciar en la Figura 3.10 esta pantalla brinda un resumen de los problemas, algoritmos y medidas seleccionados. En particular, el campo *Problem-algorithm pairs* indica la cantidad de pares de problema-algoritmos a partir de los factores definidos en las pantallas anteriores. En este caso, el ejemplo muestra 16 pares los cuales se convertiran en un experimento en específico.

En la zona de la derecha, se pueden establecer la semilla aleatoria inicial (la que será informada en tiempo de ejecución al algoritmo), el número de cambios de la simulación, número de ejecuciones por cada par problema-algoritmo, y finalmente el número de hi-

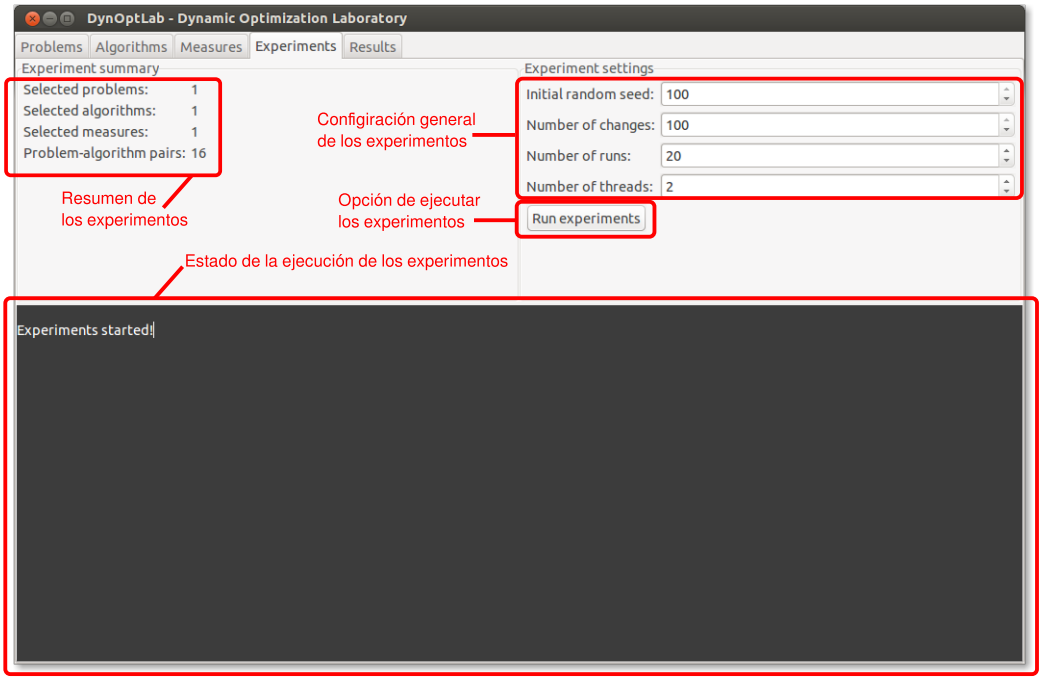

Figura 3.10 Pantalla dedicada a la gestión de los experimentos

los para ejecutar de forma paralela la cola de experimentos. Esta última información es suministrada a la clase *Simulator* descrita en la sección anterior.

La simulación comienza con la selección del botón *Run experiments* y acto seguido el panel inferior indica el estado de la misma. En particular, la simulación termina cuando en este panel se muestra la información *Experiments completed!*.

Cada terminación de un experimento (par problema-algoritmo) ocurre con la salva automática de información relacionada con los resultados de las medidas consideradas, y de la descripción del experimento en cuestión. Ambos tipos de información aparecerán en la carpeta dedicada al experimento. Las Figuras 3.11-a) y 3.11-b) muestran el formato de los ficheros de los detalles del experimento y de las medidas de rendimiento.

Visualización y análisis de los resultados. Las ejecuciones realizadas en el módulo *Experiments* es resumida en función de los resultados en la pestaña *Results*. Esta pestaña se muestra a partir de varias vistas en la Figura 3.12.

Como se aprecia en esta imagen, DynOptLAB ofrece la posibilidad de navegar por el conjunto de experimentos a través del panel izquierdo. Cuando el usuario selecciona uno de estos experimentos, en el panel derecho se le muestran los resultados de cada ejecución correspondientes a la medida seleccionada en la lista desplegable de la parte superior. Adicionalmente, en la parte inferior derecha, se resumen los principales estadígrafos

Figura 3.11 Formatos de los ficheros de salida de un experimento.

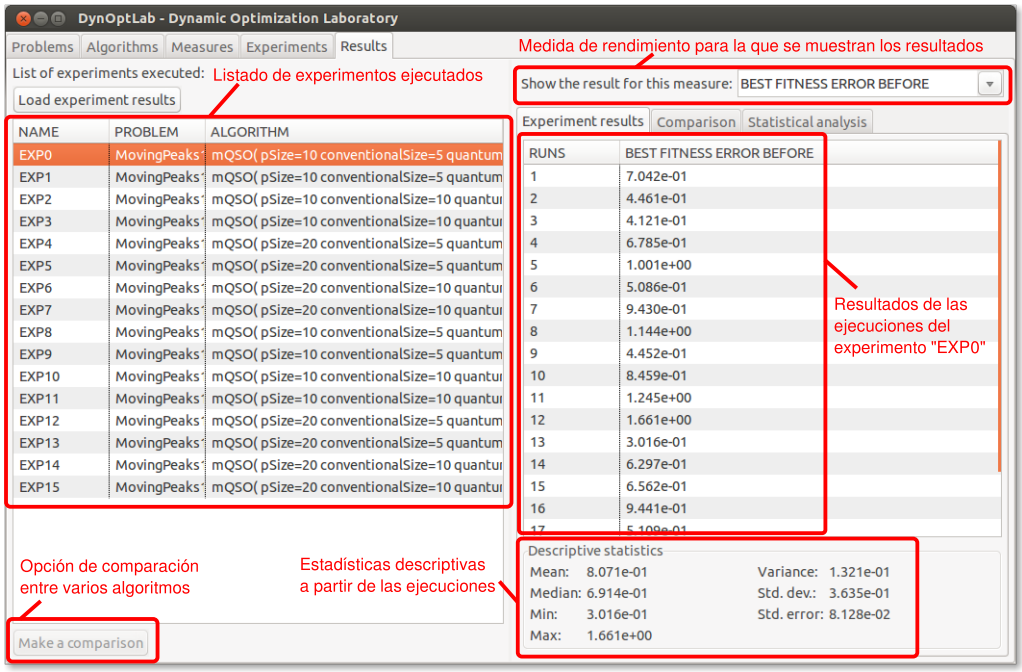

Figura 3.12 Pantalla *Results*, dedicada a la visualización y análisis de los resultados de los experimentos.

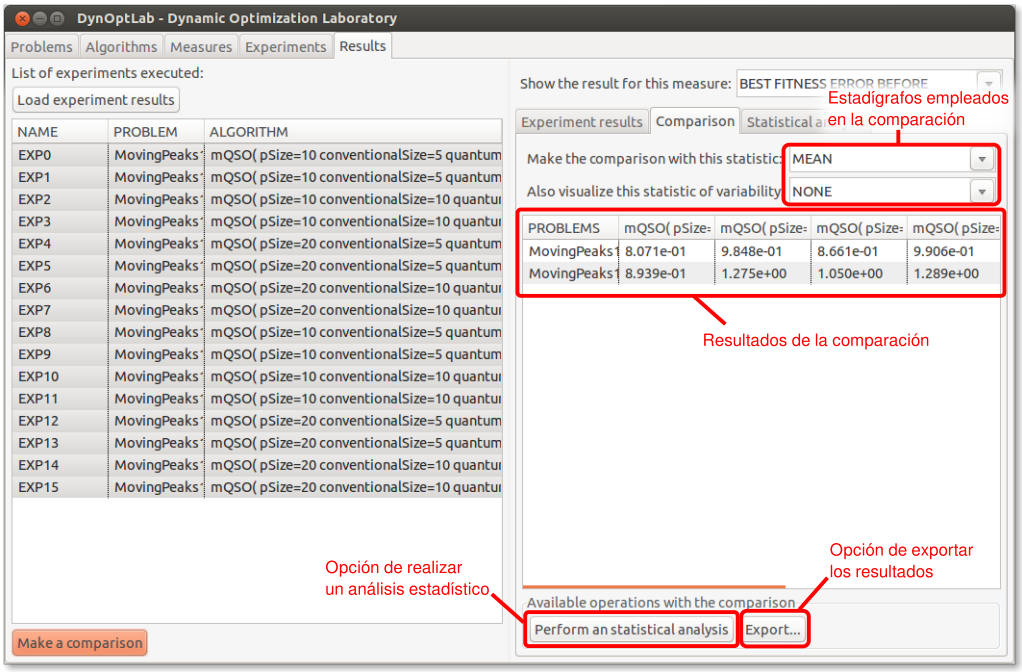

Figura 3.13 Ejemplo de la funcionalidad de comparación entre algoritmos en la pantalla *Results*

descriptivos de los datos de las ejecuciones. En este sentido se consideraron la media, la mediana, los valores máximo y mínimo, así como estadígrafos de dispersión más comunes: la varianza, la desviación y el error estándar.

No obstante esta información, habitualmente resulta de interés comparar los resultados de los algoritmos en determinados problemas. En este caso, DynOptLAB permite realizar una comparación de los resultados en función de los experimentos seleccionados en la parte izquierda de esta pantalla. Note que en la parte inferior de esta misma zona existe un botón (*Make a comparison*) que posibilita esta acción. Este botón solo se habilita si se seleccionan dos o más experimentos. La Figura 3.13 muestra un ejemplo en el que se ha desarrollado una comparación empleando todos los experimentos.

Aquí es posible ver en la pestaña *Comparison* los resultados de la comparación. En la parte superior se muestran los estadígrafos empleados en la comparación. Específicamente el primero (*MEAN*) se utiliza para agregar los resultados de las ejecuciones de los experimentos, mientras que el segundo (en este caso *NONE*) puede emplearse para visualizar adicionalmente un estadígrafo de dispersión. De manera que el usuario puede seleccionar ambos criterios para realizar la comparación. Los datos por cada problema y algoritmo se muestran en la tabla ubicada en la zona central de esta pestaña. Note que en la parte inferior el investigador tiene dos opciones en relación a esta comparación: la primera, realizar un análisis estadístico y la segunda exportar la tabla de la comparación. En este último caso, si

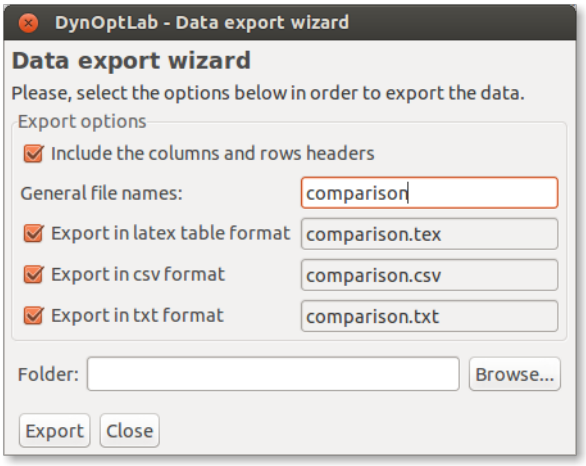

Figura 3.14 Pantalla para exportar los resultados de la comparación.

el usuario selecciona esta opción se le mostrará la ventana de la Figura 3.14. Es importante notar en esta pantalla que el usuario puede exportar los datos en tres formatos estándares: como una tabla en formato Latex, como un fichero en formato CSV, y finalmente como un fichero de texto simple.

En relación a la opción del análisis estadístico (botón *Perform an statistical analysis*), el usuario obtendrá los resultados que se muestran en la Figura 3.15. Aquí los datos de la tabla de comparación han servido de entrada para la realización de las pruebas estadísticas no paramétricas Friedman e Iman-Davenport.

En esta pestaña (*Statistical analysis*) se puede ver en la parte superior muestra numéricamente los rangos (posiciones) promedios de los algoritmos según la prueba de Friedman. Adicionalmente, estos rangos promedios son visualizados de manera gráfica a través de barras en la zona intermedia de la pestaña. Finalmente, los detalles de ambas pruebas se muestran en la zona inferior. En este sentido, hemos incluido los estadígrafos (*Statistic*), los grados de libertad (*Degree of freedom*), y los $p-$valores (*p-value*).

3.3.3 Un caso de estudio

A modo de comprobar las bondades que brinda *DynOptLAB* en esta Sección se describirá un caso de estudio. Se trata de incluir el conocido problema del Movimiento de Picos (MPB) (Branke, 1999b), y el algoritmo Multi-enjambre Quantum PSO (mQSO) (Blackwell y Branke, 2006), el cual es un referente del estado del arte que ha sido probado en este problema tipo. Como medida de rendimiento hemos considerado el $offlineError$ (error fuera de línea) definido en la Sección 2.4.4.

La forma en que se han incluido el generador MPB, el algoritmo mQSO, y la medida de

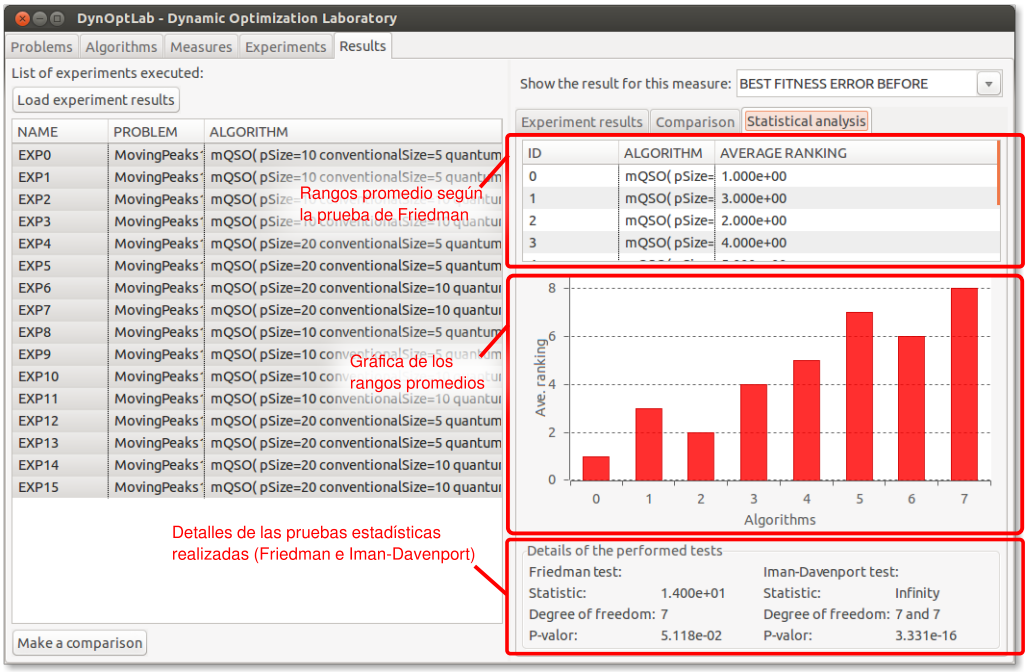

Figura 3.15 Pantalla para exportar los resultados de la comparación.

rendimiento *offlineError* en el framework de DynOptLAB se muestra en el diagrama de clases de la Figura 3.16. Como se aprecia, esto se lleva a cabo implementando los puntos calientes del framework correspondientes. En el caso particular de la medida, aunque en el diagrama se aprecia que se ha implementado la clase *BestFitnessError* es importante aclarar que el nombre de esta medida hace alusión a la medida puntual empleada en la definición del error fuera de línea. De manera que, siguiendo el marco de trabajo propuesto para las medidas de rendimiento en la Sección 3.1.2, el error fuera de línea deberá estar compuesta por la medida puntual *BestFitnessError*, el tiempo de medición *Evaluaciones* y la función de agregación *Media*. Precisamente, la Figura 3.17 muestra como puede lograrse esta configuración en DynOptLAB.

Dado que en la descripción de las pantallas anteriores se vio como configurar y ejecutar los experimentos, en lo que sigue nos dedicaremos a comentar los resultados del caso de estudio y analizar sus semejanza con los obtenidos en el trabajo original de Blackwell y Branke (2006). En este trabajo se analiza específicamente el escenario 2 del MPB, cuyos parámetros fueron descritos en la Sección 2.4.2. Los mejores resultados para este escenario corresponden a la configuración de mQSO mostrada en la Figura 3.18. En general los autores realizaron 50 ejecuciones y se obtuvo un error fuera de línea promedio igual a 1.75 y un error estándar de 0.06.

Para lograr una réplica fiel del experimento realizado en el trabajo original, se han

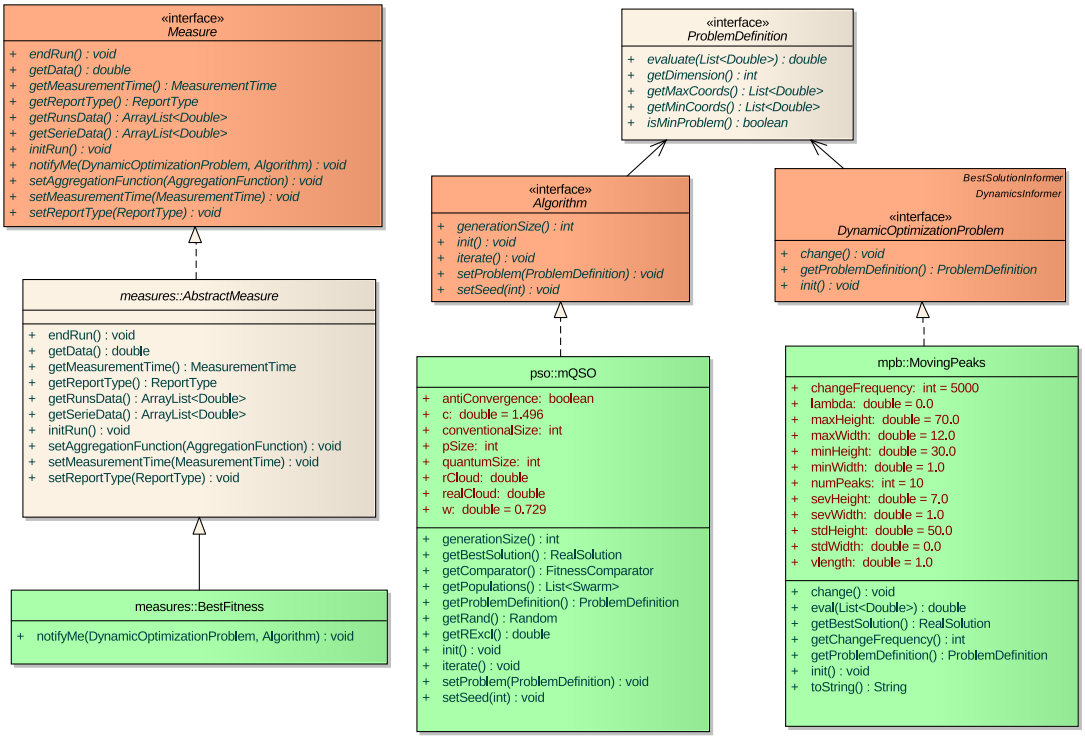

Figura 3.16 Inclusión del generador Movimiento de Picos, el algoritmo mQSO y la medida de rendimiento error fuera de línea (*BestFitnessError*) en el framework de DynOptLAB. Las clases con tonalidad verde son las implementadas.

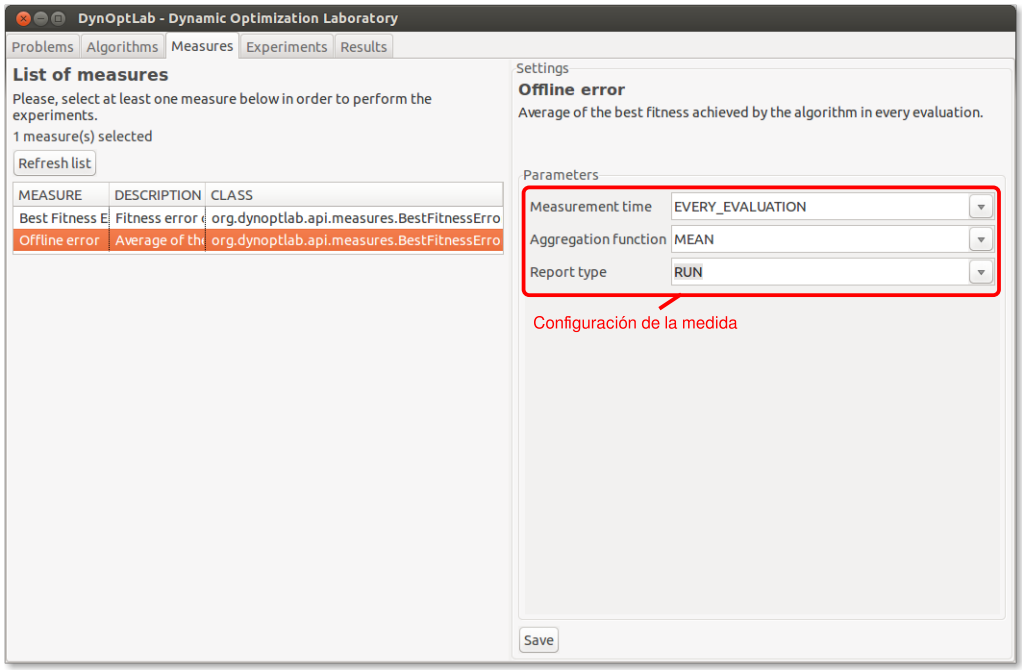

Figura 3.17 Pantalla correspondiente a la configuración de las medidas en DynOptLAB. La figura muestra la configuración de la medida error fuera de línea (*offline error*).

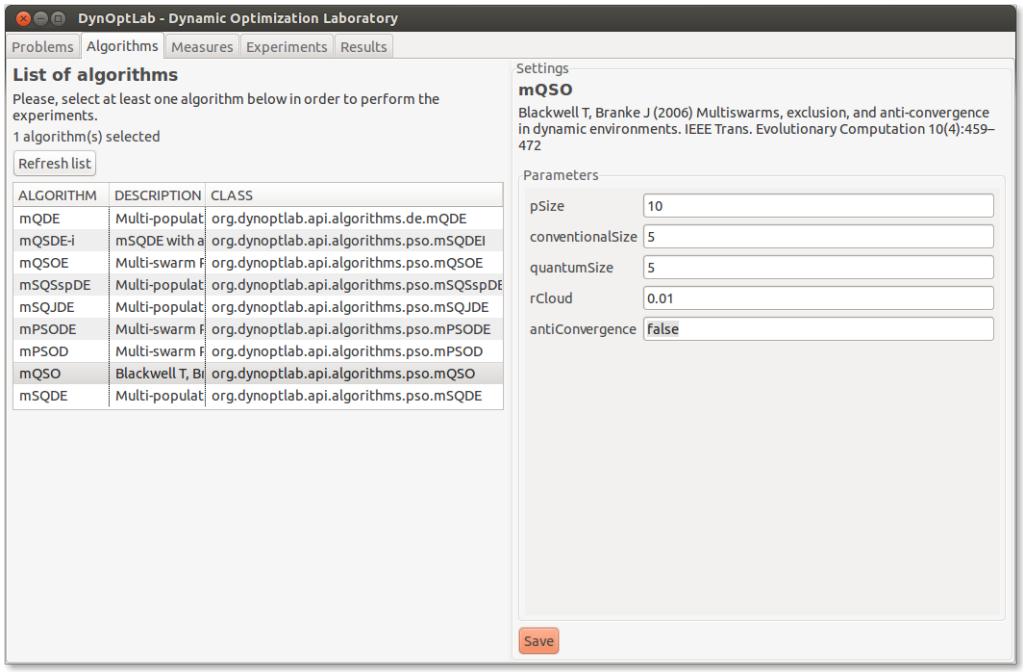

Figura 3.18 Pantalla correspondiente a la configuración de los algoritmos en DynOptLAB. La figura muestra la configuración del algoritmo mQSO (Blackwell y Branke, 2006).

considerado igualmente 50 ejecuciones y por supuesto, los mismos parámetros tanto para el algoritmo como para el problema. El resultado final de nuestro experimento puede ser observado en la Figura 3.19. Note que en nuestro caso hemos obtenido una media del error fuera de línea igual a 1.78, y un error estándar aproximadamente igual a 0.06. Sin dudas valores muy similares a los reportados en el trabajo original.

No obstante los resultados alcanzados hasta aquí, consideramos que este es un primer paso en la obtención de una mejor herramienta. Nuestros trabajos futuros versarán sobre la inclusión de nuevos problemas y algoritmos, con el objetivo de obtener una librería con los principales exponentes del estado del arte en esta área. Con esta librería se podrá facilitar la comparación entre propuestas, y el ahorro de tiempo en la codificación de nuevos algoritmos y problemas. También se incluirá el análisis post-hoc descrito en la Sección 3.2.

La futura disponibilidad de esta herramienta será lo más pronto posible, siendo de libre acceso desde el sitio *Intelligent Strategies in Uncertain and Dynamic Environments*[7].

3.4 Conclusiones

Al concluir este capítulo se pueden arribar a las siguientes conclusiones:

[7]http://www.dynamic-optimization.org/

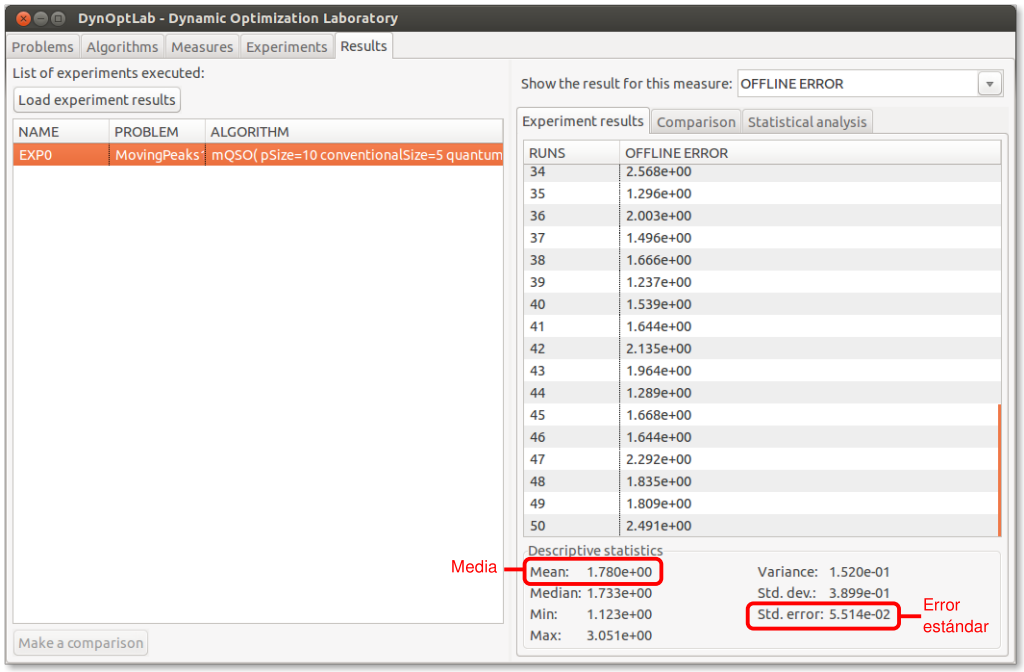

Figura 3.19 Pantalla *Results* de DynOptLAB donde se muestra la media y el error estándar correspondiente al algoritmo mQSO en el escenario 2 del generador MPB.

- El campo de las medidas de rendimiento en ambientes dinámicos es aún un tema en desarrollo, con algunas insuficiencias, entre ellas la ausencia de un marco de trabajo que organice convenientemente las propuestas actuales.
- El marco de trabajo propuesto en esta investigación permite organizar adecuadamente las medidas de rendimiento existentes a través de las categorías: información del algoritmo, información del problema, función de agregación, tiempo de medición. Lo que permite no solo identificar relaciones entre las propuestas existentes sino también una fácil implementación de las mismas.
- La aplicación del marco de trabajo propuesto a la revisión realizada sobre medidas de rendimiento en ambientes dinámicos permitió identificar los desarrollos actuales, principales tendencias y posibles medidas aún no propuestas o estudiadas.
- Una de las tendencias en este contexto es la utilización de medidas basadas en el fitness y que emplean como función de agregación la media. Asimismo se observa la utilización de una sola medida de rendimiento en la mayoría de los estudios, no obstante la existencia de otras medidas complementarias como la reacción y la estabilidad.
- En la actualidad aún existen análisis estadísticos basados en la comparación directa de los resultados o en la aplicación de pruebas paramétricas sin tener en cuenta supuestos de normalidad y el tamaño de la muestra. En este contexto, de acuerdo a (García et al,

2009), lo más apropiado resulta la aplicación de pruebas no paramétricas.

- Las metodologías existentes para el análisis estadísticos de los datos en el contexto de la optimización estacionaria pueden ser aplicadas sin mayores dificultades en ambientes dinámicos.
- Existen varias formas de visualizar los resultados de las pruebas no paramétricas, sin embargo, en ninguna se incluyen al mismo tiempo, las pruebas a nivel de grupo y los resultados de las pruebas post-hoc de comparaciones múltiples. En este sentido, la gráfica propuesta resulta eficaz en el cumplimiento de este requerimiento, pues muestra de manera intuitiva y compacta ambos tipos de resultados.
- En la actualidad se carece de herramientas para la experimentación en ambientes dinámicos. En ese sentido, la aplicación propuesta (*DynOptLAB*) constituye una paso hacia esa dirección.
- *DynOptLAB* permite la inclusión de nuevos algoritmos, problemas y medidas de rendimiento mediante la extensión del framework orientado a objetos que provee.
- *DynOptLAB* gestiona eficientemente, a partir de una interfaz gráfica, el diseño y ejecución de múltiples experimentos. Asimismo, en relación a los resultados de los experimentos, provee una visualización intuitiva de los mismos, realizar comparaciones y pruebas estadísticas no paramétricas.

4 Aumento de la eficiencia en enfoques PSO multi-enjambres

Los enfoques multipoblacionales se encuentran entre los más exitosos para lidiar con el denominado *problema de la convergencia* en ambientes dinámicos. Dentro de este enfoque, los algoritmos propuestos por (Blackwell y Branke, 2006), basados en el paradigma Optimización con Enjambre de Partículas (PSO), son importantes exponentes que han dado lugar al desarrollo posterior de extensiones más sofisticados. En este caso se encuentran las estrategias que se proponen en este Capítulo, orientadas a incrementar la eficiencia del enfoque de (Blackwell y Branke, 2006). El Capítulo comienza con una revisión del paradigma PSO en ambientes dinámicos, con el objetivo de describir cómo los progresos en esta área derivaron hacia enfoques más sofisticados (e.g. multipoblacionales). Más adelante, se describen las estrategias propuestas, las cuales se analizan a partir de varios experimentos computacionales.

4.1 PSO en ambientes dinámicos

Como se pudo ver en el Capítulo 2, la mayoría de las propuestas de técnicas en ambientes dinámicos se basan en la adaptación de métodos computacionales que fueron efectivos en problemas estacionarios (Dasgupta y Mcgregor, 1992; Angeline, 1997; Pelta et al, 2009; Moser y Chiong, 2010). Uno de estos paradigmas computacionales es la Optimización con enjambre de partículas (PSO) (Kennedy y Eberhart, 1995), la cual ha sido ampliamente aplicada en diversos escenarios (Banks et al, 2007, 2008).

De manera general, las dificultades más importante que presenta PSO en ambientes dinámicos son las conocidas como:

1. memoria desactualizada (*outdated memory*), y
2. pérdida de diversidad (*diversity loss*).

La memoria desactualizada, como su nombre lo indica, ocurre cuando las memorias individual ($\mathbf{p}_i$) y la global ($\mathbf{g}_{best}$) de las partículas quedan obsoletas debido a los cambios de la función objetivo.

En cambio, la pérdida de diversidad, es la incapacidad del enjambre en encontrar al nuevo óptimo global después de un cambio en el ambiente. Cuando el óptimo es desplazado fuera del alcance actual de las partículas, el enjambre converge a un óptimo local (ej. el $\mathbf{g}_{best}$ antes del cambio) y se moverá en una forma peculiar conocida como *colapso lineal* (Blackwell, 2003), donde las partículas describirán trayectorias en una misma dirección perpendicular al falso atractor.

No obstante estas dificultades específicas de PSO, se debe tener en cuenta que como cualquier otra meta-heurística la detección de los cambios en el ambiente es crucial. Académicamente, en la mayoría de las investigaciones se asume que el algoritmo es capaz de detectar los cambios (ej. reevaluando frecuentemente una o varias soluciones). En lo que sigue se hará una revisión de PSO en ambientes dinámicos.

4.1.1 Primeros trabajos

Quizás el primer trabajo que analizó a PSO en ambientes dinámicos fue el realizado por Carlisle y Dozier (2000). En esta investigación solo se trata el problema de la memoria desactualizada, el cual es solucionado con el reemplazamiento de la memoria personal de cada partícula por su posición actual (ej. $\mathbf{p}_i = \mathbf{x}_i$). Partiendo del PSO clásico, los autores proponen dos formas de desencadenar estas sustituciones: cada cierta cantidad fija de evaluaciones, o cuando el desplazamiento detectado por el algoritmo es superior a un cierto valor prefijado. El éxito en la predicción de tal desplazamiento se debe a que la función objetivo utilizada es precisamente la distancia euclidiana entre la solución candidata y el óptimo global: lo cual resulta algo idealista. Este trabajo fue mejorado ligeramente en (Carlisle y Dozier, 2001, 2002), donde se incluye como mecanismo para la detección de cambios, una partícula denominada *sentry* que es aleatoriamente seleccionada y evaluada antes de cada iteración. De manera que si ocurre un cambio se alerta a todo el enjambre y, a diferencia del reemplazamiento *a ciegas* anterior, ahora esta sustitución depende de si el fitness de $\mathbf{x}_i$ es mejor que el de $\mathbf{p}_i$.

Un estudio interesante que utiliza funciones con determinados niveles de ruido y dinamismo fue realizado por Parsopoulos y Vrahatis (2001). En este trabajo no se le hicieron modificaciones a PSO, debido a que los cambios en el ambiente no eran tan severos, posibilitando así que el enjambre no tuviera problemas en actualizar sus memorias y proseguir el proceso de convergencia.

Otro trabajo pionero fue el de Eberhart y Shi (2001), donde similarmente (y al parecer sin conocer el trabajo de Carlisle y Dozier) se intenta estudiar la capacidad de PSO en el seguimiento de los óptimos a través del tiempo. Esta vez se asumió conocida la frecuencia de cambio del ambiente y la memoria desactualizada fue resuelta mediante la sustitución de $\mathbf{p}_i$ por "el valor del error con respecto a la posición dinámicamente actualizada del óptimo". Aquí no queda claro si este error es conocido por el algoritmo o se trata de algún tipo de estimación. En cualquier caso, se concluye que para la función utilizada (parábola en tres dimensiones) PSO tiene un comportamiento superior a dos exponentes del estado del arte de aquella época: la programación evolutiva de Angeline (1997) y las estrategias evolutivas de Bäck (1998). Finalmente, aunque el aporte de este trabajo es modesto en materia de la adaptación de PSO a PDOs, indudablemente sentó las bases del que ha sido un tema de investigación fructífero y en boga en los últimos 20 años.

En 2002 aparece un trabajo interesante por Hu y Eberhart (2002), que basados en el trabajo anterior (Eberhart y Shi, 2001) proponen una nueva variante para la detección del cambio y ocho formas de generar diversidad en PSO. En el primer caso, se incluyó una segunda memoria global (junto a la actual del enjambre). Ambas memorias son chequeadas cada cierto tiempo (ej. un número prefijado de evaluaciones) con el objetivo de poder monitorear fuera del enjambre los cambios del óptimo en caso de que PSO quedara atrapado en un óptimo local. Por otro lado, el enfoque adoptado para lidiar con la pérdida de diversidad se basó en el reinicio aleatorio de una parte del enjambre. El generador empleado en los experimentos fue el DF1 (Morrison y De Jong, 1999). En esencia, los resultados revelaron que el método de detección de cambios era lento pero podía usarse, y que la mejor estrategia era aleatorizar el 10% de la población, o simplemente a la memoria global (para la función seleccionada). Una afirmación importante que aparece al final de este trabajo es cuando se plantea que "La detección de cambios y su respectivas respuestas son nuevos métodos de inteligencia que no pueden ser alcanzados individualmente por las partículas. Se trata de una inteligencia a nivel de población; en este nivel, se puede integrar a PSO más conocimiento e inteligencia humana para lidiar con sistemas complejos". Sin dudas ya se estaba sugiriendo la inclusión de esquemas o enfoques a nivel de enjambre para tratar la dinámica de los PDOs, idea que fue bien acogida por trabajos posteriores como se verá más adelante.

Un estudio comparativo entre las variantes anteriores: (Carlisle y Dozier, 2002), (Eberhart y Shi, 2001), (Hu y Eberhart, 2002), y un *fine-grained* PSO propuesto por Kennedy y Mendes (2002) para problemas estacionarios, fue desarrollado en Li y Dam (2003), donde se utilizó nuevamente el generador DF1 (Morrison y De Jong, 1999). La principal conclusión a la que arriba este estudio es que, de manera general, el PSO clásico con las modificaciones propuestas podía optimizar eficientemente problemas más complejos que la hasta entonces

típica función de la parábola. Específicamente cuando se tratan espacios de búsqueda de dos dimensiones, no así cuando existe una dimensionalidad mayor (ej. 10), donde el *fine-grained* PSO parece ser más eficiente. El éxito de este último algoritmo se debe a la utilización de la topología Von Neumann para definir el vecindario de las partículas.

Otros esquemas más recientes que adaptan a PSO en ambientes dinámicos y que recuerdan a los trabajos iniciales, superando a estos y al PSO original, son (Hu et al, 2007; Cui y Potok, 2007; Liu et al, 2008; Dong et al, 2008).

4.1.2 Hacia un inteligencia a nivel de población

Un paso importante hacia la *inteligencia a nivel de población* sugerida anteriormente, lo realizaron Blackwell y Bentley con la creación de un PSO *cargado* (CPSO) (Blackwell y Bentley, 2002a,b; Blackwell, 2003). Dos tipos de partículas son empleadas en un esquema que posee cierta analogía con la estructura de un átomo (véase la Figura 4.1). En el núcleo (centro del enjambre) se hallan las partículas neutrales, las cuales se mueven según el PSO clásico. En una zona exterior se mueven las partículas cargadas. Estas evitan la colisión inter-partículas a través de un factor de aceleración que varía sus dinámicas. CPSO fue probado en una función multimodal similar a la propuesta por el generador DF1. Los resultados de los experimentos demostraron que este algoritmo con la diversidad generada por sus partículas cargadas, más la convergencia de las neutrales, podía exhibir un rendimiento aceptable, sin las recetas determinísticas de diversidad propuestas en (Hu y Eberhart, 2002).

En el año 2004 ya empiezan a surgir trabajos que incluyen un enfoque multi-poblacional de PSO como estrategia indirecta de detección de cambios, pero sobre todo con la intención de favorecer la diversidad. En ese sentido, Parrott y Li (2004) desarrollaron un PSO basado en especiación (SPSO) , el cual ajusta dinámicamente el número y el tamaño de los enjambres a través de la generación de una lista ordenada de partículas. El criterio para el ordenamiento se hace basado en un ranking de acuerdo a la aptitud de cada partícula, las que se juntan en especies (sub-enjambres) si se encuentran espacialmente cercanas. De manera general, SPSO mostró un buen rendimiento en el seguimiento de óptimos en el generador DF1.

Por otro lado, el enfoque atómico de CPSO es enriquecido por Blackwell y Branke (2004, 2006) con su aplicación a varios enjambres, surgiendo así el multi-enjambre CPSO (mCPSO). Un algoritmo similar, el multi-enjambre quantum PSO (mQSO), fue también propuesto en estos trabajos, esta vez, en vez de partículas cargadas se usaron partículas de tipo quantum. Estas partículas se mueven aleatoriamente alrededor del núcleo formado por las neutrales en una hiperesfera con radio r_{cloud}. Tanto CPSO como mQSO fueron probados

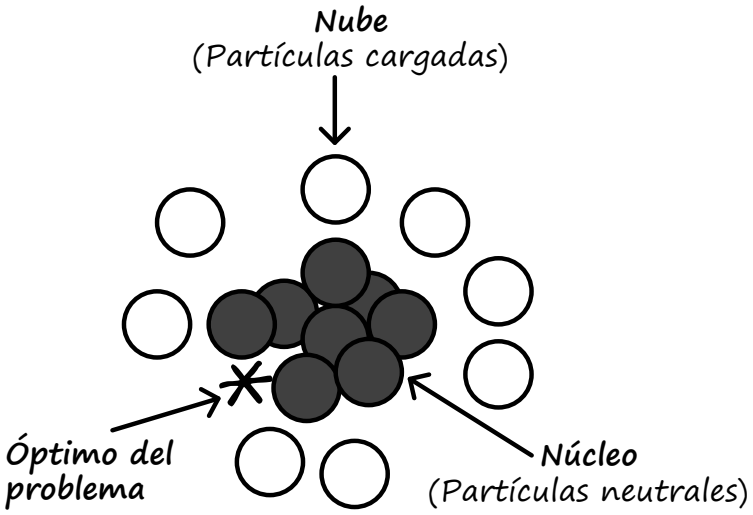

Figura 4.1 PSO con enfoque atómico. Una nube de partículas cargadas envuelve a un núcleo de partículas neutrales. Las primeras mantienen la diversidad en tiempo de ejecución, las segundas se dedican a explotar la mejor solución encontrada.

en el MPB propuesto por Branke, el cual al ser multimodal requirió la presencia de nuevas técnicas para definir la relación entre los enjambres. Dos estrategias fueron implementadas, el principio de exclusión que evita la exploración de un mismo pico por más de un enjambre, y un operador de anti-convergencia que mantiene al peor de los enjambres siempre vigilando el espacio de búsqueda al ser reiniciado aleatoriamente en cada generación. Esta última estrategia es más conveniente cuando el número de picos es superior a la cantidad de enjambres. El enfoque multi-enjambre de Blackwell y Branke será explicado con más detalle en la sección siguiente.

Otro trabajo multi-poblacional de ese mismo año fue el de Janson y Middendorf (2004), en el cual se emplea un topología jerárquica en forma de árbol para las partículas (H-PSO) que mostró ser más eficiente en problemas estáticos que el PSO original (Janson y Middendorf, 2003). Junto a H-PSO también fue propuesto la variante PH-PSO, la cual particiona a toda la población en subenjambres como respuesta a cada cambio detectado en el ambiente, y con el tiempo las vuelve a fusionar inteligentemente. Tanto H-PSO como PH-PSO fueron probados en diferentes PDOs, entre ellos el MPB. Los resultados confirmaron nuevamente la superioridad de H-PSO ante PSO pero esta vez en ambientes dinámicos unimodales, mientras que PH-PSO resultó más adecuado en los multimodales siempre y cuando la severidad de cambio no fuera tan alta.

En la literatura también es común observar el enriquecimiento de PSO con estrategias

de probada efectividad en el área de la Computación Evolutiva. Un ejemplo de ello lo constituye el trabajo de Ahmed y Kamangar (2005) en el que se usa un método de selección basado en ránking para reemplazar la mitad de la partículas con menor calidad por las mejores, cuando un cambio ocurre en el ambiente. Otro aspecto interesante es la adaptabilidad con la que dotan a PSO, al incluirle coeficientes de aceleración variables en el tiempo. El rendimiento de esta extensión de PSO, denominada RS-PSO fue probada con buenos resultados en la función parabólica DF1 para varios severidades de cambio.

Con la intención de mezclar la exploración y explotación a partir de los modelos *lBest* y *gBest*, en (Parsopoulos y Vrahatis, 2005) se analiza un PSO Unificado (UPSO) en ambientes dinámicos, originalmente propuesto para problemas estáticos en (Parsopoulos y Vrahatis, 2004). UPSO demostró efectividad en la suite de problemas de Morrison y De Jong (1999) y le permitió concluir a sus autores que también funcionaba en ambientes dinámicos.

De manera similar a UPSO, Schoeman y Engelbrecht (2006) probaron su PSO basado en vectores (*vector-based* PSO) para la identificación de nichos en varios escenarios multimodales dinámicos. Esta técnica usa información de resultados previos para encontrar óptimos después de cada cambio en el ambiente. Los resultados fueron un tanto variables, y en particular la técnica demostró ser contexto-dependiente.

Dos algoritmos de buenos resultados en problemas estáticos fueron analizados por Pan et al (2006), el PSO con núcleo de enjambres evolutivos (SCEPSO) y un híbrido de PSO con Simulated Annealing (PSOwSA). En SCEPSO toda la población es dividida en tres enjambres: el núcleo, el cercano, y el lejano, donde el primero utiliza ideas de la programación evolutiva. Los otros dos sirven de estrategias de búsqueda local y global respectivamente. En los experimentos se comparan estas dos extensiones con PSO, y el resultado es que en los modelos dinámicos utilizados como problemas, SCEPSO es superior al resto.

La idea de seguir los óptimos en el tiempo, en lugar de reoptimizar el problema desde cero, puede ser asumida de diferentes maneras (como se ha visto anteriormente). Precisamente, un enfoque interesante es el de Wang et al (2007) en el que se emplea un PSO con memoria. Este enfoque está basado en el modelo de tres islas para un algoritmo genético expuesto por Branke (1999b), donde se definen tres tipos de población: de explotación, exploración, y memoria de las mejores soluciones encontradas. Lo nuevo aquí (a parte de la hibridación con PSO) es la descripción de dos nuevos mecanismos para el acceso a memoria, y la estimación de si el enjambre ha encontrado un óptimo. Los resultados de los experimentos sobre el escenario 2 del MPB demostraron que la inclusión de la memoria en PSO supera en rendimiento al PSO clásico con estrategia de reinicio, y que el esquema de acceso a las soluciones de la memoria más adecuado es el de basado en *inmigrantes*. Básicamente, este esquema reemplaza periódicamente a las peores soluciones

de la población dedicadas a la *explotación*.

Otro paso hacia la hibridación lo proponen Li y Yang (2008a) con un PSO multi-enjambre rápido (FMPSO), en el que un enjambre padre (buscador global) va creando enjambres hijos (buscadores locales) a medida que las soluciones encontradas por el primero son buenas. Mientras el padre se mueve según el algoritmo de Programación Evolutiva Rápida (Fast Evolution Programming) (Yao y Liu, 1996), los hijos usan FPSO (PSO con mutación Cauchy y una estrategia de selección evolutiva) (Li et al, 2007). Esta mezcla resultó eficaz, en su comparación con otro algoritmo similar, se puede apreciar una amplia superioridad en el escenario 2 del MPB.

4.1.3 Enfoque PSO multi-enjambre

Debido a la importancia que reviste para el presente trabajo, en esta sección nos centraremos en describir el enfoque propuesto por Blackwell y Branke (2006), específicamente la variante mQSO.

Básicamente, el origen de este enfoque esta inspirado en los exploradores auto-organizados propuestos por Branke (2002), y en la estructura atómica del PSO cargado de Blackwell y Bentley (2002a). A nivel de enjambre se mantienen dos tipos de individuos: partículas neutrales (dedicadas a la explotación), y partículas quantum para la exploración en cierta vecindad del óptimo actual. Además, debido a ese comportamiento, las de tipo quantum resultan muy adecuadas para la generación de diversidad después del cambio, ya que si la magnitud del cambio es similar al radio de exploración de estas, entonces existen más probabilidades de encontrar al nuevo óptimo. Los pasos de mQSO son mostrado en el algoritmo 4.1.

Inicialmente se distribuyen aleatoriamente todas las partículas por el espacio de búsqueda, luego en un ciclo principal (que define las generaciones del algoritmo) se realizan tres pruebas a nivel de enjambres: Exclusión, Anti-convergencia, y Cambios. La primera tiene como objetivo prohibir que dos enjambres se disputen un mismo óptimo, esto se logra midiendo la distancia euclidiana entre los $\mathbf{g}_s$ de cada par de enjambres, de manera que si esta separación es menor que un radio r_{excl} entonces el peor de los enjambres es marcado para ser reiniciado aleatoriamente. Por su parte, la prueba de anti-convergencia verifica si todos los enjambres han convergido, y de ser así marca para reiniciar al peor de ellos. La convergencia de los enjambres es medida en función de la diversidad de las partículas, definida como la máxima diferencia en una componente del vector posición para dos partículas cualesquiera. La prueba de *cambio* evalúa el $\mathbf{g}_s$ de cada enjambre para encontrar inconsistencias en el ambiente, de manera que si $f^{(t+1)}(\mathbf{g}_s) \neq f^{(t)}(\mathbf{g}_s)$ significa que ha ocurrido un cambio en el

```
1  para cada subpoblación P_k hacer
2  |   Inicializar aleatoriamente a P_k;
3  fin
4  mientras no condición de parada hacer
5  |   Aplicar principio de exclusión;
6  |   si ningún cambio es detectado entonces
7  |   |   para cada subpopulation P_k hacer
8  |   |   |   Iterar individuos convencionales median el paradigma PSO;
9  |   |   |   Actualizar la mejor solución de P_k;
10 |   |   |   Iterar individuos quantum alrededor de la mejor solución de la P_k;
11 |   |   |   Actualizar la mejor solución de P_k;
12 |   |   fin
13 |   fin
14 |   si no
15 |   |   para cada subpopulation P_k hacer
16 |   |   |   Reevaluar las memorias de P_k;
17 |   |   fin
18 |   fin
19 fin
```

Algoritmo 4.1: Pasos generales del enfoque PSO multi-enjambre (Blackwell y Branke, 2006).

problema, entonces, como respuesta se vuelven a evaluar todas las memorias personales de las partículas. Finalmente, el algoritmo continúa con la actualización de la posición de cada partícula en dependencia de su tipo. Los restantes pasos son los mismos del PSO clásico. Para ilustrar gráficamente este enfoque multi-enjambre, la Figura 4.2 representa dicho enfoque optimizando una función multimodal de dos dimensiones.

Según los experimentos realizados por sus autores, la mejor configuración de los parámetros en mQSO para el escenario 2 del MPB, son los siguientes: 10 enjambres con 10 partículas cada uno, con igual cantidad de neutrales que de quantum (5 y 5), $r_{excl} = 31.5$ (estimado en función de las dimensiones del espacio de búsqueda y la cantidad de enjambres), $r_{cloud} = 1.0$ (de igual magnitud que la severidad de los cambios), $r_{conv} = 40.0$. La superioridad de esta configuración puede ser comprobada en el trabajo original (Blackwell y Branke, 2006), lo cual se muestra en la Figura 4.3.

Aunque la idea de utilizar tantos parámetros no suena sugerente, el principal aporte de este trabajo fue demostrar que este enfoque funciona en ambientes dinámicos, sirviendo así como base (e inspiración) para trabajos futuros, orientados sobre todo a la adaptación de sus principales parámetros durante la ejecución, como se verá más adelante en los capítulos siguientes.

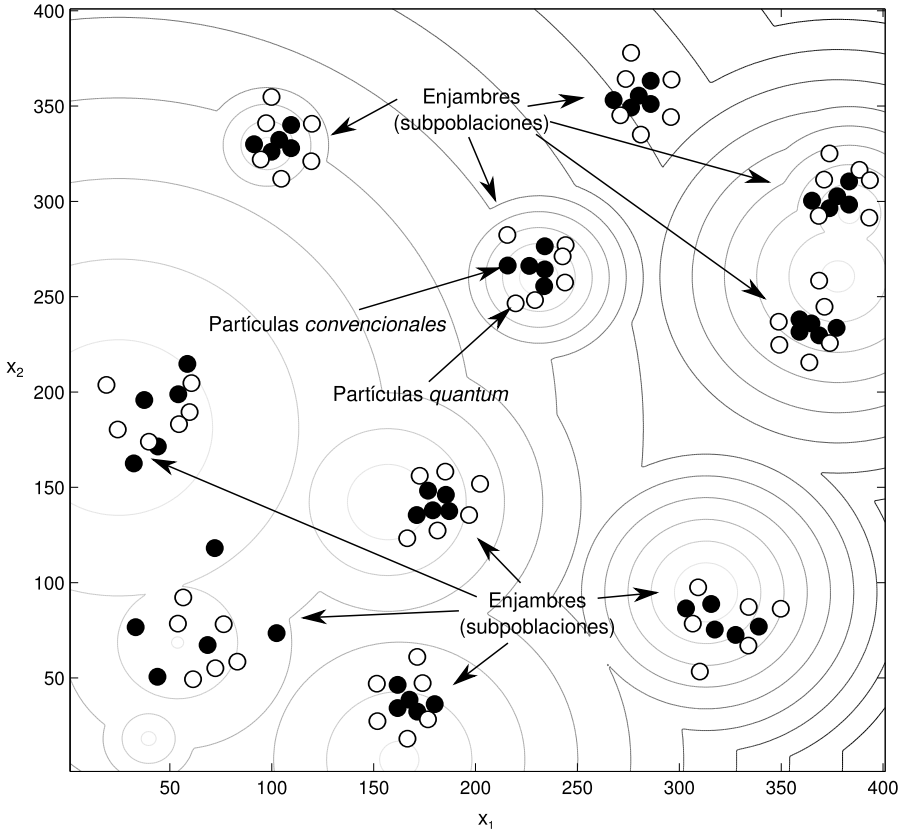

Figura 4.2 Enfoque Quantum PSO multi-enjambre (10 enjambres) en una función multimodal (10 picos) de dos dimensiones (Blackwell y Branke, 2006). Las curvas de nivel corresponden a la función objetivo.

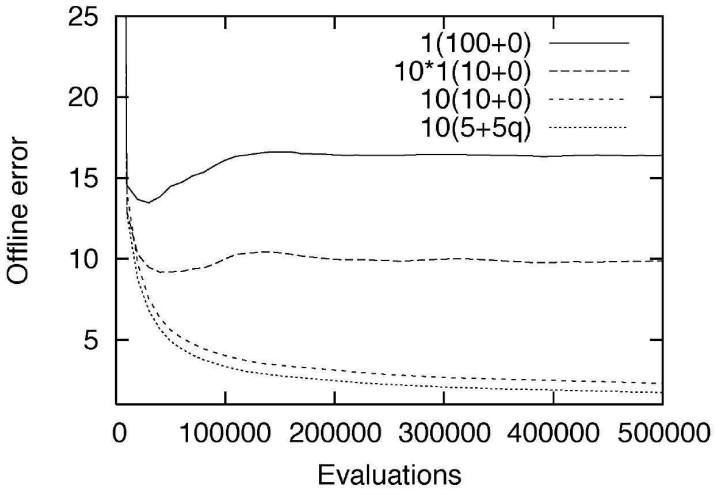

Figura 4.3 Distintas variantes del enfoque mQSO. Las series muestran el *error fuera de línea* a través del tiempo de cada variante del algoritmo (Blackwell y Branke, 2006).

4.2 Estrategias propuestas

No obstante el éxito logrado en el pasado por el enfoque PSO multi-enjambre, en nuestra opinión el mismo puede ser mejorado aún como se verá en lo que sigue. Los resultados que se describirán a continuación se encuentran en los trabajos (Novoa-Hernández et al, 2010, 2011). En esencia, se proponen dos estrategias para mejorar al enfoque multi-enjambre. La primera está relacionada con la generación de diversidad después de los cambios (a diferencia de como lo hacen los algoritmos mQSO y mCPSO, esto es, en tiempo de ejecución). La segunda es una regla de control que adapta el número de enjambres en tiempo de ejecución. El estudio brindado en esta sección provee además guías para un ajuste adecuado de los parámetros involucrados.

4.2.1 Estrategia para la generación de diversidad después de los cambios

Como se advirtió anteriormente, el problema más difícil que presenta PSO cuando es adaptado en ambientes dinámicos es la pérdida de diversidad. En este sentido, la literatura muestra diversas propuestas para lidiar con este aspecto (véase por ejemplo la sección anterior). Quizás la más simple es la reinicialización aleatoria del enjambre después de detectado un cambio en el ambiente (Hu y Eberhart, 2002). Sin embargo, en ocasiones resulta poco práctico aplicar tanta diversidad si el problema en cuestión posee ciertas características como: cambios no tan severos y ausencia de zonas del espacio de búsqueda donde el valor de la función objetivo es constante. En particular, esto último permite que las partículas estén, en la mayoría de los casos, en el seno de atracción de los óptimos del problema. Si a esto se le añade el uso de múltiples poblaciones del enfoque mPSO (que permite una exploración simultánea del espacio de búsqueda), entonces parece adecuada la generación de diversidad de manera local.

La manera que las variantes mCPSO y mQSO generan diversidad es a través del uso de partículas con movimientos diferentes. En el primero, la diversidad es mantenida por la repulsión Coulomb entre las partículas cargadas, mientras que en el segundo las partículas quantum son aleatoriamente generadas en una bola con centro en $\mathbf{g}_{best}$. Obviamente, lo que ambos enfoques tienen en común es que generan diversidad durante la ejecución. (ej. en cada iteración del algoritmo). Sin embargo, existe evidencia de que esta diversidad es mucho más efectiva como respuesta a los cambios en el ambiente (García Del Amo et al, 2010). Siguiendo esta idea, se propone la siguiente regla : una vez detectado un cambio en el problema, cada enjambre es dividido en dos grupos en relación a la calidad de las partículas. Una parte permanecerá fija mientras que el resto será diversificada alrededor

de la mejor partícula del enjambre $\mathbf{g}_{best}$. Esta perturbación alrededor de $\mathbf{g}_{best}$ es llevada a cabo siguiendo una distribución uniforme (UD) en una hiperesfera, como se muestra en (Clerc, 2006). Esta hiperesfera estará centrada en $\mathbf{g}_{best}$ y radio r_{exp}. Resulta claro que esta estrategia es similar al generación de individuos quantum en el algoritmo mQSO, sin embargo nuestra estrategia es ejecutada después de cada cambio y es aplicada a las *peores* partículas del enjambre.

El Algoritmo 4.2 resume los pasos de la estrategia de diversidad propuesta. Note que primero una lista ordenada es creada con las peores partículas en las primeras posiciones. Entonces, todas las partículas consideradas como *peores* en esta lista reemplazan sus vectores de posición por un punto generado por una distribución uniforme con centro en $\mathbf{g}_{best}$ y radio r_{exp}. Los pasos que quedan están dedicados a evaluar las nuevas posiciones y actualizar las memorias de la partícula y del enjambre. En lo que sigue se llamará mPSOD al enfoque mPSO con esta estrategia de diversidad.

```
   // Ordenar las partículas del enjambre s
1  Asignar listaOrdenada ← ordenarParticulas(s);
   // Seleccionar las peores partículas
2  Asignar peoresParticulas ← seleccionarPeores(listaOrdenada) ;
   // Generar partículas alrededor de la mejor solución de s
3  para cada partícula i en peoresParticulas hacer
4  |   Asignar x_i ← aleaSphere(g_best, r_exp);
5  |   Evaluar la nueva posición x_i ;
6  |   Actualizar las memorias personal p_i y global g_best;
7  fin
```

Algoritmo 4.2: Estrategia de diversidad propuesta en el método mPSOD.

4.2.2 Regla de control difusa para la evolución de enjambres

La regla de control que se propone en este apartado, se basa en el hecho de que en problemas multimodales no todos los enjambres se encuentran en regiones prometedoras del espacio de búsqueda. De manera que con suerte solo algunos se encontrarán optimizando en el óptimo global o soluciones cercanas a este, mientras que el resto estarán probablemente en soluciones de baja calidad malgastando recursos computacionales significativos (ej. evaluaciones de la función objetivo). En términos prácticos, estas evaluaciones de la función objetivo, traducida en tiempo, pudieran emplearse en la explotación del óptimo global en las iteraciones previas a un nuevo cambio.

Una idea sugerente sería detener esos *malos* enjambres en algún punto de la ejecución del algoritmo. Sin embargo, recuérdese que el objetivo de emplear múltiples enjambres es la exploración eficiente del espacio de búsqueda. Por lo que la pregunta que emerge de este

análisis es ¿cómo detener enjambres de baja calidad sin afectar la exploración de todo el algoritmo?

Para tratar este problema con conflictos de objetivos, nos centraremos en dos aspectos importantes del enjambre. Primero, la calidad del enjambre (expresada por la mejor solución encontrada por sus partículas, g_{best}), y en segundo lugar el grado de diversidad existente entre las partículas. En ese sentido, se considerará que cada enjambre puede ser clasificado de acuerdo en una de las posibles combinaciones de estas dos características. Esta clasificación puede ser establecida usando dos etiquetas para cada aspecto: *bajo* y *alto* para la diversidad, mientras que *malo* y *bueno* para la calidad. Note que aquí un bajo nivel de diversidad es una condición necesaria para la convergencia del enjambre.

Resulta claro que nuestra regla de control estaría dedicada a la detención de enjambres con diversidad *baja* y una calidad *mala*, ya que son estos enjambres los que con toda seguridad consumiría evaluaciones extras de la función objetivo. Sin embargo, queda por definir que se entiende por una buena o mala calidad, y que medida resulta más adecuada para representar la diversidad del enjambre.

En el primer caso, nos basaremos en un desarrollo previo que fue aplicado con éxito en el contexto de las estrategias cooperativas multi-hilo para la optimización de problemas estacionarios (Pelta et al, 2006). Básicamente, la idea es emplear definir una función de pertenencia difusa para evaluar la calidad de cada enjambre. Dicha función tiene la siguiente expresión:

$$\mu_{malo}(f_s) = \begin{cases} 0.0 & \text{si } f_s > b; \\ \frac{(b-f_s)}{(b-a)} & \text{si } a \leq f_s \leq b; \\ 1.0 & \text{si } f_s < a. \end{cases} \tag{4.1}$$

A diferencia del trabajo de Pelta et al (2006), aquí a y b son dos parámetros que varían con el tiempo. Formalmente, siendo m el número de enjambres y $\mathbf{g}_i$ la mejor solución encontrada por el enjambre i, entonces a y b se actualizan en cada iteración del algoritmo a través de las expresiones siguientes:

$$a = \frac{1}{m}\sum_{i=1}^{m} f^{(t)}(\mathbf{g}_i) \tag{4.2}$$

$$b = \max_{i=1\ldots m} f^{(t)}(\mathbf{g}_i) \tag{4.3}$$

Concretamente, la calidad de un enjambre es una magnitud relativa que depende de la calidad promedio de todos los enjambres y de la calidad del mejor de este conjunto. De manera que predefiniendo un umbral γ_{cut} (que actúa como un α corte) se puede clasificar

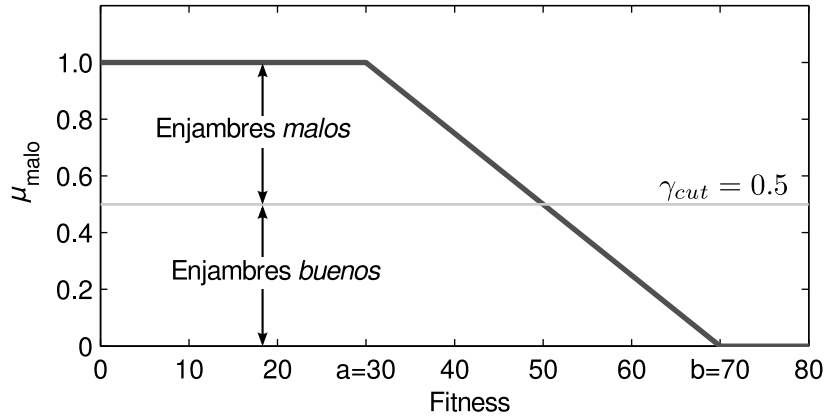

Figura 4.4 Función de pertenencia μ_{malo}

un enjambre s como de *baja calidad* si la siguiente función booleana devuelve *verdadero*:

$$bajaCalidad(s) = \begin{cases} verdadero & \text{si } \mu_{malo}(f_s) \geq \gamma_{cut}; \\ falso & \text{en caso contrario.} \end{cases} \tag{4.4}$$

A modo de ilustrar gráficamente como se define la función de pertenencia difusa μ_{malo} véase la Figura 4.4. En este ejemplo, a =30.0, b =70.0, y γ_{cut}=0.5. Note que el parámetro γ_{cut} crea dos grupos de acuerdo al grado de pertenencia de cada enjambre: enjambres *malos* y enjambres *buenos*.

Por otro lado, con el objetivo de medir el grado de diversidad de los enjambres se ha seleccionado la medida propuesta por Blackwell (2005), y que se define como la máxima separación entre dos componentes de todas las partículas del enjambre en cuestión. Esta medida fue usada previamente por el autor para activar o no el operador de anti-convergencia en el enfoque mPSO (Blackwell y Branke, 2006). La fórmula sugerida por Blackwell es la siguiente:

$$\delta_s = \max \left| x_i^k - x_j^k \right| \quad i, j = 1, ..., \mu \quad k = 1, ..., D \tag{4.5}$$

donde μ es el número de partículas de cada enjambre, y D es el número de dimensiones del espacio de búsqueda. Como en el esquema original del enfoque mPSO se asumirá que un enjambre ha convergido si la diversidad calculada por 4.5 es menor que un valor prefijado. De manera que se puede definir la siguiente función booleana:

$$haConvergido(s) = \begin{cases} verdadero & \text{si } \delta_s \leq r_{conv}; \\ falso & \text{por el contrario.} \end{cases} \tag{4.6}$$

La regla de control propuesta quedaría de la siguiente forma:

Si no hay cambio reciente **y** el enjambre *s* ha convergido **y** *s* es de baja calidad **entonces** parar enjambre *s*.

Note que adicionalmente, en la regla de control se ha incluido también la condición **no** *hay cambio reciente* que significa la ausencia de un cambio en el ambiente en la iteración actual. La presencia de esta condición es algo obvia si se tiene en cuenta que no tiene mucho sentido detener un enjambre con baja diversidad y baja calidad después de un cambio en problema, ya que puede ocurrir que este enjambre esté actualmente muy cerca de una buena solución producto al nuevo cambio.

4.2.3 Algoritmos propuestos

El Algoritmo 4.3 muestra como se pueden incluir las mejoras propuestas en las secciones anteriores en el enfoque *mPSO*.

```
1  Inicializar aleatoriamente todas las partículas en el espacio de búsqueda;
2  mientras condición de parada hacer
3      Aplicar principio de exclusión;
4      Detectar cambios en el ambiente;
5      para cada enjambre s hacer
           // Estrategia de diversidad después de los cambios
6          si se ha detectado un cambio entonces
7              Aplicar la estrategia de diversidad (Alg. 4.2) ;
8          fin
           // Regla de control de enjambres
9          si no hay cambio reciente y haConvergido(s) y bajaCalidad(s) entonces
10             Parar enjambre s ;
11             continuar;
12         fin
13         si no
14             Iterar partículas de acuerdo a su tipo (ej.convencionales y quantum);
15             Evaluar cada partícula;
16             Actualizar memorias p_i y g_best ;
17         fin
18     fin
19     Actualizar parámetros a y b mediante expresiones 4.2 y 4.3;
20 fin
```

Algoritmo 4.3: El enfoque mPSO con las mejoras propuestas.

A partir del pseudocódigo de este algoritmo es posible notar que varios algoritmos pueden obtenerse producto a la combinación o no de las mejoras propuestas. En ese sentido, obsérvese la Tabla 4.1, en la que también son mostradas los tipos de partículas empleadas

Tabla 4.1 Combinaciones posibles de las mejoras propuestas.

Algoritmo	Partículas	Estrategia de diversidad	Regla de control
mQSO	$n+q$	–	–
mQSOE	$n+q$	–	✓
mPSOD	n	✓	–
mPSODE	n	✓	✓

por cada algoritmo, donde n y q significan partículas neutrales y de tipo quantum, respectivamente. Note además que hemos excluido de los algoritmos a mCPSO debido a su bajo rendimiento en comparación con mQSO, de acuerdo a los resultados reportados por sus autores (Blackwell y Branke, 2006). Finalmente, es importante resaltar que la combinación de nuestras dos propuestas permiten obtener un algoritmo más sofisticado, denominado mPSODE.

4.3 Análisis experimental

En esta sección se analizarán las mejoras propuestas en la sección anterior a través de experimentos computacionales. Para tal propósito, se seleccionó el problema del Movimiento de Picos (MPB) (Branke, 1999b) como escenario de prueba para los experimentos, en particular el escenario 2 (véase Sec. 2.4.2). Como se mencionó en el Capítulo 2, este problema permite obtener diferentes instancias mediante la combinación de determinados parámetros (ej. número de picos, severidades de cambio, frecuencia de los cambios, etc.). En particular, se asumirá que cada instancia de este escenario cambiará 100 veces cada Δe evaluaciones. De manera que cada ejecución terminará cuando el algoritmo ha consumido $100 \cdot \Delta e$ evaluaciones de la función objetivo.

Como medidas de rendimiento se empleó el *error fuera de línea* y el *rendimiento fuera de línea* (Branke, 1999b) (véase Sec. 2.4.4). En relación a la configuración de los parámetros pertenecientes al enfoque *mPSO*, se siguieron las sugerencias reportadas en el trabajo (Blackwell y Branke, 2006), esto es, se usarán en todos los algoritmos 10 enjambres, cada uno con 10 partículas. El algoritmos que utilizan partículas quantum (mQSO, mQSOE) tendrán 5 partículas neutrales y 5 de tipo quantum. Similarmente, mPSOD y mPSODE diversificarán 5 partículas después de cada cambio. En cuanto a la configuración de PSO utilizada por cada enjambre en el movimiento de las partículas neutrales, se han seleccionado las siguientes: $\omega = 0.729$ y $c_1 = c_2 = 1.496$, como fue sugerido por Clerc y Kennedy (2002) para garantizar un rendimiento razonable en sentido general.

Dada la naturaleza estocástica de los algoritmos a analizar, se realizaron 30 ejecuciones

Tabla 4.2 Estructura de los experimentos.

Experimentos	Algoritmos	Factores	Problemas	Factores
Análisis de la estrategia de diversidad	mPSOD	r_{exp}	MPB	*sev*, Δe
Análisis de la regla de control de enjambres	mQSOE	γ_{cut}	MPB	*sev*, Δe
Resultados en otros problemas	mQSOE, mPSOD, mPSODE, mQSO(control)	Estrategia de diversidad, Regla de control de enjambres	MPB	*sev*, Δe, función pico

Tabla 4.3 Funciones picos utilizadas en las instancias del generador Movimiento de Picos.

	Función	Fórmula	Espacio de búsqueda
Func. unimodales	Cone	$f(\mathbf{x}) = \sqrt{\sum_{i=1}^{n} x_i^2}$	$[0,100]^5$
	Sphere	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$[0,100]^5$
	Schwefel	$f(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[0,100]^5$
	Quadric	$f(\mathbf{x}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	$[0,100]^5$
Func. multimodales	Rastrigin	$f(\mathbf{x}) = \sum_{i=1}^{n} (x_i^2 - 10\cos 2\pi x_i + 10)$	$[-5.12,5.12]^5$
	Griewank	$f(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-32,32]^5$
	Ackley	$f(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos 2\pi x_i) + 20 + \exp$	$[-32,32]^5$
	Weierstrass	$f(\mathbf{x}) = \sum_{i=1}^{n} (\sum_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 1/2))]) - n\sum_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$ $a = 0.5, b = 3.0, k_{max} = 20$	$[-0.5,0.5]^5$

con semillas aleatorias diferentes por cada par de instancia problema-algoritmo.

Teniendo en cuenta las consideraciones anteriores, los experimentos fueron organizados como se muestra en la Tabla 4.2:

Los dos primeros experimentos están dedicados a estudiar la influencia de las mejoras propuestas, en particular de los parámetros involucrados *versus* problemas con diferentes severidades de desplazamiento (*sev.*), y frecuencia de cambios (Δe). Por su parte, el último grupo de experimentos añade un factor extra: *Función pico*, que define el tipo de función usada por los picos (véase la Tabla 4.3). En particular, los dos primeros grupos de experimentos emplean la función pico *Cone*.

Los niveles y combinaciones de tratamientos de los factores considerados en estos grupos de experimentos serán especificados conforme se describan cada uno de estos.

Tabla 4.4 Media del error fuera de línea ± error estándar para el algoritmo mPSOD con diferentes configuraciones de r_{exp}. Los resultados han sido obtenidos a partir de 30 ejecuciones en el escenario 2 del generador MPB variando la severidad del desplazamiento y con $\Delta e = 5000$.

r_{exp}	*sev* =0.0	*sev* =1.0	*sev* =2.0	*sev* =3.0
0.3	**0.41 ± 0.05**	**1.45 ± 0.07**	2.50 ± 0.11	3.53 ± 0.13
1.5	0.45 ± 0.07	1.58 ± 0.08	**2.21 ± 0.08**	**3.06 ± 0.14**
3.0	0.62 ± 0.09	1.89 ± 0.08	2.63 ± 0.10	3.32 ± 0.13
4.5	0.53 ± 0.06	2.20 ± 0.11	3.03 ± 0.11	3.56 ± 0.13
6.0	0.59 ± 0.09	2.28 ± 0.11	3.17 ± 0.12	3.96 ± 0.14
rand	0.42 ± 0.06	1.85 ± 0.09	2.67 ± 0.10	3.40 ± 0.13

Los valores en negrita corresponden al mejor algoritmo.

4.3.1 Análisis de la estrategia de diversidad

Como se describió en la Sección 4.2.1, el parámetro r_{exp} define el radio de la hiperesfera usada para uniformemente reinicializar las peores partículas de cada enjambre. La principal utilidad de este tipo de reinicialización después de cada cambio, es resolver la pérdida de diversidad. De manera que, hipotéticamente hablando r_{exp} debería ser de una magnitud similar a la severidad de desplazamiento (*shift severity*) del problema en cuestión. Es decir, un valor bajo de r_{exp} debería ser más efectivo en problemas con baja severidad. Similarmente, debería cumplirse lo mismo para altos valores de r_{exp} frente a altos valores de severidad.

Con el objetivo de comprobar de manera empírica esta hipótesis, se han seleccionado para el presente estudio, los niveles siguientes para el factor r_{exp}: $\{0.3, 1.5, 3.0, 4.5, 6.0\}$. Si se observa con detenimiento, estos valores son relativamente pequeños si se comparan con la extensión del espacio de búsqueda $[0, 100]^5$. Además, se ha incluido una alternativa aleatoria (*rand*), la cual asigna a r_{exp} uno de estos valores de manera aleatoria en cada iteración del algoritmo.

Los resultados obtenidos por el algoritmo mPSOD con distintas configuraciones del parámetro r_{exp} se muestran en las Tablas 4.4 y 4.5. Note que en estos dos grupos de experimentos se varían la severidad del desplazamiento (*sev.*) y la frecuencia de los cambios (Δe) respectivamente. En particular, los experimentos de la Tabla 4.4 usaron $\Delta e = 5000$, mientras que los reportados en la Tabla 4.5 $sev = 1.0$.

Como era de esperar, los mejores resultados para el primer grupo de experimentos (Tabla 4.4) corresponden a las configuraciones con bajos valores del radio de exploración, esto es, $r_{exp} \leq 1.5$. Esto significa que después de los cambios, los enjambres solo necesitan un poco de diversidad ($\approx sev$) para localizar al óptimo desplazado. Por ejemplo, cuando la

Tabla 4.5 Media del error fuera de línea ± error estándar para el algoritmo mPSOD con diferentes configuraciones de r_{exp}. Los resultados han sido obtenidos a partir de 30 ejecuciones en el escenario 2 del generador MPB variando la frecuencia de los cambios y con *sev*=1.0.

r_{exp}	Δe =1000	Δe =2000	Δe =3000	Δe =4000
0.3	**4.23 ± 0.15**	**2.76 ± 0.12**	**2.12 ± 0.09**	**1.82 ± 0.11**
1.5	4.78 ± 0.17	3.11 ± 0.13	2.48 ± 0.12	2.09 ± 0.12
3.0	5.24 ± 0.16	3.45 ± 0.14	2.69 ± 0.09	2.31 ± 0.09
4.5	5.60 ± 0.16	3.68 ± 0.13	2.95 ± 0.12	2.50 ± 0.09
6.0	5.90 ± 0.17	3.85 ± 0.14	3.08 ± 0.14	2.66 ± 0.12
rand	5.10 ± 0.15	3.25 ± 0.12	2.52 ± 0.08	2.13 ± 0.11

Los valores en negrita corresponden al mejor algoritmo.

sev es 0.0 y 1.0, la versión con $r_{exp} = 0.3$ es la que mejor se comporta precisamente porque r_{exp} es el menor de los valores cercanos a esta severidad. Algo similar ocurre para $sev = 2.0$ y 3.0, para los cuales $r_{exp} = 1.5$ es la mejor alternativa. En relación a los experimentos relacionados con Δe (Tabla 4.5), el primer aspecto que salta a la vista es que la configuración $r_{exp} = 0.3$ es robusta frente a variaciones de la frecuencia de cambios. Este resultado, similar a lo que se obtuvo anteriormente, indica la fuerte relación del parámetro r_{exp} con la severidad del desplazamiento (en este caso $sev = 1.0$), al menos para esta frecuencia de cambios.

No obstante la utilidad de los resultados obtenidos en los experimentos anteriores, es importante analizar el rendimiento del algoritmo en el tiempo. Para este fin, se han seleccionado la mejor y la peor configuración ($r_{exp} = 0.3$ y $r_{exp} = 6.0$ respectivamente) sobre la instancia formada por $sev = 1.0$ y $\Delta e = 5000$. En las Figuras 4.5-a) y 4.5-b) se han representado la evolución del error y el rendimiento fuera de línea en función de los cambios del problema. Para obtener estas series temporales se han promediado las series particulares de las 30 ejecuciones realizadas.

A partir de estas gráficas es posible notar que la diferencia más importante entre las configuraciones ocurre en las primeras etapas de la búsqueda. En particular, $r_{exp} = 0.3$ es más precisa al comienzo de la ejecución. Sin embargo, existen ventanas de tiempo para las que $r_{exp} = 6.0$ es temporalmente mejor que $r_{exp} = 0.3$. Por ejemplo, obsérvese lo que sucede cerca del cambio número 30 (Figura 4.5-b): el rendimiento de $r_{exp} = 6.0$ es superior al de la configuración $r_{exp} = 0.3$. Estas son claras indicaciones de que el radio de exploración, como la mayoría de los parámetros de los algoritmos estocásticos, son solo óptimos en algunas etapas del proceso de optimización. Este fenómeno explica en parte porque la configuración *rand* muestra un buen rendimiento en sentido general en los dos

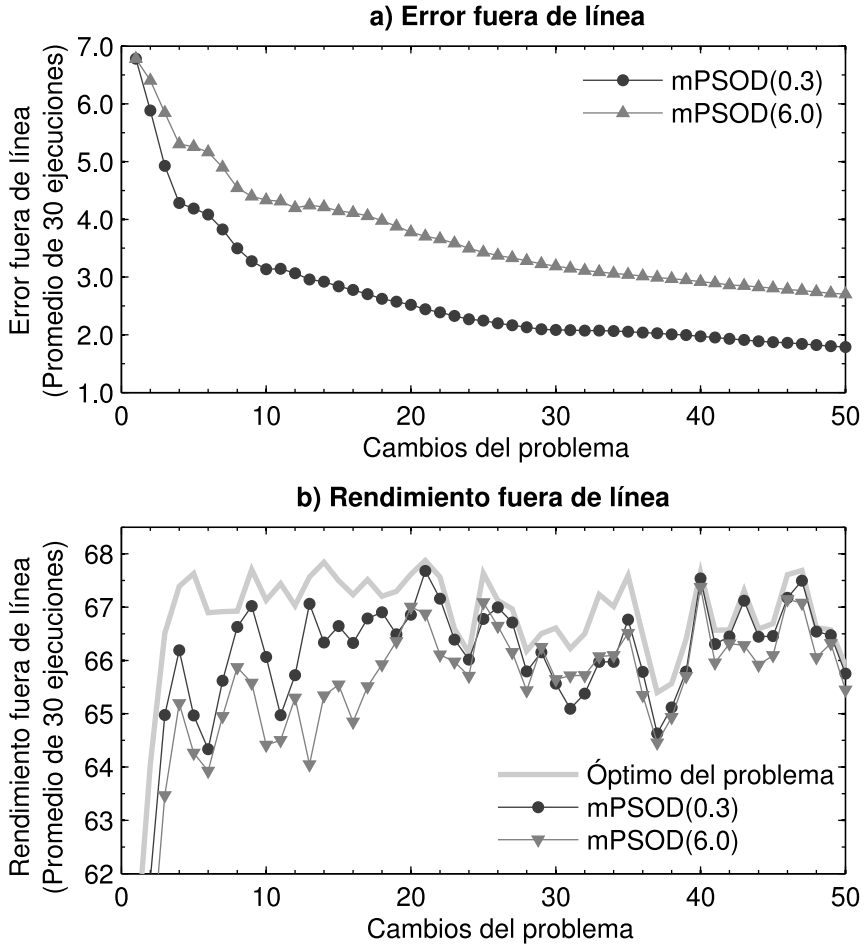

Figura 4.5 Evolución del error y el rendimiento fuera de línea (promedio de 30 ejecuciones) del algoritmo mPSOD en el escenario 2 del MPB ($sev = 1.0$, Δe=5000). Las gráficas muestran la mejor ($r_{exp} = 0.3$) y la peor ($r_{exp} = 6.0$) de las configuraciones.

grupos de experimentos. De hecho, los resultados de esta configuración se acercan (en media) al del resto ($r_{exp} = \{3.0, 4.5, 6.0\}$).

En resumen, para las instancias de problema consideradas en esta sección, las mejores configuraciones de *mPSOD* son $r_{exp} = 0.3$ y $r_{exp} = 1.5$. Un aspecto interesante es el buen rendimiento de la configuración *rand* para los dos factores estudiados.

4.3.2 Análisis de la regla de control

El objetivo de esta sección es el análisis de la influencia del parámetro γ_{cut}, el cual está relacionado con la función de evaluación de la calidad de los enjambres. En ese sentido, resulta claro que este parámetro tiene un impacto significativo con la activación de la regla de control propuesta. Por ejemplo, un valor muy pequeño de γ_{cut} (ej. ≈ 0) conlleva a una función más restrictiva, como consecuencia una gran cantidad de enjambres serán

Tabla 4.6 Media del error fuera de línea ± error estándar para el algoritmo mQSOE con diferentes configuraciones de r_{exp}. Los resultados han sido obtenidos a partir de 30 ejecuciones en el escenario 2 del generador MPB variando la severidad del desplazamiento y con $\Delta e = 5000$.

γ_{cut}	sev =0.0	sev =1.0	sev =2.0	sev =3.0
0.10	0.29 ± 0.04	1.31 ± 0.08	2.76 ± 0.13	3.89 ± 0.20
0.25	$\mathbf{0.26 \pm 0.04}$	1.09 ± 0.07	2.27 ± 0.16	3.30 ± 0.18
0.50	0.33 ± 0.05	$\mathbf{0.98 \pm 0.05}$	1.78 ± 0.10	2.63 ± 0.14
0.75	0.28 ± 0.04	1.12 ± 0.07	1.77 ± 0.09	2.49 ± 0.11
1.00	0.40 ± 0.06	1.19 ± 0.08	1.77 ± 0.09	2.39 ± 0.11
rand	0.27 ± 0.05	1.06 ± 0.06	$\mathbf{1.51 \pm 0.15}$	$\mathbf{2.27 \pm 0.19}$

Los valores en negrita corresponden al mejor algoritmo.

clasificados como *malos* y serán detenidos. En particular, cuando $\gamma_{cut} = 0.0$ la regla para a todos los enjambres en estado de convergencia (incluso al mejor), lo cual es un comportamiento indeseable dado que el algoritmo simplemente no podría iterar y así explotar la mejor solución encontrada hasta el momento. Por el contrario, un alto valor de γ_{cut} implicaría un función de evaluación demasiado flexible, solo unos pocos enjambres sería etiquetados como *malo*. Específicamente, si γ_{cut} =1.0 entonces solo aquellos enjambres que han convergido y con calidad igual o menor que la media serían detenidos.

A partir de este análisis, parece adecuada la elección de los siguientes niveles para el factor γ_{cut}: $\{0.1, 0.25, 0.50, 0.75, 1.0\}$. De manera similar a los experimentos anteriores, se ha decidido incluir la configuración *rand*, la cual asigna aleatoriamente uno de los valores mencionados al parámetro γ_{cut} en cada iteración del algoritmo. Es importante aclarar además que, como fue explicado en la Sección 4.2.2, la regla de control puede ser incluida en cualquier algoritmo basada en el enfoque mPSO. Sin embargo, en los experimentos que siguen se ha decidido usar el algoritmo mQSOE en lugar de mPSODE. La razón de tal elección responde al interés de estudiar la influencia de γ_{cut} de manera independiente a r_{exp} (el cual está incluido precisamente en mPSODE).

Las Tablas 4.6 y 4.7 muestran los resultados de los experimentos desarrollados para las distintas configuraciones del parámetro γ_{cut} y tomando en cuenta nuevamente los factores severidad del desplazamiento y la frecuencia de los cambios. Note que a diferencia de los experimentos de la sección anterior, ahora los resultados muestran menos diferencias entre las alternativas. Sin embargo, cuando los problemas poseen una baja severidad (ej. $sev = 0.0, 1.0$) resulta más efectivo detener a los enjambres que en aquellos problemas con altas severidades ($sev > 1.0$). Es por eso que las variantes con $\gamma_{cut} = \{0.25, 0.5\}$ posee buenos resultados en comparación con aquellos con $\gamma_{cut} = \{0.75, 1.0\}$. Por otro lado,

Tabla 4.7 Media del error fuera de línea ± error estándar para el algoritmo mQSOE con diferentes configuraciones de r_{exp}. Los resultados han sido obtenidos a partir de 30 ejecuciones en el escenario 2 del generador MPB variando la frecuencia de los cambios y con *sev*=1.0.

r_{exp}	Δe =1000	Δe =2000	Δe =3000	Δe =4000
0.1	3.54 ± 0.17	2.18 ± 0.10	1.73 ± 0.11	1.37 ± 0.06
0.25	3.53 ± 0.16	**2.17 ± 0.12**	1.54 ± 0.10	1.30 ± 0.07
0.50	**3.49 ± 0.15**	2.26 ± 0.12	1.71 ± 0.12	1.26 ± 0.07
0.75	3.59 ± 0.15	2.29 ± 0.13	1.81 ± 0.12	1.45 ± 0.08
1.00	3.69 ± 0.14	2.33 ± 0.12	1.71 ± 0.08	1.43 ± 0.06
rand	3.57 ± 0.15	2.21 ± 0.11	**1.52 ± 0.09**	**1.13 ± 0.07**

Los valores en negrita corresponden al mejor algoritmo.

cuando la severidad aumenta, entonces $\gamma_{cut} = 1.0$ muestra una superioridad en relación con el resto de las variantes. Además, la configuración $\gamma_{cut} = 0.1$ presenta (excepto para la instancia $sev = 0.0$) el peor rendimiento. Recuérdese que para este valor de γ_{cut} la regla de control es activada con mayor frecuencia.

En relación a los experimentos donde se varía la frecuencia de cambio, las configuraciones con $\gamma_{cut} = \{0.25, 0.5\}$, obtienen mejores rendimientos en problemas con baja Δe. Sorprendentemente, la variante $\gamma_{cut} = 0.1$ alcanza buenos resultados para estas instancias de problema también. Sin embargo, sus rendimientos son negativamente afectados conforme los valores de Δe se acercan a 4000. Adicionalmente, también hemos decidido estudiar el rendimiento de la mejor y la peor configuración a lo largo del tiempo. Esta vez, los gráficos de la Figura 4.6 muestran el comportamiento de las variantes $\gamma_{cut} = 0.5$, y $\gamma_{cut} = 0.1$ para la instancia de problema con $sev = 1.0$ y $\Delta e = 5000$. La principal diferencia aparece nuevamente en las primeras etapas del proceso de optimización.

Finalmente, se debe destacar el buen rendimiento en sentido general de la variante *rand*, especialmente en las instancias de problema más complejas (ej. aquellas con alta severidad y baja frecuencia de cambio).

4.3.3 Resultados en otros problemas

Con la intención de extender el estudio de mejoras propuestas se ha decidido desarrollar experimentos adicionales sobre problemas con diferentes funciones para los picos. La dinámica de estos problemas es esencialmente la misma que la del escenario 2 del MPB pero con picos definidos por las 8 funciones de la Tabla 4.3. Estas funciones son comunes en el campo de la optimización estacionaria, véase por ejemplo (Yao y Liu, 1996). Note que

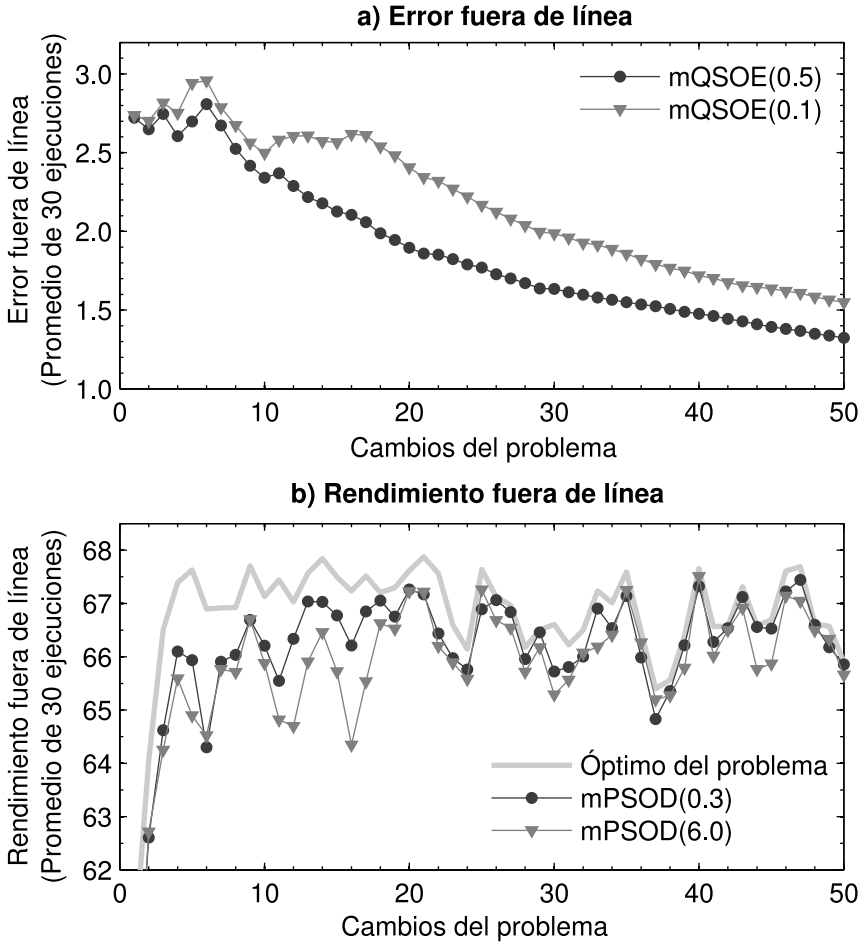

Figura 4.6 Evolución del error y el rendimiento fuera de línea (promedio de 30 ejecuciones) del algoritmo mQSOE en el escenario 2 del MPB ($sev = 1.0$, Δe$= 5000$). Las gráficas muestran la mejor ($\gamma_{cut} = 0.50$) y la peor (y $\gamma_{cut} = 0.1$) de las configuraciones.

las funciones *Cone*, *Sphere*, *Schwefel*, y *Quadric* son unimodales, mientras que *Rastrigin*, *Griewank*, *Ackley*, y *Weierstrass* son multimodales. En general, para todas las instancias de problema se han dejado las configuraciones iniciales del escenario 2 del MPB, como en los experimentos anterior, excepto en las instancias con funciones multimodales, en las cuales se estableció el número de picos igual a 1.

Los algoritmos considerados en esta ocasión son las mejores configuraciones de los experimentos previos. Dado que no existió claramente una configuración ganadora tanto para la estrategia de diversidad y la regla de control para los enjambres, se decidió emplear la configuración *rand* para r_{exp} y γ_{cut}. Sin embargo, el conjunto de valores posibles para *rand* estuvo compuesto por las mejores configuraciones de los experimentos anteriores. En el caso, de la estrategia de diversidad, el parámetro r_{exp} tomará valores del conjunto $\{0.3, 1.5\}$, mientras que la regla de control usará $\gamma_{cut} = \{0.25, 0.50, 0.75\}$. Finalmente, con el propósito de tener una referencia del rendimiento real de nuestros algoritmos, se decidió

incluir el método mQSO en su versión original (Blackwell y Branke, 2006).

Las Tablas 4.8 y 4.9 muestran los resultados de los algoritmos implementados en las nuevas instancias de problema, variando la severidad de desplazamiento y funciones picos unimodales y multimodales, respectivamente. De manera similar las Tablas 4.10 y 4.11 los muestra en función de la frecuencia de los cambios. La última columna de estas tablas incluye la tasa de mejora entre el mejor de nuestros algoritmos y el de control, en este caso mQSO. Esta tasa de mejora es calculada para cada instancia de problema *i* como sigue:

$$TasaMejora_i = 100 \cdot \left(1 - \frac{e_{i,Best}}{e_{i,mQSO}}\right) \tag{4.7}$$

donde $e_{i,Best} = \min\{e_{i,mQSOE}, e_{i,mPSOD}, e_{i,mPSODE}\}$ es el mejor (mínimo) error fuera de línea obtenido por nuestros algoritmos. En caso de que no se obtenga ninguna mejora por parte de nuestros algoritmos, se empleará la siguiente expresión para determinar la tasa de mejora del algoritmo de control en relación al mejor del resto:

$$TasaMejora_i = 100 \cdot \left(1 - \frac{e_{i,mQSO}}{e_{i,Best}}\right) \tag{4.8}$$

A partir de los resultados obtenidos tomando la severidad como factor es posible sacar algunas conclusiones interesantes. Por ejemplo, en las instancias con funciones menos complejas (eg. unimodales) los algoritmos basados en partículas quantum son más eficientes, en especial el algoritmo mQSOE que resulta el mejor en 10 de las 16 instancias con funciones unimodales. En relación a las instancias con funciones multimodales, se puede apreciar una superioridad en sentido general por parte de los métodos que incorporan la estrategia de diversidad después de los cambios (mPSOD y mPSODE). Sin embargo, en la instancia con función *Griewank*, no resulta clara esta superioridad sobre mQSO. En sentido general, la tasa de mejora de nuestros algoritmos sobre estas instancias de problemas en comparación con mQSO es más notable en la función *Weierstrass*, alcanzando un rango entre un 25% a un 92%. Un detalle importante aquí es que la regla de control no siempre resulta útil para todos los problemas. Esto puede ser visto si se observa con detenimiento el bajo rendimiento alcanzado por el algoritmo mPSODE vs. mPSOD en las instancias con función *Rastrigin*.

Las diferencias de los algoritmos propuestos con respecto a mQSO son más marcadas en los experimentos con variación del factor Δe. Las Tablas 4.8 y 4.9 muestran que mQSO es superior en los problemas con función *Sphere* y *Quadric*, aunque estas diferencias no son significativas. Aquí se mantiene el hecho de que para instancias relativamente fáciles los algoritmos que emplean partículas quantum resultan más apropiados, mientras que mPSOD y mPSODE son mejores para funciones multimodales, excepto nuevamente para el

Tabla 4.8 Error fuera de línea ± error estándar de los algoritmos en instancias del escenario 2 del MPB con diferentes severidades de desplazamiento *sev.* y funciones unimodales en los picos.

Problema	*sev.*	mQSO	mQSOE	mPSOD	mPSODE	Tasa de mejora
Cone	0.0	0.34 ± 0.05	**0.28 ± 0.04**	0.49 ± 0.09	0.43 ± 0.06	19.12%
	1.0	1.78 ± 0.06	**1.06 ± 0.08**	1.59 ± 0.10	1.44 ± 0.12	40.24%
	2.0	2.36 ± 0.10	**1.51 ± 0.10**	2.39 ± 0.12	2.25 ± 0.11	35.71%
	3.0	2.88 ± 0.11	**2.27 ± 0.14**	3.24 ± 0.13	2.90 ± 0.14	21.26%
Sphere	0.0	0.23 ± 0.05	**0.21±0.05**	0.21 ± 0.03	0.35 ± 0.06	7.05%
	1.0	0.92 ± 0.06	**0.56 ± 0.04**	0.90 ± 0.04	0.99 ± 0.07	39.68%
	2.0	**1.59 ± 0.06**	1.70 ± 0.10	1.91 ± 0.06	2.07 ± 0.07	*6.47%*
	3.0	**2.60 ± 0.08**	3.09 ± 0.12	3.12 ± 0.09	3.93 ± 0.15	*15.86%*
Schwefel	0.0	**0.39 ± 0.05**	0.43 ± 0.06	0.46 ± 0.06	0.44 ± 0.06	*9.30%*
	1.0	3.00 ± 0.10	**1.80 ± 0.11**	2.72 ± 0.10	2.50 ± 0.12	39.97%
	2.0	3.89 ± 0.15	**2.91 ± 0.11**	3.94 ± 0.13	3.61 ± 0.14	25.31%
	3.0	4.77 ± 0.16	**4.13 ± 0.17**	5.11 ± 0.18	4.69 ± 0.17	13.40%
Quadric	0.0	0.99 ± 0.17	**0.76 ± 0.10**	1.21 ± 0.10	1.19 ± 0.19	23.44%
	1.0	**1.47 ± 0.11**	1.63 ± 0.17	1.54 ± 0.10	1.85 ± 0.16	*4.55%*
	2.0	**1.85 ± 0.12**	2.81 ± 0.11	2.17 ± 0.12	2.46 ± 0.14	*14.75%*
	3.0	**2.68 ± 0.15**	4.32 ± 0.25	3.28 ± 0.10	4.95 ± 0.23	*18.29%*

Los valores en negrita corresponden al mejor algoritmo.

Tabla 4.9 Error fuera de línea ± error estándar de los algoritmos en instancias del escenario 2 del MPB con diferentes severidades de desplazamiento *sev.* y funciones multimodales en los picos.

Problem	*sev.*	mQSO	mQSOE	mPSOD	mPSODE	Tasa de mejora
Rastrigin	0.0	4.86 ± 0.33	4.68 ± 0.35	**3.65 ± 0.30**	4.08 ± 0.25	24.92%
	1.0	5.50 ± 0.47	5.16 ± 0.30	**3.31 ± 0.26**	3.53 ± 0.22	39.90%
	2.0	5.45 ± 0.33	5.64 ± 0.34	**3.69 ± 0.16**	4.19 ± 0.27	32.19%
	3.0	6.13 ± 0.33	6.57 ± 0.24	4.60 ± 0.19	**4.48 ± 0.18**	26.90%
Griewank	0.0	0.46 ± 0.01	0.40 ± 0.01	0.42 ± 0.01	**0.39 ± 0.01**	16.45%
	1.0	0.81 ± 0.01	0.81 ± 0.01	0.78 ± 0.01	**0.74 ± 0.01**	9.23%
	2.0	0.88 ± 0.01	1.07 ± 0.02	0.92 ± 0.01	**0.86 ± 0.01**	2.49%
	3.0	**0.98 ± 0.01**	1.33 ± 0.02	1.05 ± 0.01	1.01 ± 0.02	*2.97%*
Ackley	0.0	1.35 ± 0.16	2.25 ± 0.23	0.70 ± 0.10	**0.67 ± 0.10**	50.63%
	1.0	1.57 ± 0.14	1.80 ± 0.19	0.89 ± 0.05	**0.78 ± 0.05**	49.94%
	2.0	2.27 ± 0.20	3.14 ± 0.10	1.77 ± 0.07	**1.60 ± 0.09**	29.46%
	3.0	3.20 ± 0.14	5.26 ± 0.17	2.38 ± 0.05	**2.21 ± 0.05**	30.82%
Weierstrass	0.0	0.01 ± 0.01	4E-3 ± 2E-3	**1E-3 ± 2E-4**	1E-3 ± 4E-4	92.31%
	1.0	1.28 ± 0.01	**0.73 ± 0.01**	1.12 ± 0.01	0.87 ± 0.01	42.77%
	2.0	1.79 ± 0.01	1.55 ± 0.02	1.82 ± 0.01	**1.34 ± 0.01**	25.46%
	3.0	2.17 ± 0.01	1.88 ± 0.02	2.30 ± 0.01	**1.53 ± 0.01**	29.52%

Los valores en negrita corresponden al mejor algoritmo.

Tabla 4.10 Error fuera de línea ± error estándar de los algoritmos en instancias del escenario 2 del MPB con diferentes frecuencias de cambio Δe y funciones unimodales en los picos.

Problem	Δe	mQSO	mQSOE	mPSOD	mPSODE	Tasa de mejora
Cone	1000	4.27 ± 0.14	**3.58 ± 0.14**	4.57 ± 0.17	4.27 ± 0.21	16.22%
	2000	3.05 ± 0.12	**2.21 ± 0.12**	2.89 ± 0.11	2.70 ± 0.12	27.47%
	3000	2.40 ± 0.08	**1.53 ± 0.07**	2.34 ± 0.12	2.19 ± 0.16	36.21%
	4000	1.95 ± 0.07	**1.13 ± 0.06**	1.88 ± 0.08	1.81 ± 0.14	41.78%
Sphere	1000	**4.02 ± 0.25**	4.34 ± 0.30	6.23 ± 0.37	6.31 ± 0.29	*7.37%*
	2000	2.00 ± 0.10	**1.65 ± 0.14**	2.53 ± 0.15	2.69 ± 0.20	17.47%
	3000	1.62 ± 0.10	**0.97 ± 0.08**	1.72 ± 0.12	1.64 ± 0.15	40.07%
	4000	0.95 ± 0.04	**0.70 ± 0.06**	1.40 ± 0.11	1.30 ± 0.10	25.82%
Schwefel	1000	6.73 ± 0.21	**5.75 ± 0.22**	7.25 ± 0.22	6.77 ± 0.27	14.54%
	2000	4.79 ± 0.14	**3.51 ± 0.16**	4.85 ± 0.18	4.67 ± 0.21	26.69%
	3000	4.01 ± 0.15	**2.71 ± 0.16**	3.82 ± 0.14	3.48 ± 0.14	32.54%
	4000	3.31 ± 0.11	**2.20 ± 0.14**	3.21 ± 0.13	2.88 ± 0.16	33.49%
Quadric	1000	**3.26 ± 0.12**	3.51 ± 0.18	4.40 ± 0.30	4.03 ± 0.24	*7.12%*
	2000	**2.03 ± 0.11**	2.10 ± 0.13	2.28 ± 0.13	2.41 ± 0.18	*3.33%*
	3000	**1.70 ± 0.11**	1.74 ± 0.20	2.01 ± 0.11	1.92 ± 0.13	*2.30%*
	4000	**1.46 ± 0.12**	1.50 ± 0.14	1.62 ± 0.11	1.63 ± 0.13	*2.67%*

Los valores en negrita corresponden al mejor algoritmo.

Tabla 4.11 Error fuera de línea ± error estándar de los algoritmos en instancias del escenario 2 del MPB con frecuencias de cambio Δe y funciones multimodales en los picos.

Problem	*sev.*	mQSO	mQSOE	mPSOD	mPSODE	Tasa de mejora
Rastrigin	1000	6.79 ± 0.37	**5.85 ± 0.34**	5.87 ± 0.30	6.39 ± 0.32	13.82%
	2000	6.77 ± 0.44	5.53 ± 0.36	5.08 ± 0.32	**4.82 ± 0.27**	28.92%
	3000	6.49 ± 0.45	5.19 ± 0.36	4.23 ± 0.20	**4.20 ± 0.21**	35.30%
	4000	5.69 ± 0.44	5.35 ± 0.36	**3.59 ± 0.13**	3.91 ± 0.29	36.89%
Griewank	1000	**2.50 ± 0.05**	2.55 ± 0.05	2.62 ± 0.04	2.65 ± 0.06	0.00%
	2000	**1.37 ± 0.02**	1.46 ± 0.03	1.51 ± 0.02	1.41 ± 0.05	*6.16%*
	3000	1.15 ± 0.02	1.18 ± 0.02	1.12 ± 0.02	**1.04 ± 0.01**	9.99%
	4000	**0.78 ± 0.01**	0.89 ± 0.01	0.90 ± 0.01	0.87 ± 0.01	*10.34%*
Ackley	1000	3.00 ± 0.15	2.49 ± 0.09	2.79 ± 0.06	**2.29 ± 0.12**	23.78%
	2000	2.16 ± 0.12	1.73 ± 0.12	1.78 ± 0.04	**1.49 ± 0.09**	31.00%
	3000	1.95 ± 0.15	1.49 ± 0.12	1.39 ± 0.10	**0.98 ± 0.04**	49.49%
	4000	1.56 ± 0.10	1.34 ± 0.10	1.18 ± 0.07	**0.88 ± 0.05**	44.05%
Weierstrass	1000	2.64 ± 0.01	**2.20 ± 0.01**	2.55 ± 0.01	2.38 ± 0.01	16.65%
	2000	2.03 ± 0.01	**1.49 ± 0.01**	1.91 ± 0.01	1.67 ± 0.01	26.55%
	3000	1.70 ± 0.01	**1.12 ± 0.01**	1.58 ± 0.01	1.30 ± 0.01	34.10%
	4000	1.45 ± 0.01	**0.89 ± 0.00**	1.35 ± 0.01	1.05 ± 0.01	38.73%

Los valores en negrita corresponden al mejor algoritmo.

Tabla 4.12 Resultado de las pruebas Friedman e Iman-Davenport ($\alpha = 0.05$).

	Funciones pico		
Prueba	Unimodales	Multimodales	Todas
Friedman	**8.28E−8**	**1.52E−6**	**1.00E−3**
Iman-Davenport	**1.82E−9**	**1.38E−7**	**7.90E−4**

Los valores en negrita indican que existen diferencias significativas a nivel de grupo (p-valor<0.05).

caso de las instancias definidas por la función *Griewank*. En esta ocasión, los algoritmos que incluyen la regla de control alcanzan mejores resultados en más problemas que en el experimento anterior. Véase por ejemplo que el método mQSOE supera claramente a mQSO en problemas con funciones unimodales, alcanzando una mejora entre el 14.54% al 41.78%.

Con la intención de contribuir la comparación de los métodos propuestos con respecto a mQSO, se han representado gráficamente los resultados de los experimentos previos en las Figuras 4.7 y 4.8. En estas se observan las diferentes tendencias que exhiben los distintos métodos. Note por ejemplo que las curvas correspondientes al algoritmo mQSO están casi siempre por encima de las otras pertenecientes al resto de los métodos. Sin dudas es un argumento adicional que confirma la superioridad de las estrategias propuestas.

No obstante los resultados anteriores, y con la intención de formalizar este análisis, se aplicó la metodología estadística descrita en la Sección 3.2 anterior. En ese sentido, se aplicaron las pruebas de Friedman e Iman-Davenport para determinar diferencias a nivel de grupo. Los resultados de estas pruebas, en términos de p-valor, se muestran en la Tabla 4.12. Adicionalmente, la prueba de Friedman dio los rangos medios de los algoritmos que se muestran en la Tabla 4.13. Específicamente se ha dividido el análisis en tres grupos, de acuerdo al tipo de función. Asimismo, a partir de estos resultados se puede ver que en todos los casos existen diferencias, de manera general, en los algoritmos (e.g. p-valor <0.05).

Para determinar en que pares de algoritmos existen tales diferencias, se aplicó la prueba post-hoc de Holm. En la Tabla 4.14 se muestran los p-valores de estas comparaciones múltiples. A modo de resumen la Figura 4.9 muestra los rangos medios de los algoritmos, así como las relaciones de estos en función de la prueba de Holm.

Como se aprecia, aquí se confirman que las diferencias son significativas desde un punto de vista estadístico. En problemas con funciones unimodales se puede ver que mQSOE es significativamente mejor que el resto. De manera similar, mPSODE es estadísticamente el mejor algoritmo en problemas con funciones pico multimodales. Finalmente, cuando todas las instancias de problema son consideradas, estos algoritmos no poseen diferencias entre sí.

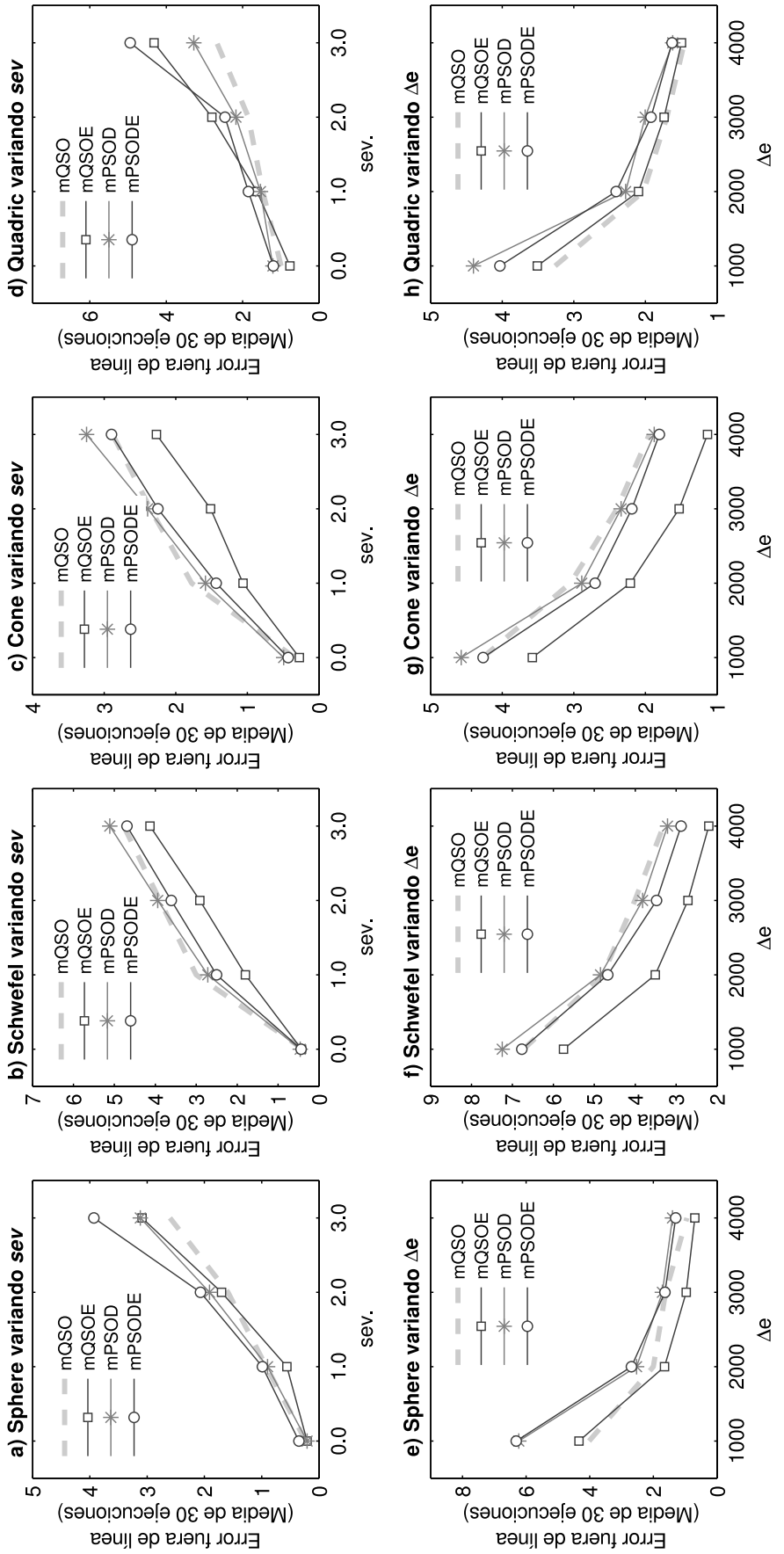

Figura 4.7 Comparativa de los métodos en instancias de problema con funciones pico unimodales.

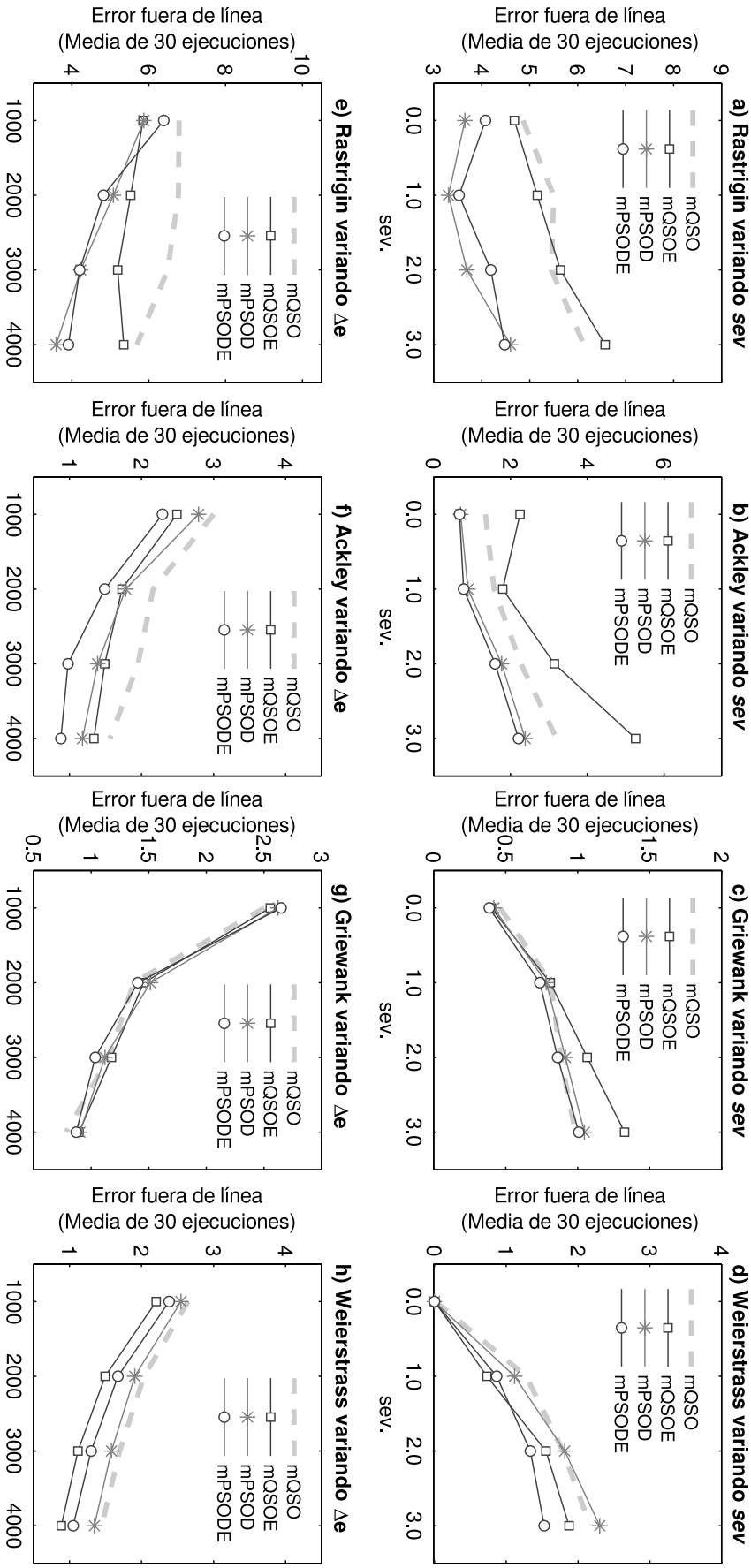

Figura 4.8 Comparativa de los métodos en instancias de problema con funciones pico multimodales.

Tabla 4.13 Rangos medios de los algoritmos a partir de la prueba de Friedman.

	Funciones pico		
Algoritmos	Unimodales	Multimodales	Todas
mQSO	2.297	3.266	2.781
mQSOE	1.484	2.734	2.109
mPSOD	3.266	2.453	2.859
mPSODE	2.953	1.547	2.250

Tabla 4.14 Resultados, en términos de p-valores ajustados, de las múltiples comparaciones empleando la prueba de Holm ($\alpha = 0.05$).

Funciones pico	Algoritmos	mQSOE	mPSOD	mPSODE
Unimodales	mQSO	**3.55E−2**	**1.07E−2**	8.40E−2
	mQSOE		**2.05E−7**	**2.67E−5**
	mPSOD			3.33E−1
Multimodales	mQSO	2.00E−1	**3.55E−2**	**6.05E−7**
	mQSOE		3.84E−1	**1.17E−3**
	mPSOD			**1.99E−2**
Todas	mQSO	**1.62E−2**	1.08E+0	5.98E−2
	mQSOE		**6.09E−3**	1.08E+0
	mPSOD			**3.03E−2**

Los valores en negrita indican que existen diferencias significativas entre los algoritmos (p-valor$<$0.05).

4.4 Conclusiones

En este capítulo se analizaron varios aspectos relacionados con el enfoque PSO multi-enjambre propuesto por Blackwell y Branke (2006). Primeramente se realizó una revisión de los desarrollos más importantes que adaptan al paradigma PSO para optimizar problemas dinámicos. Los dos aspectos que deben ser notados en dicha revisión son los siguientes: 1) la demostrada eficiencia de PSO para lidiar con este tipo de problemas, y 2) la tendencia actual hacia el uso de enfoques multipoblacionales. Algo que no solo ocurre en el caso de PSO, sino también en otros paradigmas computacionales.

Adicionalmente, se propusieron dos mejoras a este enfoque multipoblacional. La primera de ellas estuvo dedicada a diversificar inteligentemente a los enjambres después de los cambios. Específicamente, esta operación se basa en un parámetro que permite la exploración de soluciones cercanas a la mejor solución del enjambre. En este sentido, a través de

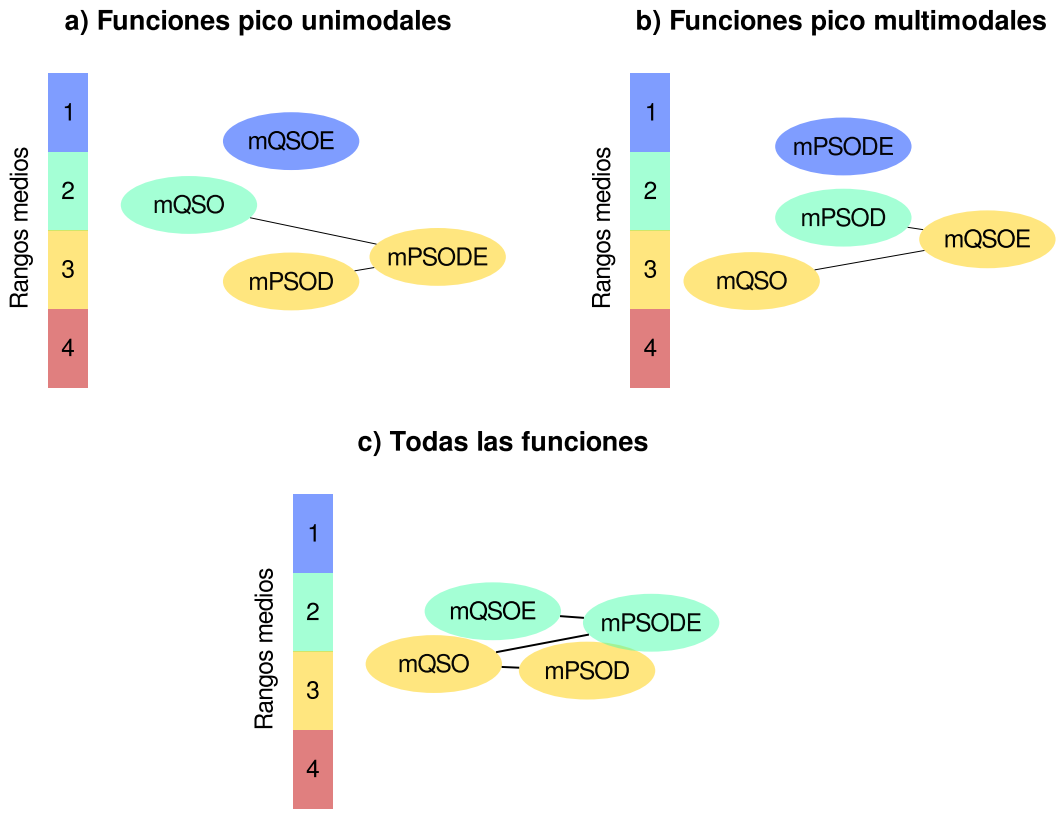

Figura 4.9 Posiciones relativas de los algoritmos de acuerdo a sus rangos medios (prueba de Friedman, $\alpha = 0.05$). Los arcos directos indican, según la prueba de Holm, que no existen diferencias entre los algoritmos involucrados.

los experimentos computacionales desarrollados, se pudo constatar que en problemas con moderada severidad en el desplazamiento de los óptimos, las mejores variantes fueron las que incluyeron un radio de exploración cercano a la severidad de los cambios. Asimismo, en problemas con funciones multimodales esta estrategia resultó ser más eficiente que la diversidad durante la ejecución presentada por el algoritmo mQSO.

La segunda estrategia tuvo como objetivo mejorar la eficiencia del enfoque PSO multi-enjambre a través de la detención de determinados enjambres asociados a regiones poco prometedoras del espacio de búsqueda y con un considerable nivel de convergencia. Los resultados de los experimentos desarrollados en este sentido, confirmaron una mejora significativa en aquellos algoritmos que usaron esta estrategia. La razón de esta mejora es simple: los recursos computacionales (evaluaciones de la función objetivo) no fueron malgastadas en zonas poco prometedoras del espacio de búsqueda. En particular, se pudo observar que la inclusión de esta regla de control es más efectiva cuando el algoritmo se enfrenta a problemas con funciones unimodales en los picos.

No obstante los resultados obtenidos con estas mejoras, en nuestra opinión consideramos que las estrategias propuestas pudieran ampliar su eficacia si se se le aplicaran alguna

técnica de control de parámetros. Hasta ahora ambas estrategias clasifican como adaptativas, aunque la respuesta de la estrategia de diversidad posee una respuesta determinística al ser fijo el radio de exploración. En relación a la regla de control, aunque los parámetros que definen la función de pertenencia son variables en el tiempo, esta variación es de tipo informativa (no evolutiva), de igual forma el criterio para determinar si un enjambre ha convergido puede ser un aspecto dependiente del tipo de función del problema, por lo que el umbral de convergencia (r_{conv}) pudiera ser un parámetro a controlar evolutivamente. En resumen, la auto-adaptación de algunos de estos parámetros pudiera ser útil en este contexto. En el próximo capítulo se verá precisamente como uno de los parámetros del enfoque *quantum* puede ser objeto de auto-adaptación.

5 Rol de la auto-adaptación en ambientes dinámicos

El presente capítulo describe una estrategia, simple y efectiva, para el control auto-adaptativo de la diversidad durante la ejecución (Novoa-Hernández et al, 2013a). En particular se emplea el mismo enfoque multipoblacional del capítulo anterior, pero en esta ocasión se utiliza la *evolución diferencial* como paradigma en las subpoblaciones, en contraste al Enjambre de partículas del enfoque original. La elección de este paradigma responde principalmente a tres cuestiones fundamentales: primero, su efectividad en diferentes escenarios, segundo su fácil implementación (comparable con la del propio PSO), y tercero la posibilidad de explotar, debido a su estrategia de mutación, determinadas interacción es entre sus individuos. Específicamente, la estrategia propuesta está orientada a controlar, a través del radio de diversidad, la generación de individuos quantum en tiempo de ejecución. Adicionalmente, se propone un esquema de interacción entre los individuos convencionales y los de tipo quantum, el cual tiene por objetivo aprovechar las mejoras locales obtenidas por estos últimos. El capítulo termina con el análisis experimental desarrollado, el cual tiene entre sus objetivos mostrar que la auto-adaptación tiene sentido en ambientes dinámicos. Los experimentos fueron desarrollados sobre distintas instancias de problema provenientes de los generadores dinámicos: Movimiento de Picos (MPB), y el Generador Dinámico Generalizado (GDBG). Los resultados mostraron que la inclusión de estas propuestas permiten no solo obtener algoritmos más robustos, sino también igual de competitivos en relación a conocidos referentes del estado-del-arte.

5.1 Autoadaptación en ambientes dinámicos

En esta sección se revisan los progresos actuales sobre la aplicación de la auto-adaptación en ambientes dinámicos. En nuestra opinión estas investigaciones pueden organizarse, de

Tabla 5.1 Posibles grupos de investigaciones que incluyen modelos auto-adaptativos.

	Elementos del algoritmo objeto de auto-adaptación	
Grupos	Paradigma	Enfoques de adaptación dinámica
Grupo I	Sí	No
Grupo II	No	Sí
Grupo III	Sí	Sí

acuerdo a sus objetivos, en tres grupos principales: 1) estudios básicos, 2) trabajos teóricos, y 3) desarrollos avanzados. En el primero se encuentran los trabajos que estudian sí los modelos auto-adaptativos, tal y como han sido concebidos para la optimización estacionaria, resultan igual de efectivos en ambientes dinámicos. En el segundo grupo, aparecen las investigaciones de corte teórico, mientras que en el tercero se incluyen las propuestas de métodos más avanzados que incorporan alguno de los enfoques de adaptación para lidiar con la dinámica del problema. También es posible clasificar estos trabajos atendiendo a qué elemento del algoritmo se auto-adapta: paradigma o enfoque de adaptación dinámica. Dentro de esta clasificación es fácil distinguir los tres grupos que se muestran en la Tabla 5.1.

5.1.1 Estudios básicos

Dado que los primeros (y probablemente más conocidos) paradigmas son las SA-ES y EP, es razonable esperar que la mayoría de los trabajos sobre auto-adaptación en ambientes dinámicos estén basados en estos paradigmas. Por ejemplo, dentro de los primeros trabajos de este grupo se encuentran los estudios de Angeline et al (1996); Angeline (1997). En el primero de estos, los autores se centraron en la comparación de dos reglas de actualización auto-adaptativas: la mutación *Lognormal* (Schwefel, 1981) y la variante *Gaussiana* (Fogel et al, 1991). El paradigma seleccionado como base en esta comparación fue lo que los autores llamaron como programa evolutivo multi-mutacional auto-adaptativo, el cual estaba compuesto por cinco modos de mutación para los individuos. Estos a su vez fueron considerados máquinas de estado finito. Los algoritmos obtenidos fueron analizados experimentalmente en tres escenarios artificiales basados en una rudimentaria tarea de predicción. Dentro de las principales conclusiones de este trabajo, los autores destacan que los algoritmos auto-adaptativos mostraron un rendimiento superior a las variantes adaptativas. En particular, los algoritmos con el esquema de mutación *Gaussiana* fue ligeramente superior a la variante *Lognormal*.

Por su parte, el otro trabajo (Angeline, 1997) no solo empleó escenarios dinámicos distintos en los experimentos, sino también algoritmos diferentes. En este caso, el autor estudia el rendimiento de dos algoritmos basados en el paradigma EP, uno con el modelo auto-adaptativo Gaussiano del trabajo anterior, mientras que el otro incluyendo un modelo adaptativo basado en el fitness, como se muestra a continuación por cada individuo i:

$$\hat{\mathbf{x}}_i = \mathbf{x}_i + \beta \cdot f_i^{(t)} \cdot N(0,1) \tag{5.1}$$

donde β es un factor de escala, en este caso $\beta = 1/D^2$. D es la dimensión del espacio de búsqueda (en este caso $D = 3$), mientras que f_i es el fitness del individuo i (ej. $f_i = f(\mathbf{x}_i)$. Ambos algoritmos fueron comparados en 27 instancias diferentes de problema basados en un modelo 3-dimensional de la función esfera y variando tres factores: 1) tipo de cambio (lineal, circular, y aleatorio), 2) severidad del desplazamiento del óptimo (0.01, 0.1, 0.5), y la frecuencia de los cambios (1, 5, y 10 generaciones). En esta ocasión Angeline concluye que para la mayoría de los escenarios analizados la variante auto-adaptativa experimenta un comportamiento caótico: más preciso al comienzo, pero perdiendo la habilidad de seguimiento durante la ejecución. Por su parte, la variante adaptativa no solo fue capaz de alcanzar un mejor rendimiento, sino que experimentó un patrón de seguimiento estable durante la ejecución. Aunque Angeline no analizó este aspecto con mayor profundidad (ej. estudiando el comportamiento en función del tiempo de los parámetros estratégicos), a juzgar por las bajas frecuencias de los cambios en los escenarios empleados, es posible deducir que la variante auto-adaptativa no tuvo tiempo suficiente para *aprender*, en sus parámetros estratégicos, la dinámica del problema. Por lo general, en estos escenarios caóticos (y muy similares a lo que se conoce como optimización con ruido), la mejor estrategia es localizar rápidamente al óptimo del problema, característica que parece lograrse mejor con la variante adaptativa.

Diferente a las conclusiones de este último trabajo de Angeline, en (Bäck, 1998) afirma que existen algunos escenarios en los que los algoritmos auto-adaptativos pueden resultar efectivos (ej. aquellos con baja severidad y alta frecuencia de los cambios).

Similar al trabajo de (Angeline, 1997), en 1999 (Weicker y Weicker, 1999) lleva a cabo un estudio comparativo interesante entre variantes del paradigma ES, definidas por cuatro tipos de modelos de mutación para el factor de mutación: adaptación uniforme (un solo factor para todas las componentes), adaptación separada (un factor por cada componente del vector solución), adaptación de la matriz de covarianza (CMA) (Hansen y Ostermeier, 1996), y adaptación esférica. Esta última fue propuesta en este trabajo con el objetivo de resolver las dificultades presentadas por los modelos más sofisticados, como la adaptación separada y CMA. En esencia, esta adaptación esférica se basa en dos parámetros estratégicos: m la

mutación más pequeña posible, y w el rango de la mutación. Formalmente, las expresiones para actualizar el factor de mutación mediante este modelo en coordenadas polares son las siguientes:

$$\Delta\varphi = U(0, 2\pi) \tag{5.2}$$

$$\Delta r = m + U(0, w) \tag{5.3}$$

donde $U(0)$ es una función que devuelve valores aleatorios en el rango [0,2π], siguiendo una distribución uniforme. Los cuatro algoritmos fueron estudiados en un escenario artificial con función objetivo en forma de espiral. Como se mencionó anteriormente, el autor encontró que los algoritmos con los modelos más sofisticados (adaptación separada y CMA) eran los de peor rendimiento. En el caso de CMA, su rendimiento mejoraba conforme el número de descendientes se incrementaba ($\lambda > 50$). Sorprendentemente, los modelos simples de mutación, tales como los de adaptación uniforme y la esférica, mostraron un comportamiento estable en los escenarios considerados. Una posible explicación a esta conclusión es que los escenarios empleados en los experimentos poseen una baja dimensionalidad (ej. $D = 2$) y una baja frecuencia de los cambios (ej. igual a 1 generación). Como se puede observar estas configuraciones son muy similares a las usadas por (Angeline, 1997), el cual arribó a conclusiones similares sobre las limitaciones de la auto-adaptación en ambientes dinámicos.

En (Deb y Beyer, 2001) se analiza el comportamiento auto-adaptativo de los algoritmos genéticos de codificación real (RCGAs) con cruzamiento binario simulado (SBX) y la estrategia evolutiva auto-adaptativas (SA-ES). Entre los escenarios empleados en los experimentos, los autores estudiaron, con propósitos ilustrativos una versión dinámica del modelo de esfera con 30 dimensiones. La dinámica de los cambios fueron algo simple: variando el óptimo del problema en el rango $[-1, 1]$ cada 1000 generaciones). Como resultado, los autores encontraron que los RCGAs con SBX *"pueden incrementar rápidamente la diversidad de la población y converger al nuevo óptimo. Primero, la población se vuelve más diversa para aproximarse al óptimo actual y entonces bajan su diversidad para acercarse cada vez más al óptimo actual*". En consecuencia, concluyeron que, incluso en este contexto, existe un comportamiento auto-adaptativo similar entre RCGAs y SA-ES, un hecho que se demostraría posteriormente en (Beyer y Deb, 2001). Esta conclusión parece contradecir los resultados obtenidos por Angeline and Weicker, pero es importante notar que los escenarios empleados son significativamente diferentes a los anteriores. Además, en los experimentos desarrollados en este trabajo, no se prueban otros algoritmos diferentes (ej. variantes no auto-adaptativas), por lo que no se pudo contar con rendimiento base para comparar en que medida son buenos los resultados obtenidos por este paradigma.

Las capacidades de aprendizaje de los factores de mutación de SA-ES y ES con la adaptación de la matriz de covarianza (CMA-ES)(Hansen y Ostermeier, 2001), fueron estudiados por (Boumaza, 2005). Nuevamente una variante dinámica del modelo de la esfera fue empleado en los experimentos, la cual fue combinado con tres tipos de cambios en cuanto a la velocidad de cambio del óptimo del problema: 1) constante, lineal y caótico. La principal conclusión fue que ambos paradigmas son capaces de reflejar la naturaleza de los movimientos de los óptimos en los factores de mutación. En particular, CMA-ES puede también aprender la dirección del movimiento a través de la matriz de covarianza, esto es, los vectores propios dominantes tienen la misma dirección que el óptimo desplazado.

Otra investigación experimental empleando el paradigma SA-ES fue desarrollado por Schönemann (2007). En esta ocasión el autor estudia la influencia del número de factores de mutación que garantice un seguimiento efectivo del óptimo del problema después de la ocurrencia de un cambio en el ambiente. El trabajo concluye que este número debe ser igual a la dimensión del espacio de búsqueda (ej. un factor por cada componente del vector solución). Los experimentos fueron desarrollados inicializando a los algoritmos en el óptimo antes de cada cambio y midiendo sus rendimientos cada 1000 generaciones. En un sentido esta investigación resulta interesante dado que muestra, al menos empíricamente, que SA-ES es capaz de recuperarse a partir de estados de baja diversidad, pero por otro, la inicialización del algoritmo en el óptimo del problema es algo idealista y no considera, como (Boumaza, 2005) otros escenarios dinámicos más complejos.

Recientemente, Chun-Kit y Ho-Fung (2012) estudian el rendimiento de tres variantes de CMA-ES en escenarios dinámicos. Específicamente, las variantes consideradas fueron: la no elitista (μ, λ)-CMA-ES (Hansen y Ostermeier, 2001), la elitista (1+1)-CMA-ES (Igel et al, 2006), y la sep-CMA-ES (Ros y Hansen, 2008). Como base, se empleó la variante adaptativa (1+1)-ES Rechenberg (1973), la cual es particularmente conocida por incluir la regla de actualización 1/5 (véase Sec. 2.3.2). Aquí es importante destacar que la variante (1+1)-CMA-ES está inspirada en esencia en (1+1)-ES. En esta ocasión los cuatro algoritmos fueron analizados en distintas instancias del generador de problemas GDBG (Li et al, 2008), pero sin el tipo de cambio dimensional. Los autores incluyeron otros escenarios variando la severidad de los cambios y la dimensión del espacio de búsqueda. A partir de los resultados obtenidos en estos escenarios los autores concluyeron que las variantes elitistas, (1+1)-CMA-ES e incluso la simple (1+1)-ES mostraron una clara superioridad sobre las variantes no elitistas (y más sofisticadas). Sin embargo, en problemas con alta dimensionalidad ($D > 10$), esta diferencia entre ambos grupos se hizo menor, aunque las variantes no elitistas nunca superaron a las elitistas. Los autores argumentan que estos resultados se deben al hecho de que, cuando la dimensionalidad del problema aumenta, "*la dinámica del problema se vuelve más difícil y emplear estrategias basadas en poblaciones*

son necesarias con el objetivo de alcanzar rendimientos aceptables."

5.1.2 Trabajos teóricos

Desde el punto de vista teórico, aunque pocos, se han hecho algunos avances en relación al estudio de modelos auto-adaptativos en ambientes dinámicos. Por ejemplo, en (Stanhope y Daida, 1999) analizan una versión simplificada de un algoritmo genético (de un solo individuo y con mutación como único operador de variación) en una función dinámica de *matching*. Según los autores, "*las pequeñas perturbaciones de la función objetivo durante el tiempo afecta negativamente en el rendimiento del GA para una tasa de mutación dada*." Incluso, empleando el valor óptimo de tasa de mutación (el cual fue derivado a partir del tipo de cambio y la función objetivo considerada) no resultó en una mejoría significativa en el rendimiento de este algoritmo. En particular, los autores argumentan que esta pobre rendimiento "*implica que las técnicas como la auto-adaptación, la cual se basa en la optimización implícita de la tasa de mutación, pudiera no tener utilidad alguna en este contexto.*"

En contraste al trabajo anterior, en 2002, Arnold y Beyer realizan un análisis interesante en (Arnold y Beyer, 2002) en el que concluyen algo diferente pero en el caso del paradigma $(\mu/\mu,\lambda)$-ES con mutación isotrópica (ej. un solo factor de mutación). Lo nuevo de esta investigación fue el análisis de la adaptación acumulativa del factor de mutación, propuesto por (Hansen y Ostermeier, 2001). El modelo teórico utilizado para analizar este paradigma fue la esfera con dinámica aleatoria. Como resultado se obtuvo una ley de escalamiento entre la distancia de la mejor solución del algoritmo al óptimo del problema, y el factor de mutación. Esta expresión fue resuelta analíticamente con el objetivo de encontrar el valor óptimo del factor de mutación, esto es, que permita alcanzar al óptimo del problema. Los autores comprobaron que empíricamente el algoritmo puede alcanzar dicho valor, concluyendo que la adaptación acumulativa del factor de mutación "*funciona perfectamente*".

Posteriormente, los mismos autores realizaron un estudio similar en el mismo modelo de la esfera pero con dinámica lineal en los cambios (Arnold y Beyer, 2006). En esta ocasión, los autores concluyen que, aunque el factor de mutación de ES no es óptimo para este tipo de cambio, la adaptación llevada a cabo por el modelo auto-adaptativo de este paradigma, asegura un seguimiento adecuado del óptimo del problema.

Hasta aquí, lo que tienen en común los trabajos vistos hasta ahora es el interés por estudiar si la auto-adaptación presente en algunos paradigmas evolutivos tienen sentido en ambientes dinámicos. En otras palabras, los investigadores estuvieron motivados en

responder si los modelos auto-adaptativos, diseñados para la optimización estacionaria, podrían resultar igual de efectivos en ambientes dinámicos. En ese sentido, es importante analizar lo que postularon Beyer y Deb (2001) sobre el comportamiento auto-adaptativo en ambientes dinámicos, en esencia ellos plantean que "*sería importante incrementar la fuerza de la mutación (o varianza de la población) después de un período de encogimiento. Por consiguiente, los algoritmos deberían tener la capacidad de incrementar esta varianza. Una forma de asegurar tal comportamiento es implementar un cierto sesgo hacia el incremento de la varianza de la población*".

Resulta claro que una consecuencia importante de este postulado es que, la auto-adaptación como fue concebida para la optimización en ambientes estacionarios, no es, en sentido general, suficiente para tratar problemas dinámicos. Esto se debe principalmente porque los modelos de estos paradigmas están orientados a converger en conjunto con la población de individuos, y por tanto, el algoritmo falla en generar la diversidad necesaria para seguir eficientemente al óptimo del problema después de un cambio. Por tal razón, varios autores se han orientado hacia la incorporación de estrategias explícitas que, en conjunto con el modelo de auto-adaptación del paradigma en cuestión, ayuden al algoritmo en la fase de seguimiento. En la próxima Sección se describirán estos trabajos.

5.1.3 Trabajos avanzados

No obstante el progreso alcanzados por las investigaciones de las secciones anteriores, otros autores han propuesto algoritmos más sofisticados, aprovechando convenientemente los beneficios de los modelos auto-adaptativos y los enfoques para lidiar con el *problema de la convergencia*.

Quizás los primeros trabajos que aplicaron desarrollos avanzados a problemas dinámicos fueron los de (Bäck, 1997; Bäck, 1999; Grefenstette, 1999) en los que los algoritmos genéticos utilizados incluyeron factores de mutación auto-adaptativos. Específicamente, el último emplea auto-adaptación en los que los autores llamaron tasa de *hipermutación*. Estos trabajos son ejemplos del Grupo II según nuestra clasificación.

Más recientemente, algunas variantes auto-adaptativas del paradigma Evolución Diferencial (DE) (Storn y Price, 1997) han sido adaptadas por Brest et al (2009) y du Plessis y Engelbrecht (2011) a través del empleo de enfoques de adaptación multipoblacionales. En el primer caso, la variante empleada fue el algoritmo jDE propuesta por (Brest et al, 2006, 2007). Originalmente, jDE fue diseñado para resolver problemas estacionarios, siendo su mayor característica la auto-adaptación de los parámetros CR y F (véanse por ejemplo las Secciones 2.2.5 la diversidad durante la ejecución y 2.3.2).

El enfoque adoptado en jDE es un modelo especial de auto-adaptación, esto es, diferente a los empleados por los paradigmas SA-ES y EP. El alcance de la auto-adaptación de jDE es el individuo, y no emplea información global a diferencia de SA-ES la cual emplea la mutación de sus parámetros estratégicos mediante la recombinación (e.g la media) de las realizaciones de los individuos padres. Como consecuencia de esta auto-adaptación *no-recombinada* los parámetros F_i y CR_i experimentan evoluciones independientes, por tanto evitan la convergencia prematura. Otra diferencia con respecto a los modelos clásicos es que en el caso del parámetro F_i este varía aleatoriamente en el rango $[0.1, 1.0]$, los cuales no pudieran ser necesariamente los límites óptimos en algunos escenarios, sin mencionar en los escenarios dinámicos.

Además de emplear varias poblaciones, esta extensión de jDE incluye tres adaptaciones para resolver lidiar con la dinámica del problema: 1) una estrategia basada en la edad de los individuos con el objetivo de reinicializarlos periódicamente, 2) una estrategia de reinicialización para aquellos individuos que se encuentran muy cerca de las mejores soluciones de las subpoblaciones, y 3) el empleo de memorias para llevar a cabo alguna de estas reinicializaciones. El rendimiento de este algoritmo fue analizado en 49 instancias de problema del generador GDBG, empleado en la competición CEC'2009 (Li et al, 2008). Es importante destacar que este método fue el de mejor rendimiento en dicha competición.

Similar a esta investigación, (du Plessis y Engelbrecht, 2011) llevo a cabo un estudio experimental sobre el efecto de un novedoso enfoque multipoblacional, denominado *Competitive Diferential Evolution, CDE*, en tres variantes de DE auto-adaptativas: SDE (Salman, 2007), jDE (Brest et al, 2006), y *Bare Bones* DE (Omran et al, 2009). Este enfoque evalúa secuencialmente cada subpoblación de acuerdo a su calidad, esto es, la mejor es evaluada constantemente hasta que su rendimiento se hace peor que el de otra subpoblación. El rendimiento de cada subpoblación k es calculado en cada generación g como sigue:

$$P_k^{(g)} = \left(\Delta f_k^{(g)} + 1\right)\left(R_k^{(g)} + 1\right) \tag{5.4}$$

donde Δf_k es el error del *fitness* de la mejor solución de la subpoblación k:

$$\Delta f_k^{(g)} = \left| f_k^{(g)} - f_k^{(g-1)} \right|$$

mientras que R_k es el error relativo de la subpoblación k con respecto a la peor de las K subpoblaciones. Por ejemplo, en problemas en donde el objetivo sea encontrar el máximo, este $R_k^{(g)}$ es calculado en cada generación g de la siguiente forma:

$$R_k^{(t)} = \left| f_k^{(t)} - \min_{q=1,\ldots,K} \left\{ f_q^{(t-1)} \right\} \right|$$

Adicionalmente, CDE incluye también una nueva forma de interacción entre las subpoblaciones (*crowding*), llamada por los autores como *Reinitialization Midpoint Check* (RMC). Básicamente, RMC evita que dos subpoblaciones se encuentren explorando un mismo óptimo del problema, basado en el *fitness* del punto medio entre las dos subpoblaciones. Los experimentos de este trabajo fueron desarrollados empleando dos generadores de problemas dinámicos: el Movimiento de Picos (Branke, 1999b) y GDBG (Li y Yang, 2008b). La principal conclusión de este trabajo fue que el enfoque multipoblacional propuesto permite obtener algoritmos de rendimientos competitivos con otros del estado-del-arte (ej. la extensión jDE (Brest et al, 2009). Aunque la intención de los autores fue estudiar la influencia del enfoque propuesto, que no es auto-adaptativo, este trabajo resulta interesante dado que es una clara evidencia de que paradigmas auto-adaptativos en combinación con los enfoques de adaptación adecuados, conlleva a la obtención de algoritmos más efectivos en ambientes dinámicos. En un trabajo posterior (du Plessis y Engelbrecht, 2012), el enfoque multipoblacional propuesto fue combinado con el paradigma DynDE (Mendes y Mohais, 2005), el cual no es auto-adaptativo, pero el algoritmo obtenido superó en rendimiento a otros de la literatura: al propio DynDE, a la extensión de jDE (Brest et al, 2009), mQSO (Blackwell y Branke, 2006), entre otros.

Estas investigaciones basadas en el paradigma DE son buenos ejemplos de trabajos avanzados pertenecientes al Grupo I según nuestra clasificación.

Sin embargo, hasta donde se conoce, no existen en la actualidad técnicas que puedan ser clasificadas en los Grupos II y III, como se puede ver en la Tabla 5.2, la cual resume las trabajos existentes sobre auto-adaptación en ambientes dinámicos. Note que en la tabla los trabajos están organizados según el tipo de investigación, qué elemento del algoritmo es objeto de auto-adaptación, y el tipo de problema empleado para analizarlos. A partir del resumen de esta Tabla 5.2 se puede concluir que:

- La mayoría de los trabajos actuales sobre auto-adaptación en ambientes dinámicos utilizan la auto-adaptación tal y como fue concebida para tratar problemas estacionarios. Dentro de este grupo, sobresalen por su número los trabajos básicos, de los cuales es posible concluir que la auto-adaptación es efectiva en determinados escenarios (ej. alta frecuencia y baja severidad de los cambios), mientras que poco efectiva en otros (ej. aquellos con baja frecuencia de los cambios). En nuestra opinión, una posible causa de este pobre rendimiento en este último caso es que los modelos auto-adaptativos presentes en los paradigmas empleados necesitan de un tiempo considerable para adaptar la búsqueda. Por otro lado, a excepción de (Chun-Kit y Ho-Fung, 2012), en este tipo de investigaciones se emplean escenarios dinámicos en general simples, esto es, caracterizados por un solo tipo de función objetivo y con

dinámica de los cambios que recuerda a la definición de problemas de optimización con ruido.

- En relación a los trabajos teóricos en la actualidad existen muy pocos trabajos, siendo las conclusiones derivadas de estas investigaciones muy limitadas a determinados paradigmas que han sido bien estudiados en el contexto de la optimización estacionaria. En nuestra opinión esta falta de resultados teóricos se debe principalmente a la inherente complejidad de los métodos y problemas dinámicos actuales. No obstante, creemos que algunos de los resultados existentes, o al menos los enfoques adoptados en estos análisis, pueden servir de base para estudiar otros paradigmas y problemas.
- La principal tendencia en los trabajos avanzados es la aplicación de modelos auto-adaptativos provenientes del paradigma utilizado, los cuales han sido diseñados para dotar al algoritmo de una convergencia adecuada en períodos de tiempo en que el problema no cambia. Aunque esta característica ha sido convenientemente aprovechada por varios autores, en nuestra opinión la auto-adaptación pudiera ser más efectiva si es aplicada de manera explícita a los enfoques de adaptación para lidiar con la dinámica de los problemas. En este sentido, no se conoce hasta el momento si un algoritmo con auto-adaptación tanto en el paradigma como en los enfoques de adaptación dinámica puede mostrar una mejor adaptación de la búsqueda y por tanto un mejor rendimiento. Esta y otras interrogantes serán tratadas en las próximas secciones.

5.2 Control auto-adaptativo de la diversidad durante la ejecución

A partir de los trabajos revisados en la sección anterior, es posible observar que la tendencia actual sobre la aplicación de la auto-adaptación en ambientes dinámicos es el empleo de paradigmas auto-adaptativos diseñados originalmente para problemas estacionarios. En otras palabras, la auto-adaptación es una característica heredada del paradigma estacionario seleccionado, el cual ha sido diseñado principalmente para converger. Esta característica obviamente ayuda al algoritmo a encontrar buenas soluciones en períodos cuando el problema no cambia, y en determinados escenarios puede ser útil para la localizar el óptimo después de los cambios (como fue investigado por Schönemann (2007)). Sin embargo, resulta curioso la ausencia de trabajos que apliquen auto-adaptación a los enfoques existentes para ambientes dinámicos. En nuestra opinión, la aplicación de la auto-adaptación en este contexto puede ayudar a explotar mejor las características del problema, en partic-

Tabla 5.2 Distribución de los trabajos atendiendo el tipo de investigación, objeto de la auto-adaptación (paradigma o enfoque de adaptación), y el tipo de escenario dinámico.

Tipo de investigación	Trabajo	Paradigma	Enfoque de adaptación	Escenario dinámico
Estudios básicos	(Angeline et al, 1996)	**Multi-mutational evolutionary algorithm** (†)	–	Tarea de predicción con máquinas de estado finito
	(Angeline, 1997)	**EP** con reglas de actualización *Lognormal* y *Gaussiana*	–	3D-Esfera con cambios lineales, circular y aleatorios
	(Weicker y Weicker, 1999)	**CMA-ES, SA-ES, ES**	–	2D-Espiral con cambios circulares
	(Boumaza, 2005)	**SA-ES**, **CMA-ES**	–	2D-Esfera con velocidad de movimiento: constante, lineal y cuadrático.
	(Schönemann, 2007)	**SA-ES** (regla de actualización Gaussiana istrópica y no istrópica	–	Esfera dinámica, Rastrigin y Ackley.
	(Chun-Kit y Ho-Fung, 2012)	(1+1)-ES (con regla de actualización 1/5), **(1+1)-CMA-ES,** (λ,μ)**-CMA-ES, sep-CMA-ES**	–	GDBG (6 tipos de cambios), Rotational GDBG variando severidad y dimensiones del espacio de búsqueda.
Trabajos teóricos	(Stanhope y Daida, 1999)	**(1+1) GA solo con mutación**	–	Función de *matching*.
	(Arnold y Beyer, 2002)	**CMA-ES**	–	Esfera dinámica (con dinámica Gaussiana).
	(Arnold y Beyer, 2006)	**ES**	–	Esfera dinámica (con cambios lineales).
Trabajos avanzados	(Brest et al, 2009)	**jDE**	Multipoblacional, memorias, diversidad durante la ejecución	GDBG (7 tipos de cambios).
	(du Plessis y Engelbrecht, 2011)	**SDE, jDE, Bare Bone DE**	Multipoblacional	Escenario 2 MPB y GDBG (6 tipos de cambio).
	(Novoa-Hernández et al, 2013a)	DE	Multipoblacional, **diversidad durante la ejecución**	Escenario 2 y GDBG (7 tipos de cambio).

(†) Los paradigmas y enfoques de adaptación en negrita indica la presencia de modelos auto-adaptativos.

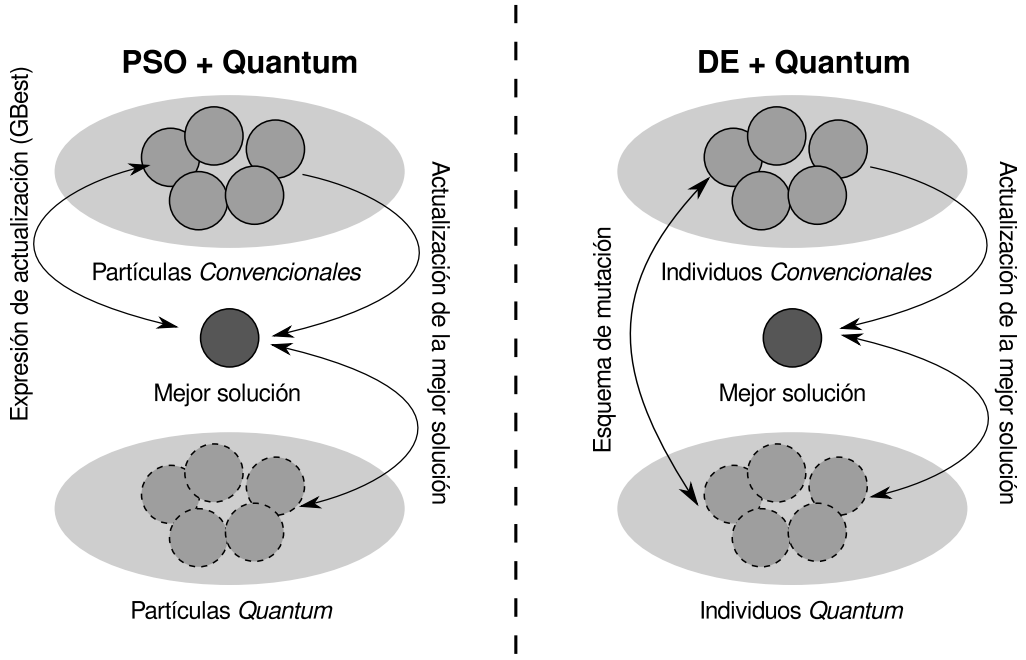

Figura 5.1 Diferencias en los esquemas de interacción entre individuos quantum y los convencionales de los paradigmas PSO y DE.

ular, aquellas relacionadas con la dinámica de los cambios. Hasta el momento, esta área de investigación está prácticamente inexplorada y pudiera abrir nuevos caminos hacia el desarrollo de algoritmos más inteligentes y robustos en ambientes dinámicos.

5.2.1 Descripción de la estrategia propuesta

La estrategia auto-adaptativa propuesta en este apartado tiene por objetivo el control de la diversidad de la población durante la ejecución del algoritmo. A diferencia de las propuestas del Capítulo anterior, esta estrategia varía la diversidad de las subpoblaciones en cada iteración, no solo contribuyendo al proceso de convergencia del algoritmo en los períodos de tiempo en que el problema no varía, sino también al seguimiento de los óptimos desplazados después de la ocurrencia de un cambio.

El enfoque de adaptación que se pretende extender con la estrategia propuesta es el empleo de individuos *quantum*, perteneciente al algoritmo mQSO (Blackwell y Branke, 2006). Específicamente, la idea es controlar inteligentemente la forma en que son generados estos individuos.

Un aspecto importante que surge cuando se emplean simultáneamente individuos convencionales y de tipo quantum en la misma población, es la posible interacción entre ellos. Obviamente este aspecto posee una clara dependencia del paradigma que se emplee para actualizar los individuos convencionales. Un ejemplo de esta situación aparece en la Figura

5.1 para los paradigmas PSO y DE. En el caso de PSO, la única información compartida entre ambos tipos de individuos es la mejor solución de la subpoblación. Esto se debe a que el modelo de actualización empleado es el conocido *Gbest*, de manera que las partículas (convencionales) son actualizadas a partir de la mejor solución (*Expresión de actualización* en la Fig. 5.1). Por otro lado, si se emplea el paradigma DE, entonces esta interacción puede tornarse más rica. Esto es posible gracias a los esquemas de mutación de este paradigma, que por lo general, necesitan de la selección aleatoria de individuos de la población, dentro de los cuales también están los quantum (*Esquema de Mutación* en la Fig. 5.1).

Como se vio en el Capítulo 4, el parámetro que más influye en el proceso de creación de los individuos quantum es el radio de la *nube quantum* (r_{cloud}), el cual guarda una estrecha relación con la severidad de los cambios del problema. En este sentido, Blackwell y Branke (2006) establecen que para garantizar un buen rendimiento del algoritmo es necesario inicializar el parámetro en un valor aproximadamente igual a la severidad de desplazamiento del óptimo del problema. Obviamente, esta severidad es un tipo de información que en escenarios reales puede no conocerse de antemano, por lo que el rendimiento de mQSO y en general de aquellos algoritmos que empleen este enfoque de diversidad, se verán afectados cuando las condiciones del problema cambien o simplemente se desconozcan.

Incluso, aunque pudiera conocerse de antemano la severidad de los cambios, dado que este enfoque está orientado a la diversidad durante la ejecución, no existen garantías de que la diversidad generada por los individuos quantum es la necesaria para el proceso de convergencia del algoritmo (proceso contrario al del seguimiento del óptimo). Lo ideal sería que el algoritmo pudiera auto-ajustarse de acuerdo al estado actual de la búsqueda en diferentes contextos, aumentando así su eficacia. De manera que en este escenario la aplicación de estrategias basadas en la auto-adaptación se justifican plenamente.

En lo que sigue se describirán las principales características de nuestra propuesta a través de los pasos del Algoritmo 5.1.

En esencia la estrategia propuesta asume que cada individuo convencional $\mathbf{y}_i$ está asociado con uno de tipo quantum q_i. De esta manera, la población del algoritmo estará compuesta por la misma cantidad de los dos tipos de individuos, esto es, una mitad de individuos convencionales y la otra de tipo quantum. Formalmente, cada individuo convencional estará definido como $\mathbf{y}_i = \langle \mathbf{x}_i, f_i, rc_i \rangle$, donde $rc_i \in \mathbb{R}^+$ es una realización del parámetro r_{cloud}. En otras palabras, cada $\mathbf{y}_i$ almacena un valor del parámetro r_{cloud} en su genotipo, el cual será utilizado para generar al individuo de tipo quantum asociado.

Intuitivamente, resulta claro que los valores de rc_i deberían ser menores que el radio de exclusión r_{excl}. Por el contrario, si $rc_i > r_{excl}$, es muy probable que algunos individuos de tipo quantum sean generados en determinadas áreas del espacio de búsqueda donde otras subpoblaciones se encuentran optimizando. En consecuencia, el principio de exclusión se

ejecutaría más frecuentemente, lo cual obviamente afectaría la convergencia de las subpoblaciones. Por otro lado, un valor muy bajo para los rc_i puede ser útil para la explotación la mejor solución actual de la subpoblación en cuestión, pero al mismo tiempo pudiera ser insuficiente para seguir al óptimo desplazado después de un cambio. Por tales motivos, se propone relacionar ambos parámetros a través del factor de escala λ, el cual permite inicializar los rc_i como sigue:

$$rc_i = \mathsf{alea}(0,1) \cdot \lambda \cdot r_{excl} \tag{5.5}$$

donde $\lambda \in [0,1]$ es un parámetro fijo. De manera informal, la expresión 5.5 significa que cada realización i será inicializada a partir de un valor aleatorio en el intervalo $[0, \lambda \cdot r_{excl}]$ (como se muestra en el paso 3 del Algoritmo 5.1).

Dentro del ciclo principal del algoritmo, cada individuo convencional genera un individuo quantum usando un valor de rc_i mutado. Específicamente, la mutación es llevada a cabo de la siguiente forma:

$$\widetilde{rc}_i \leftarrow \begin{cases} \mathsf{alea}(0,1) \cdot \lambda \cdot r_{excl} & \text{si } \mathsf{alea}(0,1) < \tau \\ rc_i & \text{por el contrario.} \end{cases} \tag{5.6}$$

donde $\tau \in [0,1]$ es un tasa de mutación predefinida al comienzo de la ejecución del algoritmo. Desde el punto de vista probabilístico, esta expresión de mutación permite obtener un nuevo valor $\widetilde{rc}_i$ con probabilidad τ a partir de una variación aleatoria de r_{excl}. Por otro lado, con probabilidad $1-\tau$ el valor original rc_i permanece invariable.

Luego de aplicar esta operación de mutación, el valor $\widetilde{rc}_i$ es empleado para generar el vector solución del individuo quantum, siendo evaluado en el siguiente paso. Estas operaciones son mostradas por los pasos 20 y 21, Algoritmo 5.1. Particularmente, en el paso 20, la función aleaSphere representa la forma en que el vector solución es perturbado, esto es, uniformemente dentro de la hiperesfera con radio $\widetilde{rc}_i$ y centro en $\mathbf{x}_{best}$ (la mejor solución de la subpoblación). Para más detalles sobre la implementación d esta función, el lector interesado puede consultar los trabajos de Mendes y Mohais (2005) y Clerc (2006).

Más adelante, si el individuo quantum generado es mejor que la mejor solución encontrada en la subpoblación, entonces el valor mutado de $\widetilde{rc}_i$ es heredado por el individuo convencional $\mathbf{y}_i$, siendo la mejor solución también actualizada (ver pasos 22–25 del algoritmo del Algoritmo 5.1). Esta operación está dedicada a preservar las realizaciones exitosas del parámetro rc_i con el objetivo de usarlas en en próximas iteraciones del algoritmo. Por el contrario, si el nuevo individuo quantum no mejora a la mejor solución de la subpoblación, entonces el valor original de rc_i se mantiene invariable, aunque el nuevo individuo quantum se introduce en la población.

En resumen, hasta el momento se ha propuesto un algoritmo multipoblacional con la estrategia auto-adaptativa de diversidad, la cual depende de dos parámetros: la tasa de mutación (τ) y el factor de escala (λ).

5.2.2 Esquema de interacción de los individuos

La interacción entre individuos convencionales y de tipo quantum ocurre en varias etapas del proceso de optimización. Primeramente, ellos intercambian información a través del operador de mutación definido por DE (que puede seleccionar individuos quantum para crear convencionales), y a través de la actualización que ambos tipos hacen de la mejor solución de la subpoblación (la cual es utilizada en la generación de los quantum). En segundo lugar, la estrategia de diversidad propuesta se puede ver como un tipo de interacción entre estos dos tipos de individuos, ya que los convencionales *crean* a los quantum, mediante el uso del parámetro rc_i almacenado en su genotipo.

En ese sentido, note que hasta ahora la preservación de valores exitosos del parámetro rc_i depende únicamente de la calidad global del individuo quantum generado con dicho parámetro. Decimos global pues la referencia que se toma para evaluar la calidad del individuo es la mejor solución de la subpoblación a la que este pertenece. Sin embargo, también pudiera ser útil aprovechar las posibles mejoras locales que el individuo quantum pueda hacer, esto es, evaluar la calidad de un quantum tomando como referencia la calidad de su variante convencional. Algo similar a lo que ocurre en el esquema original de DE, en donde si el nuevo descendiente es mejor que su padre, entonces el primero toma el lugar del segundo en la población.

De manera que, se puede proponer la siguiente interacción: si el nuevo quantum generado es mejor que su *generador*, entonces este último es reemplazado por el quantum, heredando también el exitoso $\widetilde{rc}_i$). Esta interacción es mostrada en los pasos 26–28 de la Figura 5.1. La consecuencia más clara de este interacción propuesta es el decrecimiento de la diversidad de la subpoblación debido a la presencia en esta de dos soluciones idénticas al mismo tiempo. Hipotéticamente, esta disminución de la diversidad pudiera ser particularmente útil en escenarios donde se requiera acelerar el proceso de convergencia, por ejemplo, problemas con una baja severidad de los cambio y/o compuesto por funciones unimodales.

5.2.3 Detección de los cambios

Como en el capítulo anterior, asumiremos que los cambios del problema son de alguna manera detectables por el algoritmo, específicamente mediante la reevaluación de la mejor

solución del algoritmo en cada generación (iteración). De manera que si existiese alguna inconsistencia entre los valores previo y actual del fitness en dicha solución, entonces el algoritmo asumirá la ocurrencia de un cambio. Este mecanismo de detección de cambios ha sido brevemente denotado por el paso 15 en el algoritmo de la Algoritmo 5.1.

Una vez detectado un cambio, los individuos convencionales son reevaluados (pasos 33–35 en el Algoritmo 5.1), mientras que las realizaciones rc_i se mantienen invariables. Aunque en este caso, una simple reinicialización de rc_i pudiera resultar adecuada y fácilmente ejecutada (ej. a través de la expresión 5.5), hemos decidido dejar este ajuste a la estrategia auto-adaptativa propuesta. Nuestra intención es analizar si dicha estrategia sería capaz de reaccionar y generar la diversidad requerida no solo en la fase de explotación, sino también como respuesta a los cambios. Precisamente, uno de los experimentos de la sección siguiente estará dedicado a esta cuestión.

5.2.4 Algoritmos propuestos

Para un perspectiva algorítmica, el Algoritmo 5.1 muestra como la estrategia auto-adaptativa y el esquema de interacción propuestos en esta sección pueden ser incluidos en un algoritmo multipoblacional basado en el paradigma Evolución diferencial.

Dado que varias configuraciones pueden ser obtenidas a partir de la combinación (o no) de nuestras propuestas, es necesario identificarlas de una manera más clara. En ese sentido, en lo que sigue llamaremos mSQDE al algoritmo que incluye solo la estrategia auto-adaptativa de diversidad. Por su parte, el algoritmo que incorpore ambas, la estrategia de diversidad y el esquema de interacción a nivel de individuos, será denotado como mSQDE-i. Note que el esquema de interacción es activado en el pseudo-código si la variable *interaccion* vale *verdadero* (véase paso 26 del Algoritmo 5.1). Por motivos de comparación, también se ha considerado una versión básica del enfoque propuesto, el cual será llamado *mQDE*. En este algoritmo, nuestras propuestas no están presentes, solo la generación de individuos quantum de manera *adaptativa* (ej. empleando un valor de r_{cloud} fijo en el tiempo), como el enfoque original mQSO.

5.2.5 Análisis experimental

En esta sección se analizará experimentalmente los algoritmos propuestos. Los experimentos se organizaron de acuerdo a la estructura de la Tabla 5.3.

En el primer grupo, se analiza a través de un experimento multifactorial el impacto de los parámetros que controlan la estrategia propuesta. Adicionalmente, el esquema de

```
1  para cada subpoblación P_k hacer
2      para cada individuo convencional y_i ∈ P_k hacer
3          Inicializar aleatoriamente x_i ∈ Ω y rc_i con la expresión 5.5;
4          Evaluar f_i ← f(x_i);
5          Actualizar la mejor solución de P_k con y_i;
6      fin
7      para cada individuo quantum q_i ∈ P_k hacer
8          Inicializar aleatoriamente x_qi ∈ Ω;
9          Evaluar f_qi ← f(x_qi);
10         Actualizar la mejor solución de P_k con q_i;
11     fin
12 fin
13 mientras no condición de parada hacer
14     Aplicar principio de exclusión;
15     si no se ha detectado un cambio en el ambiente entonces
16         para cada subpoblación P_k hacer
17             para cada individuo convencional y_i ∈ P_k hacer
18                 Actualizar y evaluar posición de y_i de acuerdo al paradigma DE (ej. Alg.
                   2.3);
                   // Estrategia auto-adaptativa de diversidad
19                 Con el valor rc_i de y_i obtener un el paramétro mutado r̃c_i mediante la
                   expresión 5.6;
                   // Generación de individuos quantum
20                 Asignar x_qi ← aleaSphere(x_best, r̃c_i);
21                 Evaluar f_qi ← f(x_qi);
                   // Actualizar memorias y herencia de r̃c_i
22                 si f_qi ≽ f_best entonces
23                     Actualizar la mejor solución de P_k con q_i;
24                     Reemplazar el parámetro rc_i de y_i con r̃c_i ;
25                 fin
                   // Esquema de interacción de individuos
26                 si interaccion = verdadero y f_qi ≽ f_i entonces
27                     Asignar y_i ← ⟨x_qi, f_qi, r̃c_i⟩ ;
28                 fin
29             fin
30         fin
31     fin
32     si no
33         para cada subpoblación P_k hacer
34             Reevaluar individuos convencionales;
35         fin
36     fin
37 fin
```

Algoritmo 5.1: Principales pasos de los algoritmos mSQDE y mSQDE-i

Tabla 5.3 Estructura de los experimentos.

Experimentos	Algoritmos	Factores	Problemas	Factores
Efecto de la tasa de mutación y el factor de escala	mSQDE, mSQDE-i	τ, λ, esquema de interacción	MPB	*sev*, Δe, función pico
Análisis de la auto-adaptación del parámetro rc_i	mSQDE, mSQDE-i	esquema de interacción	MPB	*sev*, función pico
Comparación con otros algoritmos	mQDE, mSQDE, mSQDE-i	estrategia auto-adaptativa, esquema de interacción	MPB	*sev*, Δe, función pico
			GDBG	función pico, tipo de cambio

interacción es también analizado a partir de la presencia de los algoritmos mSQDE (sin el esquema) y mSQDE-i. Como problemas test, se emplearon varias instancias del generador Movimiento de Picos (MPB), variando los factores: severidad del desplazamiento (*sev*), la frecuencia de los cambios (Δe), y el tipo de función para los picos.

En el segundo grupo de experimentos, se recolectó información acerca de la evolución del parámetro rc_i en el tiempo, con el objetivo de observar si la estrategia propuesta es capaz de adaptar realmente el parámetro rc_i durante la ejecución. En este caso se emplearon instancias del generador MPB, variando la severidad del desplazamiento y el tipo de función para los picos.

Finalmente, en el último grupo de experimentos se comparan los algoritmos propuestos con otros representantes de la literatura. También se incluye la variante básica mQDE, con el objetivo de observar si la presencia de la estrategia auto-adaptativa propuesta mejora significativamente el rendimiento del algoritmo. En estos experimentos se emplearon instancias del generador MPB variando los factores *sev*, Δe, y la función para los picos. Adicionalmente, con la intención de favorecer la comparación con el algoritmo jDE (Brest et al, 2009), se incluyeron las instancias del generador GDBG, las cuales fueron obtenidas variando el tipo de función para los picos y el tipo de cambio.

Es importante especificar que las funciones pico empleadas por las instancias del generador MPB, son las que se muestran en la Tabla 5.4, siendo 10 el número de picos. Como se aprecia en esta tabla, se emplearon cuatro funciones unimodales y cuatro multimodales de distinta complejidad. Más detalles sobre los niveles de los factores a estudiar, serán introducidos conforme se describan los experimentos.

A menos que se establezca lo contrario, en los experimentos se realizaron 20 ejecuciones por cada par algoritmo/problema con diferentes semillas aleatorias en el caso de los

Tabla 5.4 Funciones picos utilizadas en las instancias del generador Movimiento de Picos.

	Función	Fórmula	Espacio de búsqueda
Func. unimodales	Cone	$f_1(\mathbf{x}) = \sqrt{\sum_{i=1}^{n} x_i^2}$	$[0,100]^5$
	Sphere	$f_2(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$[0,100]^5$
	Schwefel	$f_3(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[0,100]^5$
	Quadric	$f_4(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	$[0,100]^5$
Func. multimodales	Rastrigin	$f_5(\mathbf{x}) = \sum_{i=1}^{n} (x_i^2 - 10\cos 2\pi x_i + 10)$	$[0,100]^5$
	Ackley	$f_6(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1)$	$[-150,150]^5$
	Griewank	$f_7(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600,600]^5$
	Alpine	$f_8(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1 x_i \rvert$	$[0,100]^5$

algoritmos. Como medida de rendimiento se seleccionó el mejor error antes del cambio (Li et al, 2008) (véase Sec. 2.4.4.) En todos los casos, los algoritmos estuvieron compuestos por 10 poblaciones de 10 soluciones cada una, con el mismo número (5) de individuos convencionales y de tipo quantum.

Efecto de la tasa de mutación y el factor de escala

En esta sección el rendimiento de los algoritmos mSQDE y mSQDE-i es analizado empleando diferentes configuraciones para los parámetros τ y λ. El objetivo es obtener algunos elementos para encontrar una configuración adecuada de esos parámetros. En ese sentido, se decidió desarrollar experimentos multifactoriales teniendo en cuenta a estos parámetros como factores, incluyendo además algunos relacionados con las instancias del escenario 2 del MPB. En este último caso, se consideraron los siguientes factores y niveles: $sev = \{1.0, 5.0, 10.0\}$, $\Delta e = \{1.0, 5.0, 10.0\}$, y $F_{pico} = \{Cone, Rastrigin\}$.

Por otro lado, los niveles de los factores τ y λ fueron: $\tau = \{0.0, 0.3, 0.6, 1.0\}$, y $\lambda = \{0.25, 0.50, 0.75, 1.0\}$, respectivamente. Es importante destacar que cuando $\tau = 0.0$ el parámetro rc_i permanece fijo, esto es, no es posible variarlo durante la ejecución. Así, los algoritmos obtenidos son similares a la variante básica *mQDE*. Por otro lado, cuando $\tau = 1.0$ la variación de rc_i se ejecuta siempre, de manera que la trasmisión de información entre iteraciones del algoritmo resultará en cierto modo de poca utilidad. En el caso del parámetro λ ocurre algo similar, las consecuencias de los valores extremos de este factor se

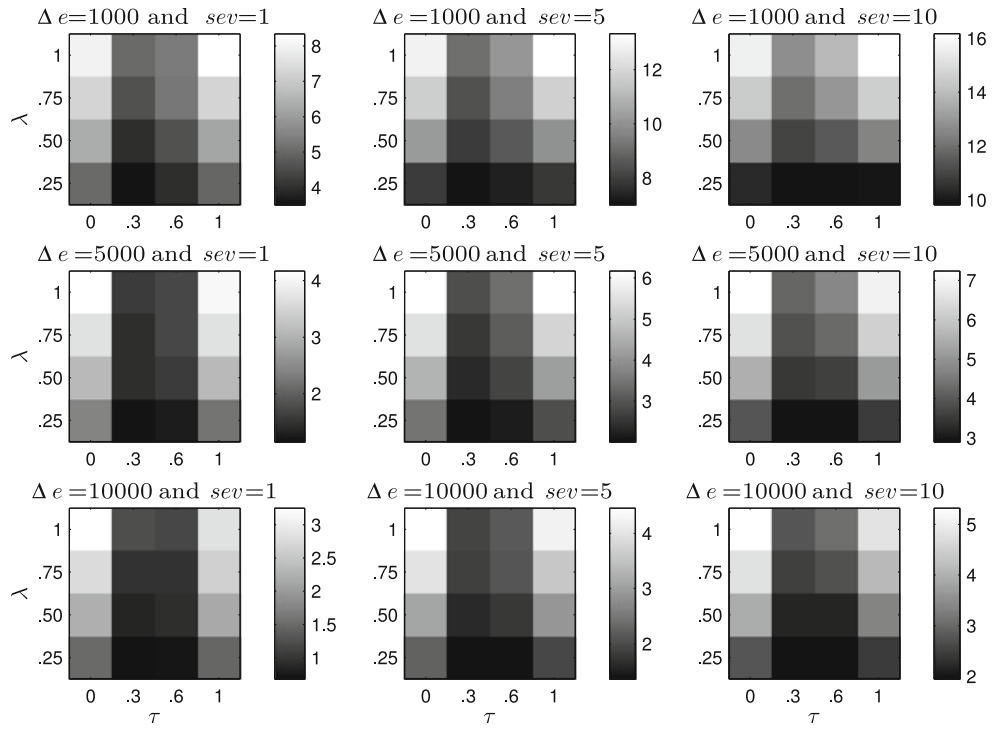

Figura 5.2 Mapas de color a partir del mejor error antes del cambio (avg. 20 ejecuciones), del algoritmo mSQDE en diferentes instancias del Problema Movimiento de Picos con función *Cone*. Cada combinación de los parámetros τ y λ corresponde a una instancia de mSQDE.

pueden identificar fácilmente. Por ejemplo, si $\lambda = 1.0$ la máxima variación de rc_i es igual a r_{excl}. Por otro lado, un valor cercano a 0 (ej. $\lambda = 0.25$) da como resultado una solución similar a la mejor solución de la población.

Con el objetivo de facilitar la comprensión del efecto de los parámetros τ y λ se han empleado mapas de color para representar los resultados de múltiples experimentos. Aunque menos precisos que otras herramientas visuales (ej. tablas cruzadas), hemos considerado emplear estos mapas de color para brindar una panorámica general de los resultados. Las Figuras 5.2 y 5.4 corresponden a los resultados obtenidos por las distintas variantes del algoritmo mSQDE, en problemas con funciones pico *Cone* y *Rastrigin*, respectivamente. De igual forma, los resultados para las instancias del algoritmo mSQDE-i están representadas por las Figuras 5.3 y 5.5. Es importante notar que en todas las figuras, cada gráfica contiene 16 resultados del error antes del cambio, correspondientes a las combinaciones posibles de los factores τ y λ en una instancia de problema en particular, la cual a su vez está determinada por los factores Δe y *sev*. En ese sentido, los mejores valores de esta medida de rendimiento están asociados a las zonas más oscuras de las gráficas, mientras que las más claras corresponden a los peores errores. Para una información más específica sobre los valores absolutos del error, cada gráfica incluye una escala de colores que relaciona tonalidades con los valores reales de esta medida.

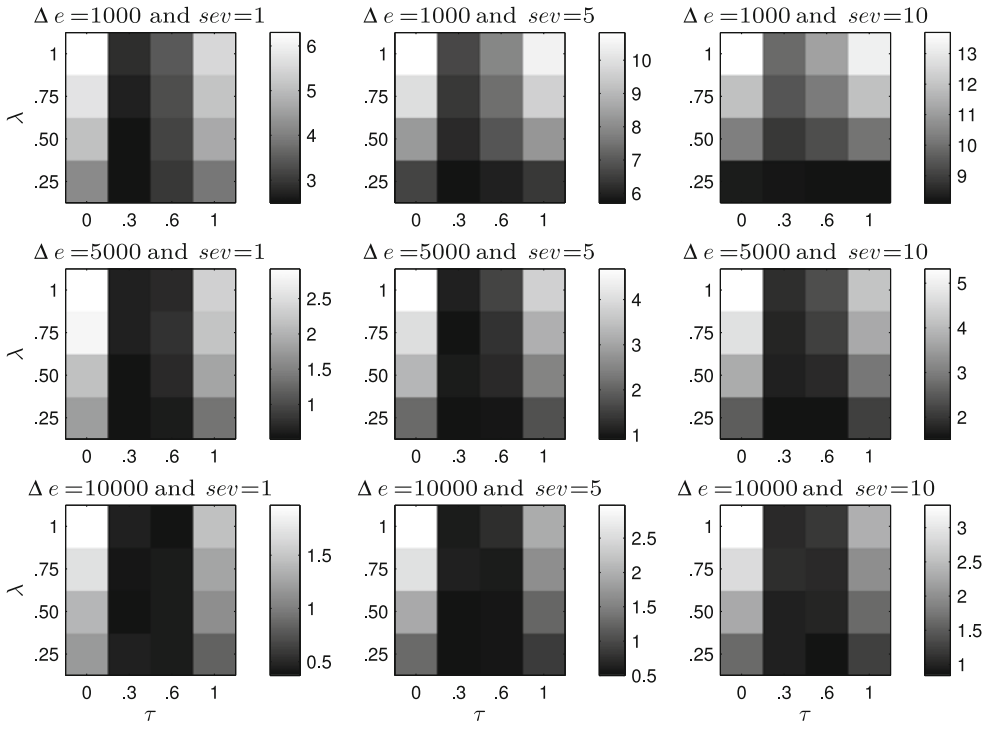

Figura 5.3 Mapas de color a partir del mejor error antes del cambio (avg. 20 ejecuciones), del algoritmo mSQDE-i en diferentes instancias del Problema Movimiento de Picos con función *Cone*. Cada combinación de los parámetros τ y λ corresponde a una instancia de mSQDE-i.

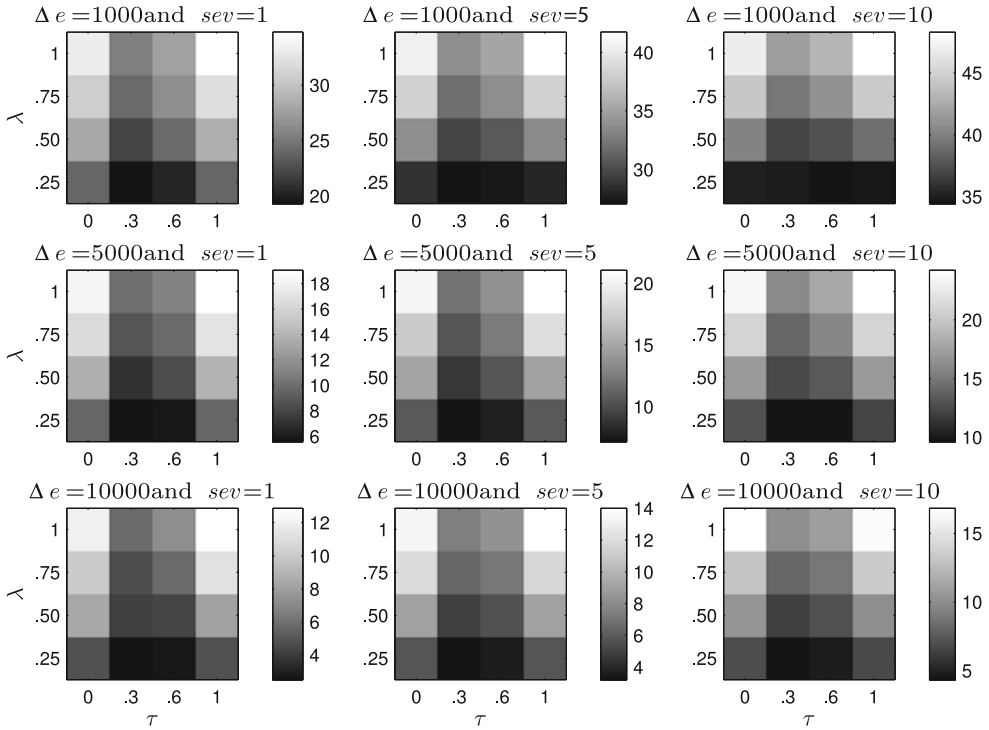

Figura 5.4 Mapas de color a partir del mejor error antes del cambio (avg. 20 ejecuciones), del algoritmo mSQDE en diferentes instancias del Problema Movimiento de Picos con función *Rastrigin*. Cada combinación de los parámetros τ y λ corresponde a una instancia de mSQDE.

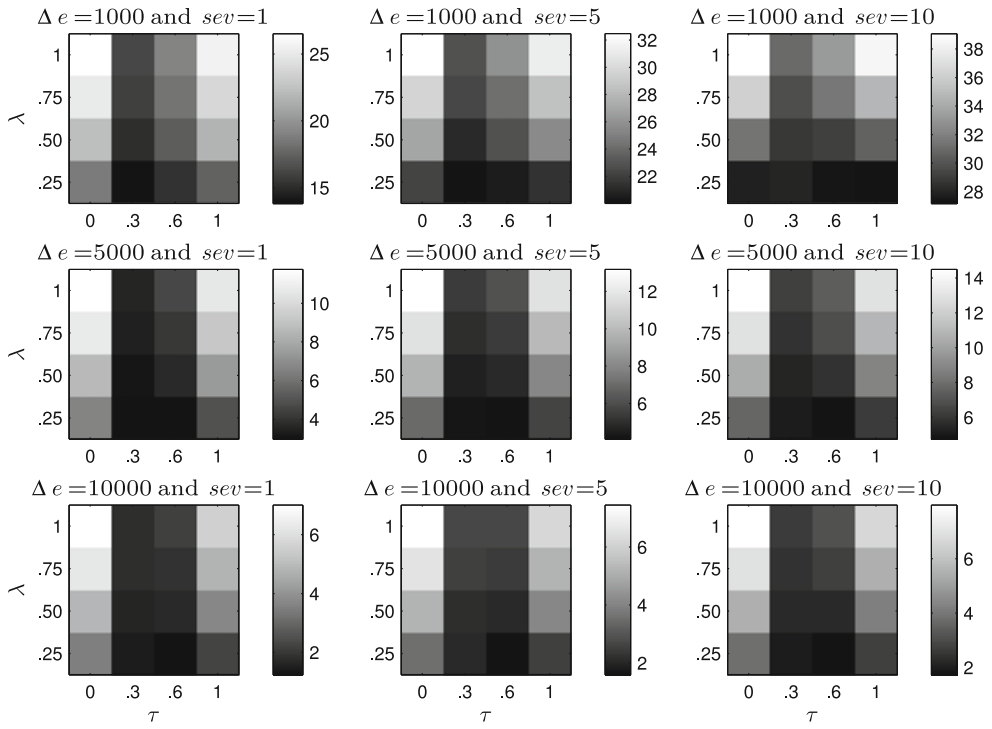

Figura 5.5 Mapas de color a partir del mejor error antes del cambio (avg. 20 ejecuciones), del algoritmo mSQDE-i en diferentes instancias del Problema Movimiento de Picos con función *Rastrigin*. Cada combinación de los parámetros τ y λ corresponde a una instancia de mSQDE-i.

A partir de estos resultados el primer aspecto a resaltar es que los mejores rendimientos corresponden a las configuraciones $\tau = \{0.3, 0.6\}$ (véase la zonas oscuras del centro de las gráficas). Esencialmente, esto significa que la adaptación *inteligente* del parámetro rc_i tiene sentido en todos los escenarios evaluados, pues estas configuraciones poseen mejores rendimientos que aquellas que dejan fijo al parámetro rc_i (ej. $\tau = 0.0$) o las que lo varían siempre (ej. $\tau = 1.0$).

También resulta interesante los resultados en los escenarios con $\Delta e = 1000$ y $sev = 10$ en todos los casos (Figuras 5.2-5.5). Aquí la mejor estrategia para los algoritmos es emplear $\lambda = 0.25$, independientemente cuán frecuente es llevada a cabo la mutación de rc_i. Note que estos escenarios con una baja frecuencia en los cambios y una alta severidad de desplazamiento son los más caóticos. En este contexto, y dado el poco tiempo para converger, es una buena estrategia tratar de intensificar la explotación de la búsqueda (ej. empleando un valor bajo para λ).

Finalmente, si se comparan los valores mínimos de las barras de colores entre las Figuras 5.2 vs. 5.3, y 5.4 vs. 5.5, se puede ver que el algoritmo mSQDE-i obtiene mejores resultados que mSQDE. Esto significa que, al menos para estas instancias de problema, el esquema propuesto para la interacción entre individuos (donde los quantum reemplazan a los convencionales cuando una mejora es obtenida), resulta adecuada.

Análisis de la auto-adaptación del parámetro rc_i

Con la intención de brindar algunas ideas del comportamiento interno de la estrategia auto-adaptativa propuesta, en este apartado se estudiará como el parámetro rc_i evoluciona durante la ejecución. Esto nos ayudará a descubrir, entre otros aspectos, si la estrategia propuesta es capaz de variar inteligentemente dicho parámetro durante el proceso de búsqueda. Debido a que cada individuo convencional posee una realización de este parámetro, resulta algo complejo estudiar estas realizaciones de manera independiente. Otro elemento que torna complejo este análisis es la presencia de múltiples poblaciones en los algoritmos propuestos (ej. mSQDE y mSQDE-i).

Por tales razones, en lo que sigue, se considerará una versión simplificada de estos métodos, la cuál solo tendrán una sola población de 10 individuos (5 convencionales que generan 5 de tipo quantum), mientras que $\tau = 0.5$ y $\lambda = 0.25$. Básicamente, la idea es seguir la evolución en el tiempo de las cinco realizaciones de rc_i mediante el promedio de las mismas. Esta media, denotada por $\overline{rc}^{(g)}$ es recolectada en cada generación (iteración) g del algoritmo como se muestra a continuación:

$$\overline{rc}^{(g)} = \frac{1}{5}\sum_{i=1}^{5} rc_i^{(g)} \tag{5.7}$$

donde $rc_i^{(g)}$ es el valor del parámetro rc perteneciente al individuo convencional $\mathbf{y}_i$ en la generación g.

En este análisis, los problemas seleccionados son los mismos empleados en la sección anterior, pero en este caso todos poseen un solo pico y $\Delta e = 5000$. Tales simplificaciones han sido hechas con el objetivo de simular la situación en que una subpoblación se encuentra optimizando un solo pico. Además, dado que existe una relación estrecha entre el parámetro rc_i y la severidad del desplazamiento de los problemas, hemos estudiado tres posibles escenarios definidos a través de los niveles $sev = \{1.0, 5.0, 10.0\}$. Hipotéticamente, para tales escenarios es de esperar que nuestra estrategia de diversidad auto-adapte el parámetro rc_i de acuerdo a la severidad de los cambios (ej. un valor bajo de rc_i cuando $sev = 1.0$ y uno alto cuando $sev = 10.0$).

La evolución del parámetro $\overline{rc}^{(g)}$ para los algoritmos propuestos son mostrados en la Figuras 5.6 y 5.7, correspondientes a los primeros 20 cambios del problema. Si se observan con detenimiento estas gráficas, es posible observar que nuestra hipótesis sobre la auto-adaptación del parámetro rc_i es cierta. Véase por ejemplo que en ambos algoritmos el parámetro $\overline{rc}$ posee diferentes valores de acuerdo a la severidad del desplazamiento en los distintos escenarios considerados. Por ejemplo, cuando la severidad es baja, la estrategia no interfiere con la convergencia del algoritmo. Asimismo, puede observarse que la estrategia

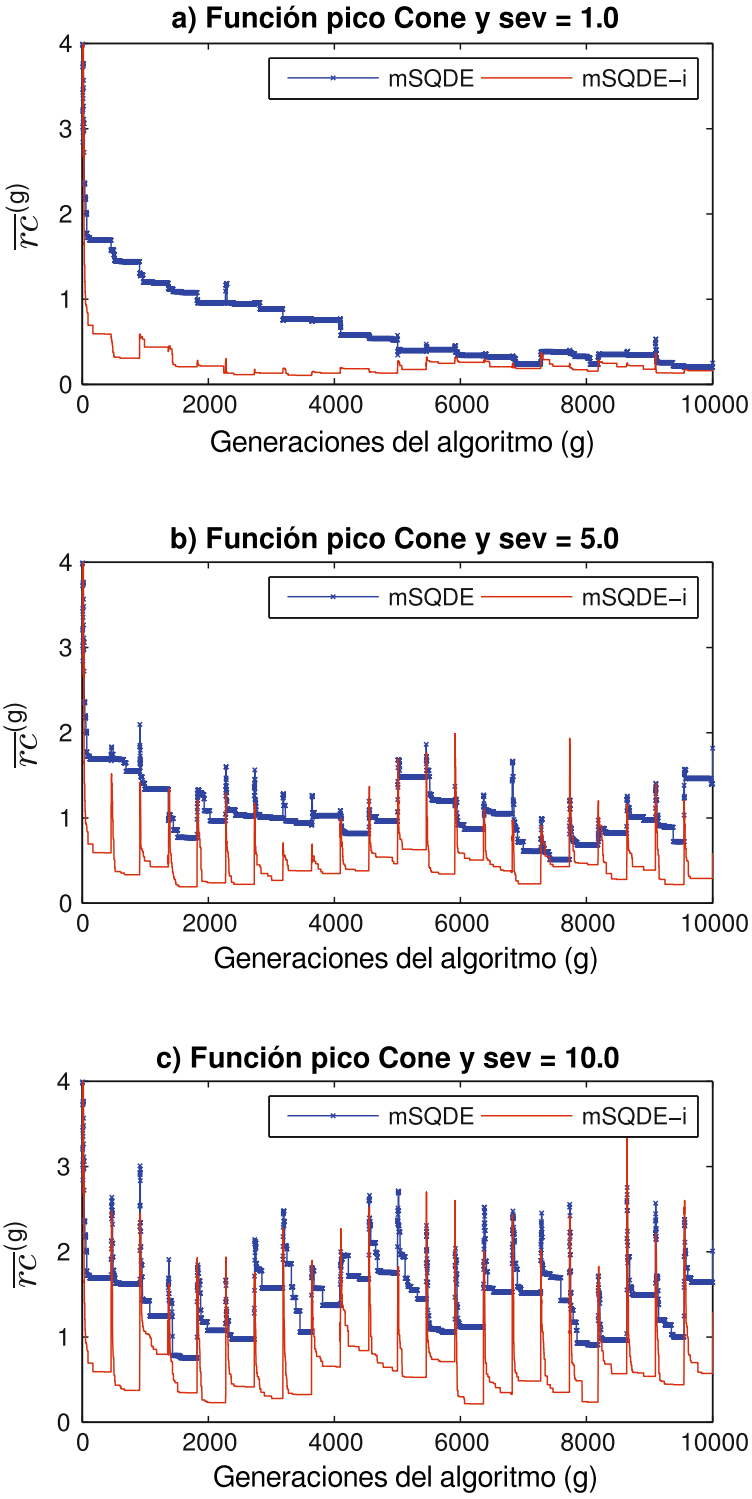

Figura 5.6 Evolución del parámetro $\overline{rc}^{(g)}$ (media de 20 ejecuciones) para los algoritmos mSQDE y mSQDE-i en instancias diferentes del MPB con: un solo pico (función *Cone*), $\Delta e = 5000$ y 20 cambios.

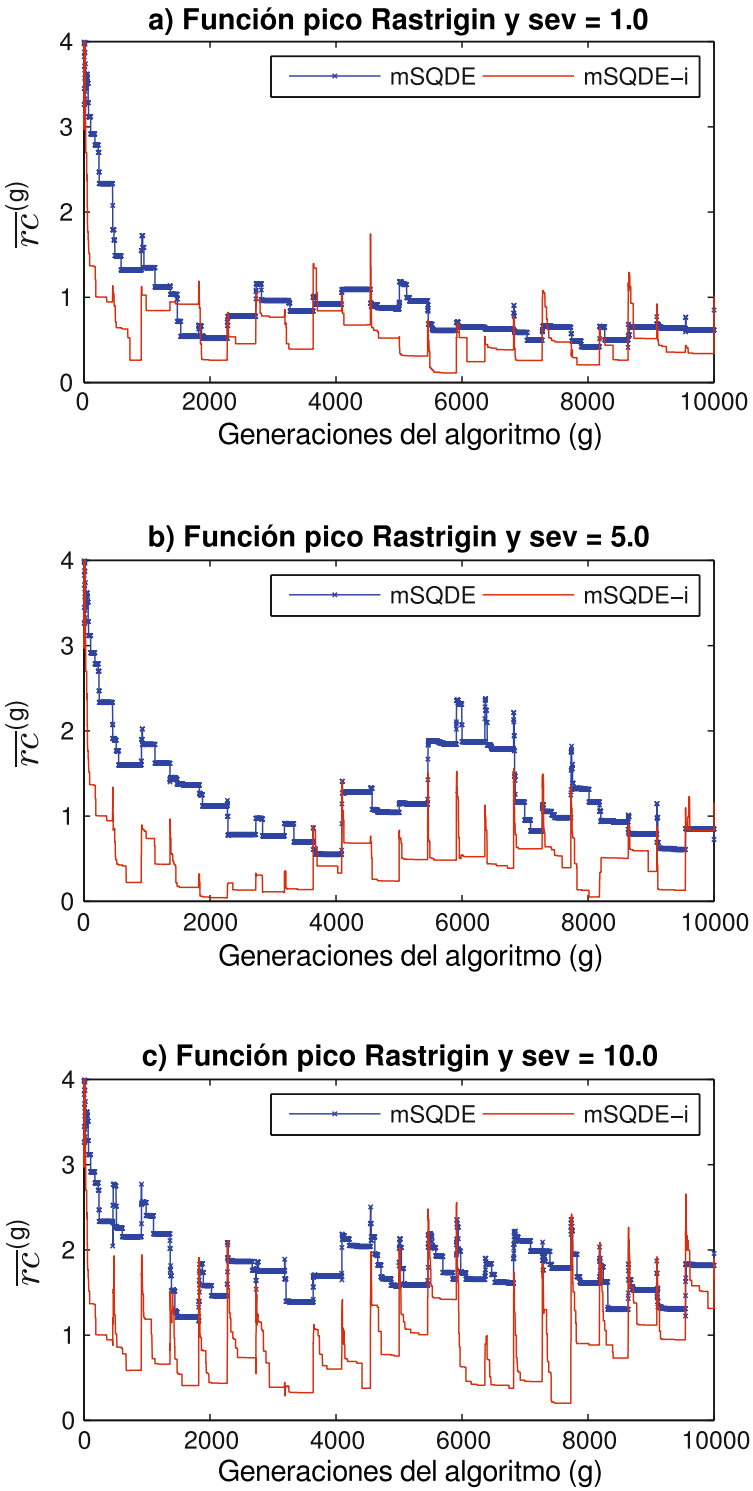

Figura 5.7 Evolución del parámetro $\overline{rc}^{(g)}$ (media de 20 ejecuciones) para los algoritmos mSQDE y mSQDE-i en instancias diferentes del MPB con: un solo pico (función *Rastrigin*), $\Delta e = 5000$ y 20 cambios.

también es capaz de responder a los cambios del problema. Esto último puede verse mejor si se tiene en cuenta que los picos en las líneas de las gráficas corresponden a los cambios del problema. En ese sentido, note como aumenta su magnitud este parámetro después de cada cambio, contribuyendo así a la exploración y por tanto al seguimiento del nuevo óptimo desplazado. Posteriormente, en las generaciones consecutivas el parámetro decrece su valor de manera progresiva, lo cual indica una contribución a la fase de explotación del algoritmo. Esto último se traduce en una generación de individuos muy cercana a la mejor solución del algoritmo.

Otra observación, a nuestro juicio importante, es el rápido decrecimiento que experimenta el parámetro $\overline{rc}^{(g)}$ para el caso del algoritmo mSQDE-i en relación a mSQDE. Esta acelerada adaptación pudiera ser la explicación de la superioridad de mSQDE-i sobre mSQDE en estos escenarios, como se pudo ver en la sección anterior.

Comparación con otros algoritmos

El objetivo principal de los experimentos desarrollados en esta sección es determinar cuan competitivos son los algoritmos propuestos en relación a otros referentes de la literatura.

En primer lugar, se compararon los algoritmos con otros basados en múltiples poblaciones. En ese sentido, se han seleccionado los algoritmos mQSO (Blackwell y Branke, 2006) y mPSODE [1] (Novoa-Hernández et al, 2011), los cuales han reportado muy buenos resultados en diferentes instancias del generador Movimiento de Picos. El primer método es el popular Quantum PSO multi-enjambre propuesto por Blackwell y Branke, mientras que el segundo es una versión mejorada de mQSO, particularmente adecuada para problemas multimodales, estudiada en el Capítulo 4. Recuérdese que la principal característica de mPSODE es el empleo de una regla de control difusa para detener enjambres con un alto grado de convergencia y una baja calidad. Como elemento de control, se consideró incluir además la versión básica de nuestros algoritmos, mQDE, la cual no incluye ni la estrategia de diversidad, ni el esquema de interacción entre individuos.

Adicionalmente, hemos considerado la extensión multipoblacional del algoritmo jDE propuesta por (Brest et al, 2009). Este algoritmo resulta en extremo interesante si se tiene en cuenta que fue el ganador de la competición del CEC 2009, y emplea auto-adaptación para los parámetros CR y F. Específicamente, jDE es comparado usando la metodología de dicha competición la cual emplea al generador GDBG (Li y Yang, 2008b). En este sentido, dado que no se posee una implementación de este algoritmo en la comparación emplearemos

[1] Aunque el nombre del algoritmo sugiere la presencia del paradigma Evolución diferencial, el sufijo "DE" de su nombre es empleado aquí para denotar la presencia de una estrategia de diversidad y de una regla de control difusa (véase Capítulo 4).

los resultados reportados por sus autores en el trabajo original. En cuanto a los algoritmos mSQDE y mSQDE-i, estos emplearán las mejores configuraciones de los experimentos anteriores, esto es, $\lambda = 0.25$ y $\tau = 0.5$.

Resultados en el generador MPB. En el caso de los experimentos que incluyen al generador MPB, las instancias seleccionadas están basadas en las funciones definidas por la Tabla 5.4, y los niveles siguientes para la severidad de desplazamiento y la frecuencia de cambio: $sev = \{1.0, 5.0, 10.0\}$ y $\Delta e = \{1000, 5000, 10000\}$. Los resultados, en términos del error de la mejor solución antes del cambio, se muestran en las Tablas 5.14, 5.15, 5.16, y 5.17. También se han incluido en la última columna de estas tablas la tasa de mejora del mejor de nuestros algoritmos frente al mejor entre mQSO y mPSODE. En caso de no obtenerse mejoras por parte de nuestros algoritmos, la tasa mostrada en la última columna corresponde al mejor de los algoritmos de la literatura (mQSO, mPSODE) frente al mejor de los nuestros. Las expresiones para ambos casos son idénticas a las empleadas en la Sección 4.3.3 del Capítulo 4.

A partir de estas comparaciones, el resultado más importante es que las variantes auto-adaptativas de nuestros algoritmos son considerablemente superiores a la variante básica mQDE. Además, es posible ver que en general, para problemas con picos unimodales, el algoritmo mSQDE-i tiene un comportamiento similar a mQSO, y claramente superior que mPSODE. En particular, el esquema de interacción a nivel de individuos, resulta en sentido general, mejor que mSQDE. Unas conclusiones algo diferentes pueden obtenerse cuando los algoritmos son aplicados a problemas con funciones pico multimodales. En estos casos, el algoritmo mSQDE posee una clara superioridad en general, a excepción de las instancias derivadas de la función Rastrigin, donde mSQDE-i es el mejor. Estas observaciones pueden ser fácilmente comprobadas si se observa la columna correspondiente a la tasa de mejoras. Además, el algoritmo mQSO es el peor de los cinco, incluso si se compara con mQDE en estas instancias multimodales. Una posible explicación de este pobre comportamiento puede ser la necesidad de una mayor diversidad a nivel de población con el objetivo de explorar convenientemente estos escenarios complejos. Un requerimiento que mQSO parece incapaz de cumplir debido al modelo del paradigma PSO que se emplea, el cual está basado en la mejor solución del enjambre (véase la Sección 2.2.4).

Con el objetivo de justificar formalmente las conclusiones anteriores, se desarrollaron varias pruebas estadísticos no paramétricos. En este sentido, fueron consideradas todas las medias reportadas en las Tablas 5.14–5.17, dividiéndose el análisis en tres grupos: 1) problemas unimodales, 2) problemas multimodales, y 3) todos los problemas.

Como establece la metodología de la Sección 3.2, primero se aplicaron las pruebas de Friedman e Iman-Davenport (Tabla 5.5). En ese sentido se pudo observar que en todos los

Tabla 5.5 Resultados de las pruebas de Friedman e Iman-Davenport ($\alpha = 0.05$) en los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO, en instancias de problema del generador MPB.

	Funciones pico		
Prueba	Unimodales	Multimodales	Todas
Friedman	**6.82E−11**	**7.14E−11**	**1.96E−09**
Iman-Davenport	**6.26E−35**	**2.73E−14**	**3.24E−10**

Los valores en negrita indican que existen diferencias significativas a nivel de grupo (p-valor$<$0.05).

Tabla 5.6 Rangos medios de los algoritmos (según prueba de Friedman) de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO, en instancias de problema del generador MPB.

	Funciones pico		
Algoritmos	Unimodales	Multimodales	Todas
mQDE	4.722	3.111	3.917
mSQDE	3.625	1.472	2.549
mSQDE-i	1.625	2.958	2.292
mPSODE	3.292	3.236	3.264
mQSO	1.736	4.222	2.979

casos existen diferencias significativas a nivel de grupo (p-valor$<$ 0.05), obteniéndose los rangos medios según Friedman que se muestran en la Tabla 5.6. Note que las posiciones alcanzadas por los algoritmos propuestos (mQDE, mSQDE, y mSQDE-i) son consistentes con los resultados descritos con anterioridad. Dada las diferencias detectadas a nivel de grupo se procedió a realizar la prueba post-hoc de Holm para detectar diferencias entre cada par de algoritmos. Los resultados de estas comparaciones múltiples se muestran en la Tabla 5.7, para cada tipo de función.

A modo de resumen de las pruebas de Friedman y de Holm, la Figura 5.8 muestra los rangos medios de los algoritmos y sus relaciones entre ellos según los p-valores ajustados según Holm.

De manera que se puede concluir que la aplicación de la estrategia auto-adaptativa (en conjunto con el esquema de interacción propuesto), permiten obtener un algoritmo más robusto al menos para las instancias de problemas y los algoritmos considerados.

Resultados en el generador GDBG. No obstante el análisis anterior, sería más justo si se pudiera comparar estos algoritmos auto-adaptativos con otros que presenten este tipo de control de parámetros. En ese sentido, la extensión jDE (Brest et al, 2009) es un buen candidato debido a tres razones fundamentales: 1) está basado en el paradigma DE, 2) posee

Tabla 5.7 Múltiples comparaciones según la prueba de Holm (α =0.05) de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO, en instancias de problema del generador MPB.

Funciones pico	Algoritmos	mSQDE	mSQDE-i	mPSODE	mQSO
Unimodales	mQDE	**9.72E−03**	**9.51E−16**	**4.95E−04**	**1.01E−14**
	mSQDE		**6.42E−07**	7.42E−01	**2.81E−06**
	mSQDE-i			**4.65E−05**	7.66E−01
	mPSODE				**1.50E−04**
Multimodales	mQDE	**8.76E−05**	1.37E+00	1.37E+00	**1.43E−02**
	mSQDE		**4.67E−04**	**1.99E−05**	**1.59E−12**
	mSQDE-i			1.37E+00	**4.17E−03**
	mPSODE				**3.25E−02**
Todas	mQDE	**1.88E−06**	**6.98E−09**	5.30E−02	**2.62E−03**
	mSQDE		5.60E−01	**3.98E−02**	3.07E−01
	mSQDE-i			**1.80E−03**	**4.54E−02**
	mPSODE				5.60E−01

Los valores corresponden a los *p*-valores ajustados. Los valores en negrita indican que existen diferencias significativas entre los algoritmos (*p*-valor<0.05).

excelentes resultados en ambientes dinámicos, y 3) posee auto-adaptación en elementos diferentes del algoritmo (ej. en los parámetro F y CR). Para desarrollar este estudio comparativo, también hemos empleado la metodología propuesta por la competición *CEC 2009 Evolutionary Computation in Dynamic y Uncertain Environments* (Li et al, 2008).

Los resultados en términos de la media del error antes del cambio, obtenidos a partir de 20 ejecuciones y empleando los escenarios del generador GDBG, se muestran en las Tablas 5.18 y 5.19.

A partir de estos resultados se puede notar la clara superioridad de jDE en los problemas con función *F3*, *F4* y *F5*, así como también la de mSQDE-i y mQSO en *F2*, *F3* y *F4*. Además, note que en esta ocasión, a diferencia de los resultados previos a este análisis, la ventaja de mSQDE-i en relación a mSQDE resulta clara, incluso en problemas con funciones multimodales (ej. *F3-F6*). En ese sentido, es importante destacar que las instancias del generador GDBG son considerablemente diferentes que las del generador Movimiento de Picos, siendo la principal diferencia la presencia de diferentes tipos de cambios. Además, todas las funciones empleadas por las instancias de este generador son rotadas en cada cambio del problema y el espacio de búsqueda posee 10 dimensiones (el doble con respecto al MPB). De manera que se puede inferir que el esquema de interacción entre individuos resulta más beneficioso en estos escenarios complejos.

Desde el punto de vista estadístico, los resultados de las pruebas Friedman e Iman-

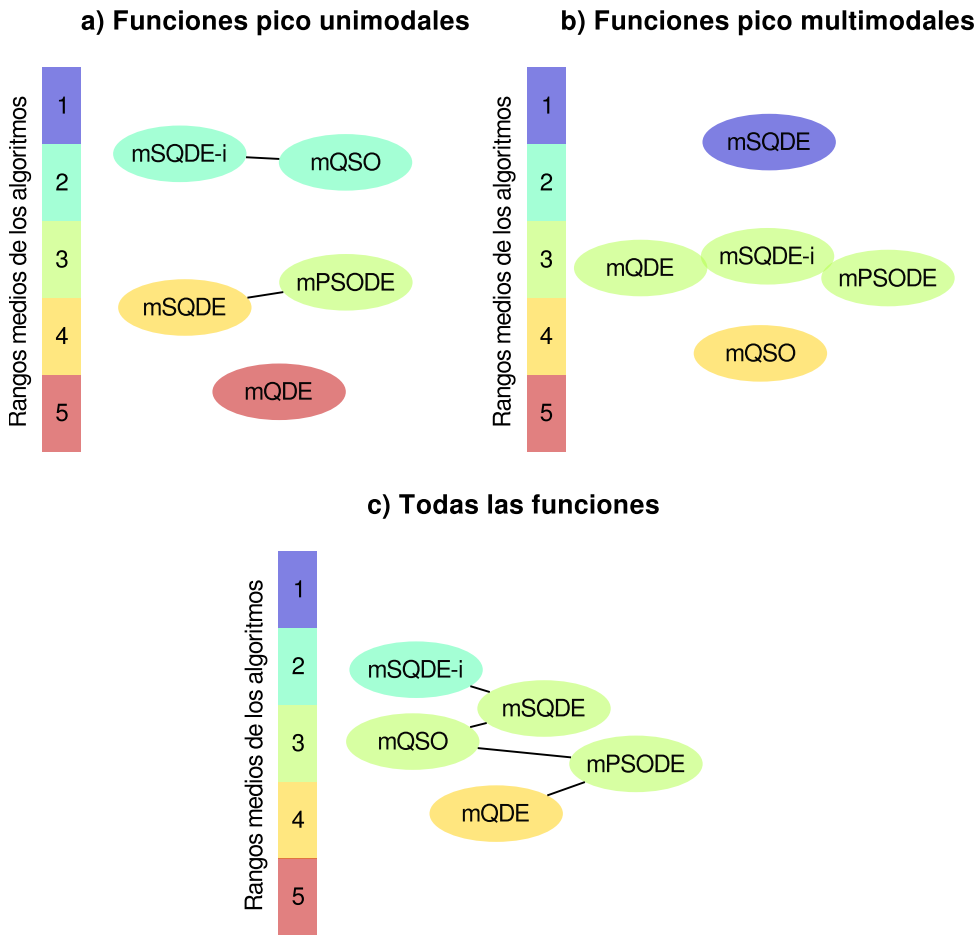

Figura 5.8 Posiciones relativas de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, mQSO, de acuerdo a sus rangos medios a partir dla prueba de Friedman en instancias del generador MPB. Los arcos directos indican, según la prueba de Holm, que no existen diferencias entre los algoritmos involucrados.

Davenport, mostrados en la Tabla 5.8, indican que existen diferencias en los tres grupos considerados: problemas con funciones unimodales, multimodales, y todas las instancias. En ese sentido, los rangos medios de los algoritmos se muestran en la Tabla 5.9. Note por ejemplo que mPSODE no es capaz de mantener su rendimiento en estas instancias del generador GDBG, de hecho este algoritmo alcanza el peor rango medio en todos los casos. El resto de los algoritmos alternan sus posiciones, por lo que en el caso en que se consideran todas las instancias, estos alcanzan rangos medios similares, excepto mQSO. Para descubrir en que escenarios estos rangos son significativamente diferentes entre sí, la Tabla 5.10 contiene los p-valores ajustados de la prueba post-hoc de Holm, correspondientes a las comparaciones múltiples de todos los pares de algoritmos. De manera resumida y gráfica, la Figura 5.9 muestra estos resultados.

Por otro lado, la metodología de la competición CEC 2009 lleva a cabo esta comparación mediante una marca que es calculada para cada instancia de problema y algoritmo,

Tabla 5.8 Resultados de las pruebas de Friedman e Iman-Davenport ($\alpha = 0.05$) en los algoritmos mPSODE, mQSO, jDE, mSQDE y mSQDE-i, mPSODE, y mQSO, en instancias de problema del generador GDBG.

	Funciones pico		
Prueba	Unimodales	Multimodales	Todas
Friedman	**3.29E−09**	**3.97E−11**	**6.16E−11**
Iman-Davenport	**6.96E−13**	**5.47E−25**	**1.00E−21**

Los valores en negrita indican que existen diferencias significativas a nivel de grupo (p-valor$<$0.05).

Tabla 5.9 Rangos medios de los algoritmos a partir dla prueba de Friedman de los algoritmos mPSODE, mQSO, jDE, mSQDE y mSQDE-i, mPSODE, y mQSO, en instancias de problema del generador GDBG.

	Funciones pico		
Algoritmos	Unimodales	Multimodales	Todas
mPSODE	4.857	4.499	4.653
mQSO	2.095	4.249	3.327
jDE	3.286	1.607	2.327
mSQDE	2.762	2.286	2.490
mSQDE-i	1.999	2.357	2.204

de manera que la suma de estas determina el rendimiento global del algoritmo en cuestión. Formalmente, el procedimiento para calcular la marca de la instancia de problema k es el siguiente:

$$mark_k = perc_k \cdot \frac{1}{\mathsf{K} \cdot \mathsf{RUNS}} \cdot \sum_{i=1}^{\mathsf{RUNS}} \sum_{j=1}^{\mathsf{K}} frel_{ij} \tag{5.8}$$

donde $perc_k$ es el porcentaje o peso (en términos de importancia) de la instancia k. Los valores específicos de dichos pesos son mostrados la Tabla 5.11). Por su parte, K y RUNS son el número de cambios y de ejecuciones, respectivamente. En particular, $\mathsf{K} = 60$ y $\mathsf{RUNS} = 20$. Además, $frel_{ij}$ es el fitness relativo del algoritmo antes del cambio j perteneciente a la ejecución i. Este es calculado para problemas de minimización como $frel = f_{optim}/\hat{f}$, mientras que para problemas de maximización como $frel = \hat{f}/f_{optim}$. Aquí, f_{optim} y $\hat{f}$ son los fitness del óptimo del problema y de la mejor solución del algoritmo, respectivamente. Finalmente, el rendimiento global es obtenido a partir de la suma de todas las marcas y multiplicado por 100 (véase la última fila de la Tabla 5.11).

Las Tablas 5.12 y 5.13 muestra las marcas obtenidas por los algoritmos. El rendimiento global se encuentra en negrita en el final de los resultados de cada algoritmo. A partir de

Tabla 5.10 Múltiples comparaciones según la prueba de Holm (α =0.05) de los algoritmos mPSODE, mQSO, jDE, mSQDE y mSQDE-i, mPSODE, y mQSO, en instancias de problema del generador GDBG.

Funciones pico	Algoritmos	mQSO	jDE	mSQDE	mSQDE-i
Unimodales	mPSODE	**1.36E−07**	**8.95E−03**	**1.40E−04**	**4.76E−08**
	mQSO		7.35E−02	5.16E−01	8.45E−01
	jDE			5.66E−01	5.04E−02
	mSQDE				4.74E−01
Multimodales	mPSODE	1.11E+00	**7.61E−11**	**1.28E−06**	**2.77E−06**
	mQSO		**3.60E−09**	**2.01E−05**	**3.74E−05**
	jDE			3.25E−01	3.04E−01
	mSQDE				1.11E+00
Todas	mPSODE	**2.30E−04**	**2.93E−12**	**1.02E−10**	**1.77E−13**
	mQSO		**8.73E−03**	**3.52E−02**	**2.65E−03**
	jDE			1.22E+00	1.22E+00
	mSQDE				1.11E+00

Los valores corresponden a los p-valores ajustados. Los valores en negrita indican que existen diferencias significativas entre los algoritmos (p-valor$<$0.05).

Tabla 5.11 Pesos asociados a cada instancia de problema del generador GDBG ($perc_k$) empleados en el cálculo de las marcas de cada instancia.

	F1	F1(50)	F2	F3	F4	F5	F6
T1	0.015	0.015	0.024	0.024	0.024	0.024	0.024
T2	0.015	0.015	0.024	0.024	0.024	0.024	0.024
T3	0.015	0.015	0.024	0.024	0.024	0.024	0.024
T4	0.015	0.015	0.024	0.024	0.024	0.024	0.024
T5	0.015	0.015	0.024	0.024	0.024	0.024	0.024
T6	0.015	0.015	0.024	0.024	0.024	0.024	0.024
T7	0.01	0.01	0.016	0.016	0.016	0.016	0.016
Marca							
Rendimiento (suma de la marca obtenida para cada caso y multiplicada por 100):							

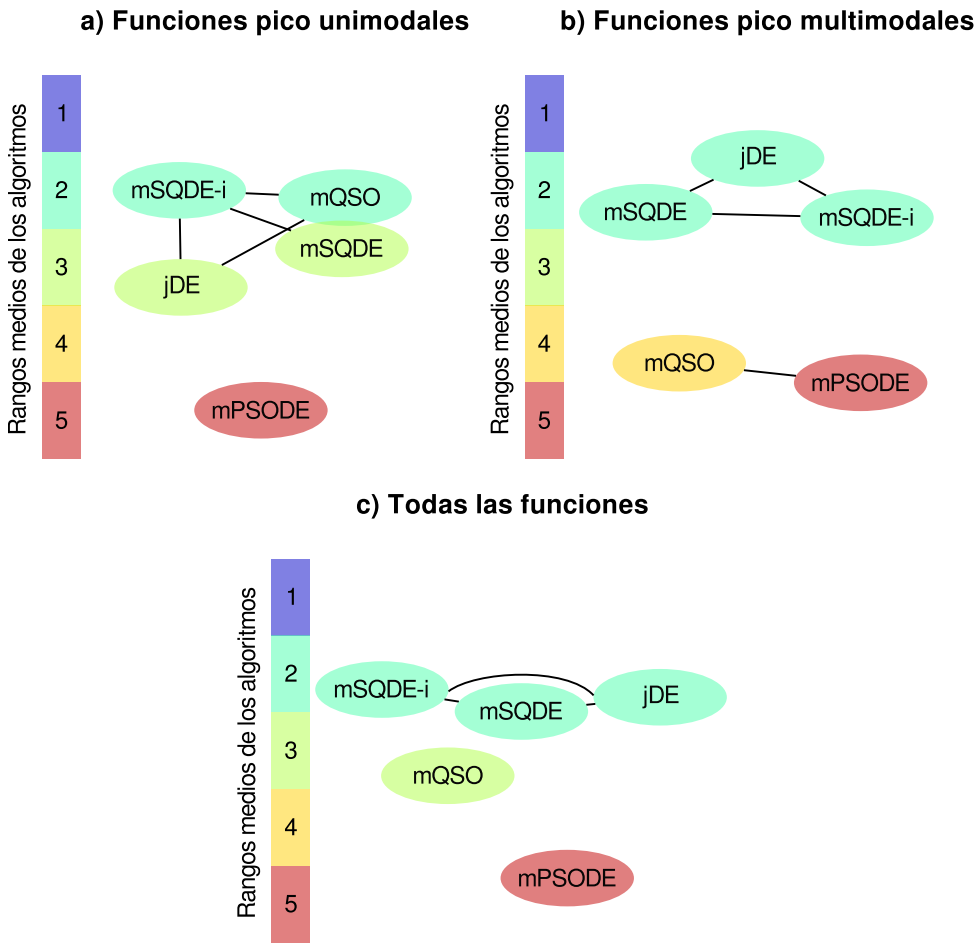

Figura 5.9 Posiciones relativas de los algoritmos mPSODE, mQSO, jDE, mSQDE y mSQDE-i, de acuerdo a sus rangos medios a partir dla prueba de Friedman en instancias del generador GDBG. Los arcos directos indican, según la prueba de Holm, que no existen diferencias entre los algoritmos involucrados.

estos rendimientos se puede apreciar la superioridad de nuestros algoritmos en relación a jDE, un hecho que demuestra los beneficios de la estrategia auto-adaptativa propuesta.

5.3 Análisis del rol de la auto-adaptación en ambientes dinámicos

La principal conclusión de los experimentos anteriores es que la auto-adaptación tiene sentido en ambientes dinámicos, fundamentalmente cuando se aplica a un enfoque de adaptación dinámica. Sin embargo, aún quedan por analizar algunas cuestiones relacionadas con este campo de investigación, tales como:

1. ¿Qué efecto tendrá la auto-adaptación cuando es aplicada en diferentes partes del

Tabla 5.12 Rendimiento de los algoritmos mPSODE, mQSO (Blackwell y Branke, 2006) y jDE (Brest et al, 2009) según la metodología de la competición CEC'2009.

Alg.	Func./Chg.	F1	F1(50)	F2	F3	F4	F5	F6
mPSODE	T1	0.01491260	0.01451690	0.02298060	0.00071190	0.02029910	0.02014030	0.01727420
	T2	0.01434830	0.01425390	0.01306690	0.00045207	0.00651110	0.00282486	0.00826216
	T3	0.01466250	0.01428830	0.01297790	0.00052273	0.00834544	0.00315346	0.00852748
	T4	0.01482020	0.01432930	0.02068210	0.00049488	0.01557920	0.00837014	0.01133240
	T5	0.01431160	0.01471520	0.01154080	0.00097498	0.00749892	0.00233849	0.01081010
	T6	0.01474530	0.01472340	0.01988930	0.00027066	0.01651660	0.00180491	0.00977543
	T7	0.00935847	0.00936911	0.01121500	0.00131227	0.00986206	0.01407180	0.00904069
	Marca	0.09715897	0.09619611	0.11235260	0.00473949	0.08461242	0.05270396	0.07502246
	Rendimiento (suma de la marca obtenida para cada caso y multiplicada por 100):							**52.2786**
mQSO	T1	0.01498900	0.01452610	0.02320500	0.00061238	0.02201620	0.01988480	0.01717200
	T2	0.01478470	0.01463940	0.02102020	0.00044066	0.01790500	0.00947049	0.00841073
	T3	0.01479610	0.01464660	0.02099880	0.00044565	0.01733200	0.00972390	0.00817718
	T4	0.01499100	0.01433810	0.02369400	0.00031119	0.02163780	0.01252920	0.01258090
	T5	0.01489400	0.01490630	0.02088130	0.00090169	0.01662590	0.01276160	0.00999263
	T6	0.01498260	0.01482570	0.02345090	0.00030550	0.02114390	0.01304480	0.01206070
	T7	0.00985016	0.00979820	0.01520070	0.00080806	0.01329900	0.01382350	0.00945046
	Marca	0.09928756	0.09768040	0.14845090	0.00382512	0.12995980	0.09123829	0.07784460
	Rendimiento (suma de la marca obtenida para cada caso y multiplicada por 100):							**64.8287**
jDE	T1	0.01476800	0.01468760	0.02110490	0.01571070	0.02066150	0.02176600	0.01704720
	T2	0.01369010	0.01359260	0.01352710	0.00298238	0.01314800	0.02086610	0.01394880
	T3	0.01382560	0.01353040	0.01308080	0.00281439	0.01354500	0.02092860	0.01419120
	T4	0.01471640	0.01469410	0.02100350	0.01276210	0.01992680	0.02219620	0.01530460
	T5	0.01394150	0.01436440	0.01239760	0.00440560	0.01237600	0.02130940	0.01551840
	T6	0.01412650	0.01387400	0.01777600	0.00734523	0.01795010	0.02073610	0.01395120
	T7	0.00911221	0.00898569	0.01018760	0.00549392	0.01018130	0.01378940	0.00942562
	Marca	0.09418030	0.09372880	0.10907800	0.05151430	0.10778900	0.14159200	0.09938700
	Rendimiento (suma de la marca obtenida para cada caso y multiplicada por 100):							**69.7269**

Tabla 5.13 Rendimiento de los algoritmos mSQDE y mSQDE-i según la metodología de la competición CEC'2009.

Alg.	Func./Chg.	F1	F1(50)	F2	F3	F4	F5	F6
mSQDE	T1	0.01496780	0.01484390	0.02163770	0.02053870	0.02160840	0.02015330	0.01959680
	T2	0.01478170	0.01465040	0.01784190	0.00232768	0.01776030	0.01828760	0.01552850
	T3	0.01479070	0.01465360	0.01796550	0.00251363	0.01760010	0.01851620	0.01526990
	T4	0.01495120	0.01461950	0.02106570	0.00974013	0.02083400	0.01922880	0.01653200
	T5	0.01486370	0.01487830	0.01861540	0.00297154	0.01751510	0.01930390	0.01771260
	T6	0.01493550	0.01482670	0.01991480	0.00591751	0.01972110	0.01840280	0.01558510
	T7	0.00985849	0.00979964	0.01330270	0.00407253	0.01312610	0.01236000	0.01046680
	Marca	0.09914909	0.09827204	0.13034370	0.04808172	0.12816510	0.12625260	0.11069170
	Rendimiento (suma de la marca obtenida para cada caso y multiplicada por 100):							**74.0956**
mSQDE-i	T1	0.01498020	0.01456430	0.02286420	0.02058060	0.02305200	0.02252120	0.02117450
	T2	0.01482540	0.01464680	0.02070090	0.00044817	0.01987380	0.02111330	0.01584270
	T3	0.01485060	0.01469000	0.02060400	0.00047322	0.01984330	0.02126110	0.01545560
	T4	0.01498310	0.01449150	0.02296270	0.00775828	0.02292200	0.02190330	0.01740910
	T5	0.01490380	0.01490140	0.02082890	0.00100388	0.01966950	0.01926540	0.01644270
	T6	0.01497820	0.01480940	0.02247230	0.00253433	0.02222430	0.01975350	0.01632690
	T7	0.00988072	0.00979782	0.01477050	0.00073419	0.01436770	0.01424640	0.01075060
	Marca	0.09940202	0.09790122	0.14520350	0.03353268	0.14195260	0.14006420	0.11340210
	Rendimiento (suma de la marca obtenida para cada caso y multiplicada por 100):							**77.1458**

Tabla 5.14 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones ± error estándar) de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO (Blackwell y Branke, 2006), en instancias del generador MPB con funciones pico *Cone* y *Sphere*.

Función	Δe	sev.	mQDE	mSQDE	mSQDE-i	mPSODE	mQSO	Tasa de mejora
Cone	1000	1.0	4.50 ± 0.18	3.51 ± 0.12	**2.49 ± 0.10**	2.84 ± 0.09	2.88 ± 0.13	12.32%
		5.0	6.32 ± 0.25	7.01 ± 0.25	5.71 ± 0.21	6.74 ± 0.23	**4.23 ± 0.16**	–
		10.0	11.95 ± 0.50	9.87 ± 0.42	8.22 ± 0.27	10.97 ± 0.41	**6.16 ± 0.19**	–
	5000	1.0	2.05 ± 0.11	1.15 ± 0.10	**0.51 ± 0.06**	0.68 ± 0.10	0.68 ± 0.09	25.00%
		5.0	2.31 ± 0.12	2.01 ± 0.12	**0.91 ± 0.09**	1.45 ± 0.13	0.97 ± 0.09	6.19%
		10.0	3.27 ± 0.16	2.90 ± 0.12	**1.51 ± 0.09**	2.65 ± 0.13	1.67 ± 0.12	9.58%
	10000	1.0	1.30 ± 0.09	0.68 ± 0.07	0.45 ± 0.10	0.45 ± 0.09	**0.41 ± 0.08**	–
		5.0	1.71 ± 0.12	1.36 ± 0.11	**0.52 ± 0.07**	1.10 ± 0.09	0.52 ± 0.09	–
		10.0	2.40 ± 0.13	1.99 ± 0.13	0.98 ± 0.09	2.25 ± 0.15	**0.95 ± 0.09**	–
Sphere	1000	1.0	6.05 ± 0.65	2.47 ± 0.11	**1.28 ± 0.18**	2.43 ± 0.19	1.71 ± 0.12	25.15%
		5.0	8.14 ± 0.63	7.48 ± 0.26	5.17 ± 0.24	7.41 ± 0.27	**2.83 ± 0.12**	–
		10.0	19.15 ± 0.95	13.28 ± 0.43	9.58 ± 0.35	15.96 ± 0.49	**5.01 ± 0.19**	–
	5000	1.0	0.68 ± 0.08	0.30 ± 0.05	**0.17 ± 0.04**	0.21 ± 0.06	0.20 ± 0.07	15.00%
		5.0	0.93 ± 0.08	0.88 ± 0.08	**0.19 ± 0.03**	0.85 ± 0.08	0.19 ± 0.04	–
		10.0	1.34 ± 0.09	1.26 ± 0.07	0.41 ± 0.07	1.16 ± 0.07	**0.33 ± 0.04**	–
	10000	1.0	0.33 ± 0.05	0.18 ± 0.04	**0.15 ± 0.04**	0.20 ± 0.06	0.17 ± 0.06	11.76%
		5.0	0.63 ± 0.07	0.47 ± 0.05	0.12 ± 0.03	0.83 ± 0.09	**0.11 ± 0.03**	–
		10.0	0.67 ± 0.07	0.58 ± 0.06	**0.18 ± 0.03**	1.11 ± 0.07	0.20 ± 0.04	10.00%

Los valores en negrita corresponden al mejor algoritmo.

Tabla 5.15 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones ± error estándar) de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO (Blackwell y Branke, 2006), en instancias del generador MPB con funciones pico *Schwefel* y *Quadric*.

Función	Δe	sev.	mQDE	mSQDE	mSQDE-i	mPSODE	mQSO	Tasa de mejora
SchwefelP222	1000	1.0	6.13 ± 0.33	5.43 ± 0.21	5.31 ± 0.35	**4.33 ± 0.24**	4.78 ± 0.21	–
		5.0	12.34 ± 0.53	12.24 ± 0.48	11.69 ± 0.49	12.69 ± 0.55	**9.20 ± 0.43**	–
		10.0	41.16 ± 1.67	**24.05 ± 0.88**	26.88 ± 0.87	36.35 ± 1.46	63.20 ± 5.16	33.84%
	5000	1.0	1.64 ± 0.08	0.82 ± 0.06	0.46 ± 0.04	**0.36 ± 0.06**	0.77 ± 0.07	–
		5.0	2.28 ± 0.09	1.93 ± 0.09	**1.05 ± 0.06**	1.78 ± 0.09	1.17 ± 0.08	10.26%
		10.0	3.34 ± 0.12	2.83 ± 0.10	**1.68 ± 0.09**	5.08 ± 0.19	3.11 ± 0.11	45.98%
	10000	1.0	0.83 ± 0.05	0.44 ± 0.05	0.28 ± 0.06	**0.23 ± 0.05**	0.34 ± 0.06	–
		5.0	1.02 ± 0.05	0.69 ± 0.03	0.41 ± 0.04	0.95 ± 0.06	**0.28 ± 0.04**	–
		10.0	1.32 ± 0.06	1.08 ± 0.06	**0.61 ± 0.05**	1.97 ± 0.12	0.85 ± 0.08	28.24%
Quadric	1000	1.0	11.43 ± 0.90	5.39 ± 0.24	3.99 ± 0.37	4.53 ± 0.30	**3.31 ± 0.30**	–
		5.0	23.03 ± 1.49	16.50 ± 0.64	11.92 ± 0.46	13.60 ± 0.48	**7.71 ± 0.35**	–
		10.0	70.21 ± 3.53	33.08 ± 0.97	23.27 ± 0.58	31.69 ± 1.16	**17.47 ± 0.59**	–
	5000	1.0	1.38 ± 0.10	0.56 ± 0.06	**0.25 ± 0.05**	0.51 ± 0.06	0.29 ± 0.04	13.79%
		5.0	1.83 ± 0.12	1.85 ± 0.07	0.62 ± 0.04	1.23 ± 0.06	**0.48 ± 0.06**	–
		10.0	2.60 ± 0.16	3.00 ± 0.10	1.18 ± 0.07	2.44 ± 0.12	**0.83 ± 0.06**	–
	10000	1.0	0.74 ± 0.04	0.37 ± 0.07	**0.10 ± 0.02**	0.37 ± 0.07	0.16 ± 0.05	37.50%
		5.0	1.26 ± 0.10	0.81 ± 0.06	0.18 ± 0.03	1.03 ± 0.07	**0.17 ± 0.03**	–
		10.0	1.43 ± 0.10	1.14 ± 0.08	**0.44 ± 0.05**	1.41 ± 0.10	0.47 ± 0.07	6.38%

Los valores en negrita corresponden al mejor algoritmo.

Tabla 5.16 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones ± error estándar) de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO (Blackwell y Branke, 2006), en instancias del generador MPB con funciones pico *Rastrigin* y *Griewank*.

Función	Δe	sev.	mQDE	mSQDE	mSQDE-i	mPSODE	mQSO	Tasa de mejora
Rastrigin	1000	1.0	64.96 ± 2.73	54.00 ± 2.48	**41.73 ± 1.73**	48.82 ± 2.00	53.19 ± 2.36	14.52%
		5.0	99.30 ± 3.95	72.97 ± 3.34	**58.40 ± 2.50**	81.35 ± 3.66	99.66 ± 4.89	28.21%
		10.0	188.42 ± 10.78	98.48 ± 4.98	**81.13 ± 3.62**	104.93 ± 4.77	125.40 ± 5.80	22.68%
	5000	1.0	20.46 ± 0.84	16.30 ± 0.73	**10.42 ± 0.39**	16.85 ± 0.63	22.76 ± 0.97	38.16%
		5.0	28.60 ± 1.21	20.53 ± 0.83	**14.83 ± 0.60**	40.02 ± 1.29	57.75 ± 2.18	62.94%
		10.0	38.81 ± 1.68	26.79 ± 1.09	**16.48 ± 0.59**	47.51 ± 1.64	69.28 ± 2.78	65.31%
	10000	1.0	10.20 ± 0.42	7.20 ± 0.36	**5.24 ± 0.19**	14.56 ± 0.46	18.67 ± 0.70	64.01%
		5.0	14.47 ± 0.50	9.16 ± 0.34	**7.20 ± 0.18**	34.95 ± 1.17	47.99 ± 1.53	79.40%
		10.0	17.96 ± 0.66	12.21 ± 0.39	**7.24 ± 0.27**	35.31 ± 1.01	53.06 ± 1.76	79.50%
Griewank	1000	1.0	**1.85 ± 0.07**	1.99 ± 0.06	2.70 ± 0.12	2.37 ± 0.08	2.27 ± 0.09	18.50%
		5.0	1.82 ± 0.05	2.26 ± 0.07	2.82 ± 0.10	2.89 ± 0.09	**1.76 ± 0.05**	–
		10.0	**1.85 ± 0.06**	2.32 ± 0.07	2.97 ± 0.13	2.92 ± 0.09	1.92 ± 0.07	3.65%
	5000	1.0	1.27 ± 0.07	**0.82 ± 0.04**	2.38 ± 0.10	1.56 ± 0.10	2.19 ± 0.11	47.44%
		5.0	1.03 ± 0.04	**0.95 ± 0.04**	2.02 ± 0.07	1.22 ± 0.04	1.70 ± 0.08	22.13%
		10.0	1.04 ± 0.05	**0.94 ± 0.04**	2.24 ± 0.10	1.35 ± 0.05	2.08 ± 0.10	30.37%
	10000	1.0	1.13 ± 0.06	**0.58 ± 0.03**	2.22 ± 0.09	1.52 ± 0.09	2.09 ± 0.10	61.84%
		5.0	0.76 ± 0.03	**0.66 ± 0.03**	1.93 ± 0.07	1.02 ± 0.05	1.81 ± 0.10	35.29%
		10.0	0.87 ± 0.05	**0.66 ± 0.03**	2.14 ± 0.07	1.31 ± 0.10	2.15 ± 0.10	49.62%

Los valores en negrita corresponden al mejor algoritmo.

Tabla 5.17 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones ± error estándar) de los algoritmos mQDE, mSQDE, mSQDE-i, mPSODE, y mQSO (Blackwell y Branke, 2006), en instancias del generador MPB con funciones pico *Ackley* y *Alpine*.

Función	Δe	sev.	mQDE	mSQDE	mSQDE-i	mPSODE	mQSO	Tasa de mejora
Ackley	1000	1.0	18.29 ± 1.09	12.98 ± 0.46	14.25 ± 0.82	**12.47 ± 0.62**	17.27 ± 1.66	–
		5.0	28.17 ± 0.98	**19.97 ± 0.92**	23.41 ± 1.43	23.51 ± 1.04	32.67 ± 1.90	15.06%
		10.0	42.55 ± 1.81	**25.31 ± 1.07**	31.96 ± 1.91	31.96 ± 1.77	45.45 ± 2.13	20.81%
	5000	1.0	6.52 ± 0.42	**4.79 ± 0.31**	6.19 ± 0.43	8.65 ± 1.02	13.43 ± 1.46	44.62%
		5.0	10.47 ± 0.52	**8.37 ± 0.52**	9.58 ± 0.80	23.39 ± 1.28	27.63 ± 1.76	64.22%
		10.0	14.54 ± 0.84	**11.04 ± 0.69**	13.09 ± 0.62	40.67 ± 1.73	44.37 ± 2.41	72.85%
	10000	1.0	4.43 ± 0.24	**4.10 ± 0.38**	5.02 ± 0.34	8.97 ± 1.06	12.64 ± 1.50	54.29%
		5.0	6.77 ± 0.47	**5.57 ± 0.42**	7.22 ± 0.54	24.15 ± 1.62	23.51 ± 2.28	76.31%
		10.0	8.94 ± 0.50	**7.14 ± 0.46**	9.08 ± 0.68	40.57 ± 2.07	44.25 ± 2.53	82.40%
Alpine	1000	1.0	6.09 ± 0.20	**4.03 ± 0.20**	8.17 ± 0.41	4.31 ± 0.34	5.87 ± 0.29	6.50%
		5.0	11.56 ± 0.49	**9.74 ± 0.39**	12.90 ± 0.61	11.06 ± 0.46	15.13 ± 0.70	11.93%
		10.0	16.20 ± 0.77	**14.39 ± 0.70**	17.30 ± 0.77	16.23 ± 0.64	18.53 ± 0.81	11.34%
	5000	1.0	3.14 ± 0.17	**1.81 ± 0.16**	3.47 ± 0.23	2.45 ± 0.21	3.13 ± 0.30	26.12%
		5.0	5.18 ± 0.24	**2.84 ± 0.22**	4.14 ± 0.23	4.20 ± 0.29	5.58 ± 0.29	32.38%
		10.0	7.39 ± 0.36	**4.28 ± 0.21**	5.13 ± 0.26	5.12 ± 0.28	6.72 ± 0.34	16.41%
	10000	1.0	2.90 ± 0.23	**1.49 ± 0.14**	3.10 ± 0.22	2.33 ± 0.19	2.79 ± 0.26	36.05%
		5.0	3.59 ± 0.20	**1.97 ± 0.14**	3.12 ± 0.18	3.57 ± 0.24	4.01 ± 0.22	44.82%
		10.0	5.10 ± 0.30	**2.89 ± 0.16**	3.82 ± 0.24	4.69 ± 0.28	4.53 ± 0.21	36.20%

Los valores en negrita corresponden al mejor algoritmo.

Tabla 5.18 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones y desviación estándar) de los algoritmos mPSODE, mQSO (Blackwell y Branke, 2006), jDE (Brest et al, 2009), mSQDE y mSQDE-i, en instancias del generador GDBG con funciones F1, F1(50), y F2.

Función	T. Cambio	mPSODE	mQSO	jDE	mSQDE	mSQDE-i	Tasa de mejora
F1(10)	T1	0.53(2.29)	0.06(0.72)	**0.03(0.44)**	0.19(0.67)	0.11(0.78)	–
F1(10)	T2	3.98(7.72)	1.30(4.70)	3.59(7.84)	1.33(4.29)	**1.06(4.12)**	19.04%
F1(10)	T3	2.06(5.70)	1.24(4.61)	3.00(7.13)	1.29(4.10)	**0.91(3.52)**	26.98%
F1(10)	T4	1.19(3.01)	0.06(0.51)	**0.02(0.29)**	0.32(0.57)	0.11(0.51)	–
F1(10)	T5	4.10(10.22)	0.64(2.20)	2.18(4.39)	0.82(2.08)	**0.58(2.07)**	9.27%
F1(10)	T6	1.68(5.69)	**0.12(0.84)**	1.15(5.73)	0.43(0.77)	0.14(0.63)	–
F1(10)	T7	5.93(12.25)	1.40(4.88)	3.50(7.90)	1.32(4.37)	**1.11(4.03)**	20.42%
F1(50)	T1	3.10(4.44)	3.01(4.58)	**0.17(0.76)**	1.00(1.91)	2.78(4.21)	–
F1(50)	T2	4.89(7.45)	2.36(3.94)	4.09(6.45)	**2.29(3.69)**	2.31(3.90)	3.13%
F1(50)	T3	4.68(6.23)	2.31(4.05)	4.29(6.75)	2.27(3.91)	**2.03(3.65)**	12.29%
F1(50)	T4	4.47(8.13)	4.41(8.54)	**0.09(0.25)**	2.54(4.79)	3.39(6.13)	–
F1(50)	T5	1.76(3.04)	**0.58(1.15)**	0.95(1.77)	0.75(1.15)	0.61(1.28)	–
F1(50)	T6	1.84(3.73)	1.16(2.28)	1.77(5.83)	**1.15(1.67)**	1.27(2.23)	0.60%
F1(50)	T7	6.19(10.02)	1.98(3.85)	4.37(6.93)	**1.97(3.52)**	1.99(3.66)	0.64%
F2	T1	1.23(3.90)	1.12(4.49)	**0.96(3.08)**	2.52(4.83)	1.40(4.64)	–
F2	T2	110.86(186.93)	5.46(17.43)	43.00(114.94)	10.89(33.23)	**4.82(15.15)**	11.77%
F2	T3	105.55(178.58)	7.03(30.14)	50.19(124.02)	11.41(35.50)	**5.93(21.86)**	15.73%
F2	T4	3.15(6.45)	**0.22(1.24)**	0.79(2.53)	1.73(1.96)	0.59(1.36)	–
F2	T5	164.16(189.02)	14.29(60.48)	67.05(130.15)	23.94(61.31)	**10.15(30.10)**	28.95%
F2	T6	6.72(32.34)	**0.39(1.63)**	3.37(12.97)	2.68(3.00)	0.90(1.74)	–
F2	T7	17.84(42.87)	**1.61(5.59)**	13.25(45.78)	4.18(5.48)	1.95(4.64)	–

Los valores en negrita corresponden al mejor algoritmo.

Tabla 5.19 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones y desviación estándar) de los algoritmos mPSODE, mQSO (Blackwell y Branke, 2006), jDE (Brest et al, 2009), mSQDE y mSQDE-i, en instancias del generador GDBG con funciones F3–F6.

Función	T. Cambio	mPSODE	mQSO	jDE	mSQDE	mSQDE-i	Tasa de mejora
F3	T1	728.76(91.64)	766.67(77.19)	11.39(58.11)	**9.26(61.70)**	19.83(112.38)	18.72%
F3	T2	1001.26(95.18)	1022.89(75.37)	**558.50(384.62)**	685.14(338.22)	985.58(80.06)	–
F3	T3	995.38(121.90)	1025.41(70.59)	**572.11(386.09)**	675.06(349.26)	979.33(87.27)	–
F3	T4	915.88(283.46)	943.09(260.71)	**65.74(208.93)**	356.56(435.96)	600.92(535.98)	–
F3	T5	949.76(115.21)	996.15(77.75)	**475.77(379.89)**	697.23(305.09)	957.82(94.78)	–
F3	T6	1112.26(320.65)	1026.29(283.75)	**243.27(384.98)**	442.83(439.58)	859.77(471.92)	–
F3	T7	751.55(305.87)	855.41(241.00)	**153.67(286.38)**	349.98(366.10)	860.77(249.93)	–
F4	T1	5.39(9.62)	2.37(5.63)	1.49(4.48)	2.26(3.99)	**0.94(3.13)**	36.66%
F4	T2	339.18(267.57)	19.33(69.68)	49.50(135.25)	10.25(27.74)	**6.83(19.77)**	64.64%
F4	T3	276.10(279.64)	30.40(102.20)	51.94(141.78)	10.02(21.23)	**6.84(19.32)**	77.49%
F4	T4	21.20(58.23)	2.31(6.15)	1.51(4.10)	1.93(2.30)	**0.60(1.30)**	59.99%
F4	T5	307.72(230.61)	65.37(144.45)	69.44(144.04)	38.53(94.00)	**17.50(54.16)**	73.22%
F4	T6	30.44(104.73)	2.94(7.86)	2.35(5.78)	2.86(3.24)	**1.21(3.20)**	48.42%
F4	T7	60.26(145.29)	12.38(49.78)	11.74(39.45)	4.85(6.75)	**3.08(6.91)**	73.80%
F5	T1	5.41(9.61)	8.97(57.27)	**0.16(1.03)**	3.71(3.55)	1.32(2.56)	–
F5	T2	1543.99(741.72)	710.99(851.92)	**0.33(1.64)**	6.21(6.28)	2.97(5.30)	–
F5	T3	1471.02(761.85)	684.93(845.91)	**0.36(1.83)**	5.88(5.46)	2.76(4.53)	–
F5	T4	719.05(825.35)	336.57(657.31)	**0.11(0.83)**	3.12(2.76)	1.28(2.25)	–
F5	T5	1676.51(599.26)	555.04(814.65)	**0.41(1.91)**	11.30(11.38)	21.28(129.61)	–
F5	T6	1721.03(619.91)	284.75(620.42)	**0.23(0.94)**	3.88(3.30)	4.75(18.11)	–
F5	T7	4.52(9.95)	13.53(111.51)	**0.43(2.23)**	5.83(5.70)	2.71(5.18)	–
F6	T1	12.77(23.99)	11.27(14.04)	6.23(10.44)	4.91(6.27)	**3.13(5.46)**	49.79%
F6	T2	266.98(348.91)	299.47(381.91)	**10.31(13.23)**	13.15(13.67)	19.47(63.91)	–
F6	T3	264.82(345.74)	296.77(374.12)	**10.95(23.30)**	15.86(36.52)	26.70(89.05)	–
F6	T4	95.30(199.81)	75.59(203.23)	**6.79(10.17)**	6.85(6.79)	6.90(9.65)	–
F6	T5	267.86(337.79)	305.53(364.58)	**14.95(45.21)**	22.63(61.86)	60.28(164.06)	–
F6	T6	113.59(220.48)	78.20(207.82)	**7.80(10.96)**	8.63(9.01)	12.97(55.93)	–
F6	T7	57.70(143.39)	51.06(149.20)	**10.74(14.73)**	12.75(13.32)	15.51(39.20)	–

Los valores en negrita corresponden al mejor algoritmo.

algoritmo (paradigma vs. enfoques para lidiar con la dinámica del problema)?
2. ¿Cuán competitivos pueden llegar a ser los algoritmos auto-adaptativos frente a otros de la literatura?

Como se observa, algunas de estas cuestiones han sido parcialmente respondidas en determinados trabajos revisados en la Sección 5.1. Sin embargo, en la mayoría de estos las respuestas han sido derivadas a partir del estudio sobre escenarios dinámicos simples, los cuales difieren sustancialmente de otros más modernos y por tanto más complejos. Además, hasta donde se conoce, no existen estudios que analicen la cuestión 1, principalmente debido a la ausencia de propuestas que incluyan al mismo tiempo modelos auto-adaptativos tanto en el paradigma, como en algún enfoque de adaptación dinámica. Finalmente, hemos incluido el importante aspecto de la competitividad de los algoritmos considerados en el estudio, lo cual que se realizará a través de la comparación con otros algoritmos actuales.

Para analizar adecuadamente la primera cuestión, hemos considerado los algoritmos mostrados en la Figura 5.10. Estos métodos han sido derivados de la combinación de distintos paradigmas basados en DE, y el enfoque de adaptación dinámica basado en la generación de individuos quantum (Blackwell y Branke, 2006). Como se aprecia en la Figura 5.10, por el eje de las x aparecen las variantes del enfoque quantum: *ninguno* (el enfoque quantum no está presente), *determinístico* (variante original (Blackwell y Branke, 2006)), y *auto-adaptativo* (propuesto en la Sección 5.2). Por otra parte, en el eje y se encuentra los paradigmas considerados: DE (Storn y Price, 1997), jDE (Brest et al, 2006) y SspDE (Pan et al, 2011). En este último caso es importante aclarar que el paradigma jDE seleccionado en este estudio no es el mismo algoritmo analizado en la sección anterior, el cual posee el mismo nombre y fue propuesto en (Brest et al, 2009). En este sentido, a menos que se establezca lo contrario, en lo que sigue cuando se haga alusión a jDE se tratará del paradigma propuesto en (Brest et al, 2006), empleado en problemas estacionarios.

Es importante notar además que los algoritmos que incluyen la variante auto-adaptativa del enfoque quantum (representada por las siglas SQ), incorporan también el esquema de interacción de individuos descrito en la Sección 5.2.2. De esta forma, el algoritmo mSQDE-i de la Figura 5.10 es el mismo estudiado en los experimentos de la Sección 5.2.5.

A continuación se describen los paradigmas jDE y SspDE. Es necesario apuntar que en los experimentos se utilizaron las 49 instancias de problema del generador GDBG (Li et al, 2008) (Sección 2.4.2). Asimismo, se siguió la metodología descrita en la Sección 3.2 para el análisis estadístico de los resultados.

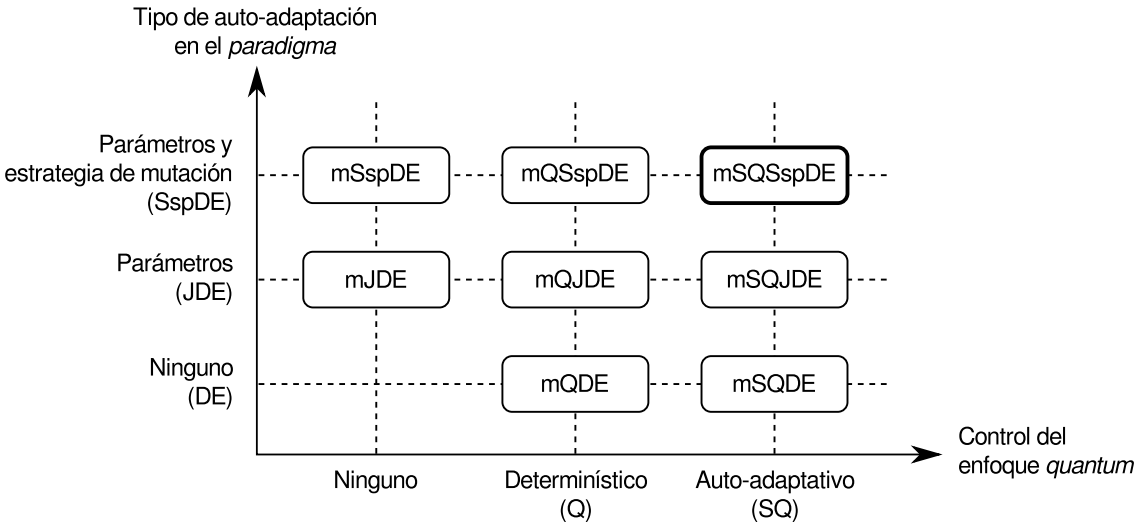

Figura 5.10 Agoritmos derivados a partir de la combinación de varios paradigmas y distintas variantes del enfoque *quantum*.

5.3.1 Paradigmas seleccionados

Los paradigmas seleccionados en nuestro estudio extienden al paradigma evolución diferencial (DE). La elección se ha basado principalmente en el éxito mostrado por estos algoritmos en el contexto de la optimización estacionaria, y por la presencia de modelos auto-adaptativos de diversa tipología. En primer lugar hemos seleccionado al propio paradigma DE (estándar), descrito en la Sección 2.2.5. En segundo lugar, se consideró la extensión jDE (Brest et al, 2006, 2007) que, en contraste con el DE estándar, presenta auto-adaptación a nivel de individuos de los parámetros CR y F. Finalmente, por sus excelentes resultados en ambientes estacionarios, y por poseer un modelo auto-adaptativo distinto, hemos considerado también al algoritmo SspDE (Pan et al, 2011). En lo que sigue, describiremos los pasos fundamentales de las extensiones jDE y SspDE.

Extensión jDE. Los pasos de jDE son básicamente los mismos que los del DE estándar. En el caso del proceso de inicialización del algoritmo, también tienen que tenerse en cuenta los parámetros estratégicos CR y F. El Algoritmo 5.2 muestra como se lleva a cabo este proceso.

En relación al ciclo evolutivo de jDE, la diferencia en relación a DE radica en que cada descendiente es creado empleando las realizaciones mutadas de los parámetros CR_i y F_i, los cuales son codificados por el individuo $\mathbf{y}_i$. Además, durante el proceso de selección, si el descendiente es mejor que el individuo $\mathbf{y}_i$, entonces los nuevos parámetros $\tilde{\mathsf{CR}}_i$ y $\tilde{\mathsf{F}}_i$ son heredados por $\mathbf{y}_i$. Estos pasos son descritos en el Algoritmo 5.3.

```
1 Asignar τ1 ← 0.1;
2 Asignar τ2 ← 0.1;
3 para cada individuo y_i ∈ P hacer
4 |   Inicializar aleatoriamente x_i en el espacio de búsqueda;
5 |   Asignar CR_i ← alea(0,1);
6 |   Asignar F_i ← 0.1 + 0.9 · alea(0,1);
7 |   Evaluar x_i;
8 fin
```

Algoritmo 5.2: Inicialización del algoritmo jDE.

```
1 para cada individuo y_i ∈ P hacer
2 |   Mutar CR_i y F_i de acuerdo a las expresiones 2.11 y 2.12, respectivamente.;
3 |   Crear un descendiente ỹ_i de acuerdo a los pasos de DE estándar usando los
  |   parámetros CR_i, F_i;
4 |   Evaluar ỹ_i;
5 |   si f̃_i ≻ f_i entonces
6 |   |   y_i ← ⟨x̃_i, f̃_i, C̃R_i, F̃_i⟩ ;
7 |   fin
8 fin
```

Algoritmo 5.3: Ciclo evolutivo del algoritmo jDE.

Algoritmo SspDE. En SspDE (Pan et al, 2011) no solo se auto-adaptan los parámetros CR y F, sino también la estrategia de mutación (véase Sec. 2.2.5). Sin embargo, a diferencia de los modelos auto-adaptativos que realizan estas adaptaciones en cada iteración, en SspDE se ejecutan cada LP generaciones y están basadas en listas *ganadoras* de estas realizaciones de parámetros y estrategia de mutación. Formalmente, cada individuo es representado de la siguiente manera:

$$\mathbf{y}_i = \langle \mathbf{x}_i, f_i, \mathsf{CRL}_i, \mathsf{FL}_i, \mathsf{SL}_i, \mathsf{wCRL}_i, \mathsf{wFL}_i, \mathsf{wSL}_i \rangle$$

donde $\mathsf{CRL}, \mathsf{FL}$ y SL son listas de longitud *LP* compuestas por realizaciones de los parámetros CR, F y las estrategias de mutación definidas en la Sección 2.2.5, respectivamente. En particular estas cuatro estrategias estarán representadas por el conjunto $\mathscr{S}$, siendo $\mathsf{aleaElement}(\cdot)$ una función que devuelve elementos aleatorios del conjunto en cuestión. Por otro lado, $\mathsf{wCRL}_i, \mathsf{wFL}_i$ y wSL_i son las listas *ganadoras*.

Al inicio de la ejecución, las primeras listas son inicializadas como se muestra en el Algoritmo 5.4. Note que en este pseucódigo se ha incluido también el contador de generaciones lp, el cual es incrementado en cada generación como se verá más adelante.

En cada generación, SspDE evoluciona a la población usando las realizaciones CRL_i^{lp}, FL_i^{lp} y SL_i^{lp}. Nuevamente, si el descendiente es mejor que el individuo *i*, estas realizaciones

```
1  Asignar LP ← 30;
2  Asignar lp ← 0;
3  Asignar RP ← 0.8;
4  para cada individuo y_i ∈ 𝒫 hacer
5      Inicializar aleatoriamente x_i en el espacio de búsqueda;
6      para l ← 1 hasta LP hacer
7          Asignar CRL_i^l ← alea(0,1);
8          Asignar FL_i^l ← 0.1 + 0.9 · alea(0,1);
9          Asignar SL_i^l ← aleaElement(𝒮);
10     fin
11     Asignar wFL_i ← ∅ ;
12     Asignar wCRL_i ← ∅ ;
13     Asignar wSL_i ← ∅ ;
14 fin
```

Algoritmo 5.4: Inicialización del algoritmo SspDE.

son añadidas a sus respectivas listas *ganadoras*. Como puede ser observado en el Algoritmo 5.5, cuando $lp = \mathsf{LP}$, entonces ocurre el proceso de auto-adaptación de las listas CRL_i, FL_i y SL_i. Específicamente, estas listas se llenan a partir del cruzamiento probabilístico, según la tasa RP, de los elementos de las listas *ganadoras* y elementos seleccionados aleatoriamente. En caso de que las listas *ganadoras* estuvieran vacías entonces se asume que las existentes son óptimas. Este tipo de auto-adaptación resulta especial si se tiene en cuenta que no se trata de adaptar un valor puntual, sino una lista de elementos. En otras palabras, este modelo controla el orden de aplicación de los parámetros estratégicos a nivel de individuo.

Finalmente es importante apuntar que los paradigmas considerados tendrán como plantilla común la del enfoque multipoblacional descrito por el Algoritmo 5.6.

5.3.2 Resultados y análisis estadístico

Los resultados de los algoritmos considerados en este estudio se encuentran, en términos del error de la mejor solución antes del cambio, en las Tablas 5.25–5.28. Además, el rendimiento relativo de estas ejecuciones se ha graficado en las Figuras 5.11 y 5.12.

Para analizar estadísticamente estos resultados, se ha seguido la metodología descrita en la Sección 3.2. De manera particular, se dividió el análisis en tres grupos, de acuerdo al tipo de función empleada por las instancias de problema. En el primer grupo se consideraron las instancias de problema con funciones unimodales (F1, F1(50) y F2), en el segundo las instancias con funciones multimodales (F3–F6), mientras que el tercero considera a todas las instancias.

Siguiendo la metodología, primero se aplicaron pruebas de Friedman e Iman-Davenport

```
para cada individuo y_i ∈ P hacer
    Crear un descendiente ỹ_i de acuerdo a los pasos de DE (Alg. 2.3) empleando los parámetros
    CRL_i^lp, FL_i^lp y la estrategia SL_i^lp;
    Evaluar a ỹ_i;
    si f̃_i ≻ f_i entonces
        Asignar y_i ← ỹ_i;
        Añadir CRL_i^lp a wCRL_i ;
        Añadir FL_i^lp a wFL_i ;
        Añadir SL_i^lp a wSL_i ;
    fin
fin
si lp = LP entonces
    Asignar lp ← 1;
    // Fase de auto-adaptación
    para cada individuo y_i ∈ P hacer
        si wCRL_i, wFL_i, wSL_i no están vacías entonces
            para l ← 1 hasta LP hacer
                si alea(0,1) < RP entonces
                    Asignar CRL_i^l ← aleaElement(wCRL_i);
                fin
                si no
                    Asignar CRL_i^l ← alea(0,1);
                fin
            fin
            para l ← 1 hasta LP hacer
                si alea(0,1) < RP entonces
                    Asignar FL_i^l ← aleaElement(wFL_i);
                fin
                si no
                    Asignar FL_i^l ← 0.1 + 0.9 · alea(0,1);
                fin
            fin
            para l ← 1 hasta LP hacer
                si alea(0,1) < RP entonces
                    Asignar SL_i^l ← aleaElement(wSL_i);
                fin
                si no
                    Asignar SL_i^l ← aleaElement(S);
                fin
            fin
        fin
        Asignar wFL_i ← ∅ ;
        Asignar wCRL_i ← ∅ ;
        Asignar wSL_i ← ∅ ;
    fin
fin
si no
    Incrementar lp ← lp + 1;
fin
```

Algoritmo 5.5: Ciclo evolutivo del algoritmo SspDE.

```
1  para cada subpoblación P_k hacer
2      Inicializar aleatoria a P_k de acuerdo al paradigma empleado (ej. DE, jDE, SspDE);
3      Actualizar la mejor solución de P_k;
4  fin
5  mientras no condición de parada hacer
6      Aplicar principio de exclusión;
7      si no se detectado un cambio entonces
8          para cada subpoblación P_k hacer
9              Iterar los individuos convencionales de acuerdo al paradigma empleado (ej. DE,
               jDE, SspDE);
10             si se utiliza el enfoque quantum entonces
11                 Iterar los individuos de tipo quantum de acuerdo al enfoque empleado (ej.
                   determinístico o auto-adaptativo);
12             fin
13             Actualizar la mejor solución de P_k;
14         fin
15     fin
16     si no
17         para cada subpoblación P_k hacer
18             Reevaluar los individuos convencionales de P_k;
19             Actualizar la mejor solución de P_k;
20         fin
21     fin
22 fin
```

Algoritmo 5.6: Plantilla general de los algoritmos.

para identificar diferencias significativas a nivel de grupo, esto es, de algoritmos. En ese sentido, la Tabla 5.20 muestra los $p-$valores correspondientes a estas pruebas aplicadas a los tres grupos de datos considerados en nuestro estudio. Los bajos $p-$valores (≤ 0.05) indican que, en sentido general, tales diferencias existen en todos los grupos. Adicionalmente, los rangos medios según Friedman se muestran en la Tabla 5.21. Para detectar en que pares de algoritmos ocurren estas diferencias, se aplicó la prueba post-hoc de Holm, el cuál arrojó los $p-$valores de las Tablas 5.22, 5.23, correspondientes a las múltiples comparaciones realizadas entre los algoritmos. Con el objetivo de mostrar una vista resumida de este último prueba y los rangos medios de los algoritmos, obtenidos a partir de la prueba de Friedman, se crearon las gráficas de la Figura 5.13.

Empleando estos elementos metodológicos, en lo que sigue se analizaran las cuestiones mencionadas al comienzo de esta Sección.

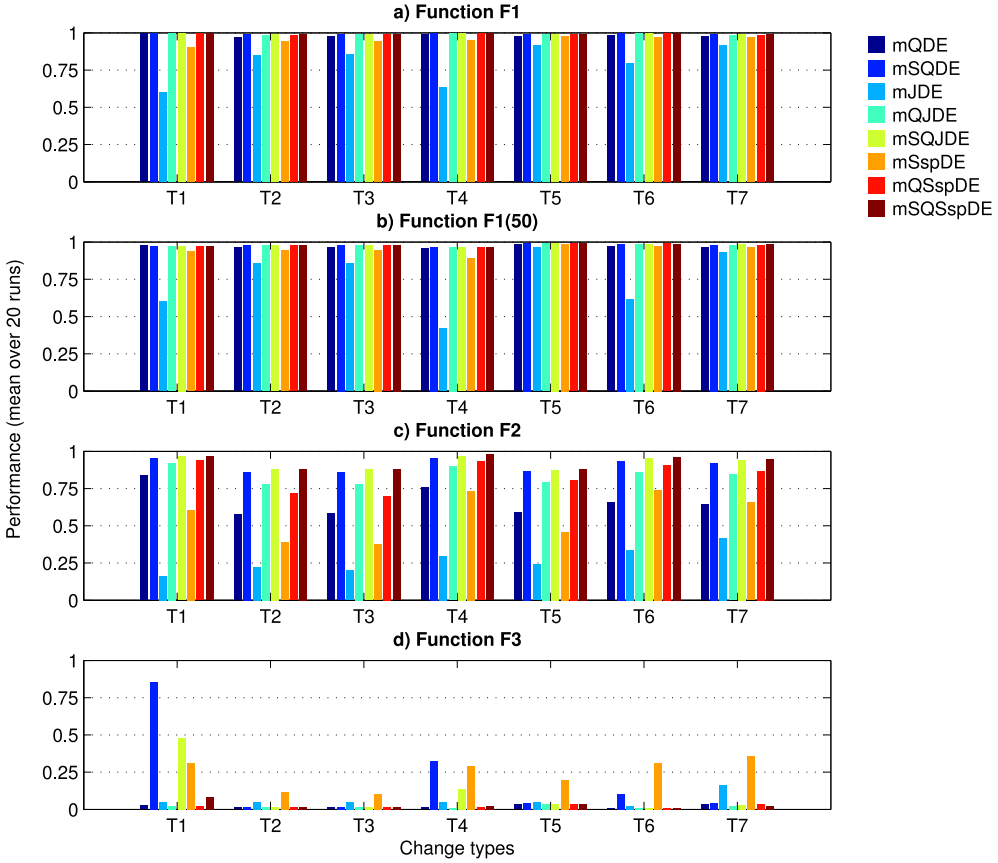

Figura 5.11 Rendimiento relativo de los algoritmos en instancias de problema del generador GDBG con funciones pico F1–F3.

Tabla 5.20 Resultados de las pruebas de Friedman e Iman-Davenport ($\alpha = 0.05$) considerando todos los algoritmos en instancias de problema del generador GDBG.

	Funciones pico		
Prueba	Unimodales	Multimodales	Todas
Friedman	**8.35E−11**	**5.28E−11**	**9.37E−11**
Iman-Davenport	**4.44E−16**	**1.50E−32**	**4.44E−16**

Los valores en negrita indican que existen diferencias significativas a nivel de grupo (p-valor<0.05).

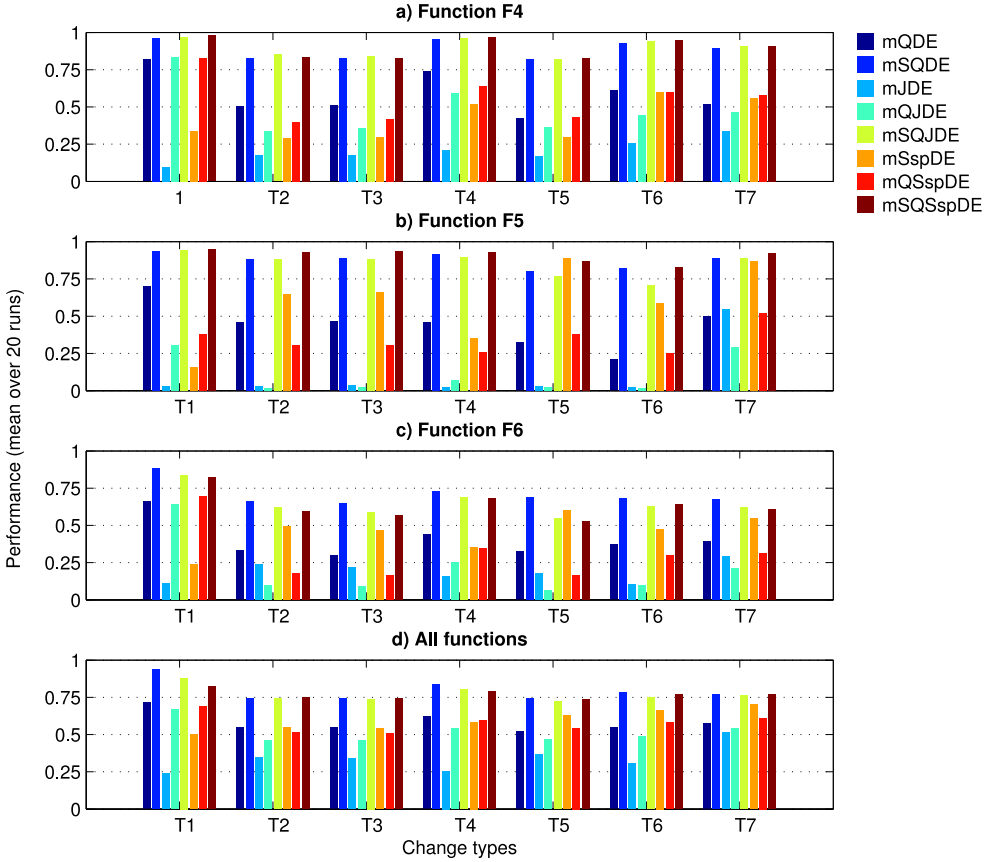

Figura 5.12 Rendimiento relativo de los algoritmos en instancias de problema del generador GDBG con funciones pico F4, F5, F6, y todas.

Tabla 5.21 Rangos medios de los algoritmos a partir de la prueba de Friedman en instancias de problema del generador GDBG.

	Funciones pico		
Algoritmos	Unimodales	Multimodales	Todas
mQDE	5.857	4.249	4.939
mSQDE	2.619	1.929	2.224
mJDE	7.762	6.571	7.082
mQJDE	4.476	7.286	6.082
mSQJDE	2.191	2.929	2.612
mSspDE	6.952	4.321	5.449
mQSspDE	4.286	5.607	5.041
mSQSspDE	1.857	3.107	2.571

Tabla 5.22 Múltiples comparaciones según la prueba de Holm (α =0.05) de los algoritmos, en instancias de problema del generador GDBG con funciones pico unimodales (F1–F2).

	mSQDE	mJDE	mQJDE	mSQJDE	mSspDE	mQSspDE	mSQSspDE
mQDE	**3.31E−04**	1.29E−01	4.74E−01	**2.59E−05**	8.84E−01	3.01E−01	**2.67E−06**
mSQDE		**2.66E−10**	1.40E−01	1.71E+00	**2.28E−07**	2.47E−01	1.42E+00
mJDE			**2.63E−04**	**4.60E−12**	1.42E+00	**8.51E−05**	**1.59E−13**
mQJDE				**3.25E−02**	1.58E−02	1.71E+00	**8.49E−03**
mSQJDE					**7.17E−09**	6.69E−02	1.71E+00
mSspDE						**7.13E−03**	**3.95E−10**
mQSspDE							**1.84E−02**

Los valores corresponden a los *p*-valores ajustados. Los valores en negrita indican que existen diferencias significativas entre los algoritmos (*p*-valor<0.05).

Tabla 5.23 Múltiples comparaciones según la prueba de Holm (α =0.05) de los algoritmos, en instancias de problema del generador GDBG con funciones pico multimodales (F3–F6).

	mSQDE	mJDE	mQJDE	mSQJDE	mSspDE	mQSspDE	mSQSspDE
mQDE	**6.26E−03**	**6.26E−03**	**7.42E−05**	4.35E−01	1.57E+00	4.20E−01	5.09E−01
mSQDE		**3.57E−11**	**7.74E−15**	6.33E−01	**4.37E−03**	**4.61E−07**	5.09E−01
mJDE			8.26E−01	**6.04E−07**	**8.24E−03**	6.33E−01	**2.66E−06**
mQJDE				**7.33E−10**	**1.19E−04**	1.34E−01	**4.34E−09**
mSQJDE					4.00E−01	**8.14E−04**	1.57E+00
mSspDE						4.46E−01	5.09E−01
mQSspDE							**2.41E−03**

Los valores corresponden a los *p*-valores ajustados. Los valores en negrita indican que existen diferencias significativas entre los algoritmos (*p*-valor<0.05).

Tabla 5.24 Múltiples comparaciones según la prueba de Holm (α =0.05) de los algoritmos, en todas las instancias de problema del generador GDBG.

	mSQDE	mJDE	mQJDE	mSQJDE	mSspDE	mQSspDE	mSQSspDE
mQDE	**7.45E−07**	**1.94E−04**	2.09E−01	**3.62E−05**	1.82E+00	2.05E+00	**2.58E−05**
mSQDE		**2.72E−21**	**1.62E−13**	2.05E+00	**1.59E−09**	**2.40E−07**	2.05E+00
mJDE			3.46E−01	**4.41E−18**	**1.07E−02**	**4.47E−04**	**2.15E−18**
mQJDE				**5.45E−11**	1.41E+00	3.19E−01	**3.15E−11**
mSQJDE					**1.98E−07**	**1.48E−05**	2.05E+00
mSspDE						2.05E+00	**1.28E−07**
mQSspDE							**1.03E−05**

Los valores corresponden a los *p*-valores ajustados. Los valores en negrita indican que existen diferencias significativas entre los algoritmos (*p*-valor<0.05).

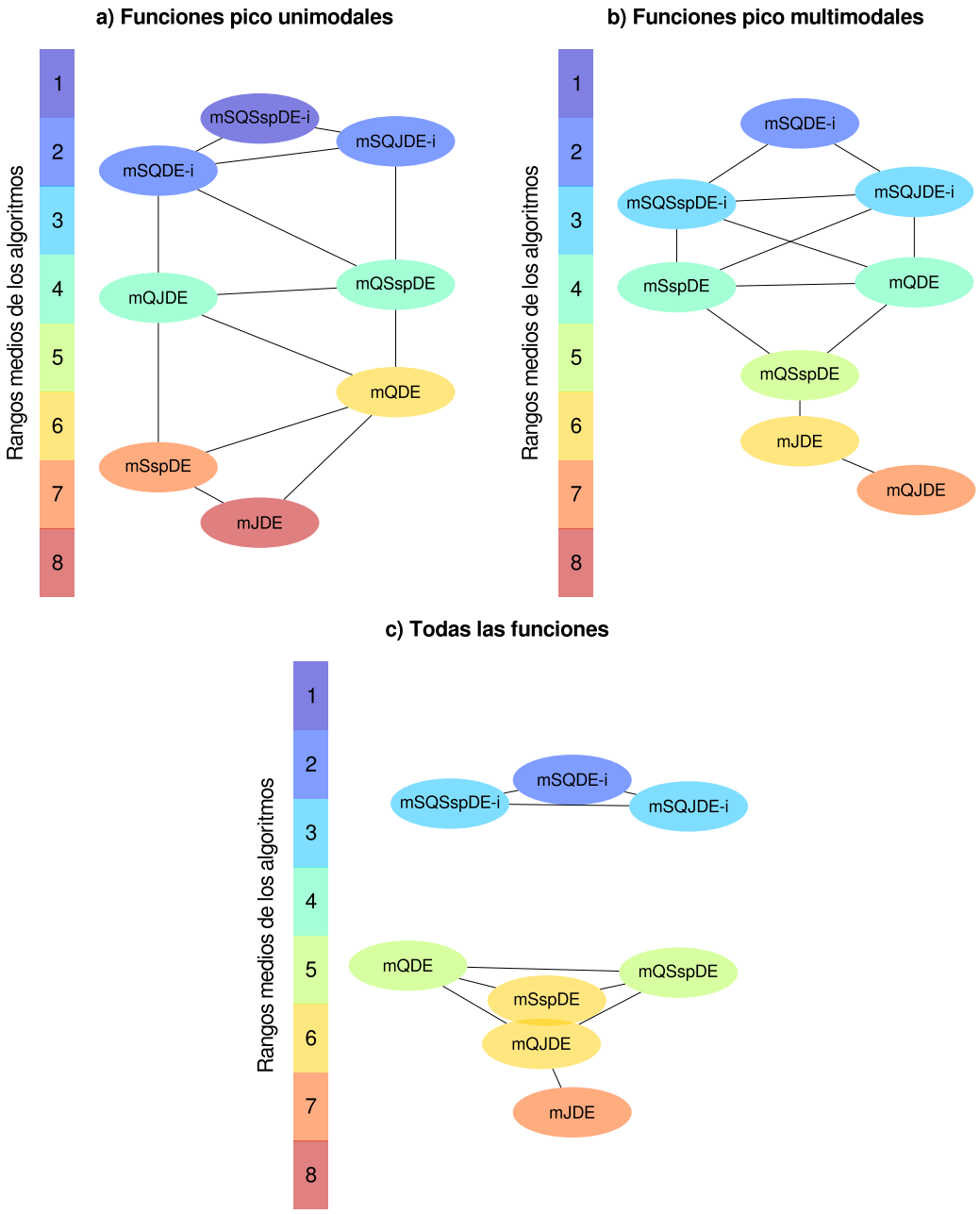

Figura 5.13 Posiciones relativas de los algoritmos de acuerdo a sus rangos medios (test de Friedman). Los arcos directos indican, según los *p*-valores ajustados de la prueba de Holm, que no existen diferencias entre los algoritmos involucrados.

Efecto de la auto-adaptación en diferentes partes del algoritmo

En este apartado se analizará en que contextos resulta más beneficioso la aplicación de modelos auto-adaptativos en ambientes dinámicos. En los algoritmos diseñados para este estudio experimental existen tres variantes, como se vio en la Tabla 5.1. En el primer grupo, se encuentran los algoritmos mJDE y mSspDE (con auto-adaptación en el paradigma, JDE y SspDE, respectivamente). En el segundo, aparece el algoritmo mSQDE, con auto-adaptación solo en el enfoque *quantum*. El tercer grupo incluye a los algoritmos más sofisticados, esto es, aquellos con auto-adaptación en el paradigma y en el enfoque quantum: mSQJDE y mSQSspDE.

A partir de la Figura 5.13-a) se aprecia que, en problemas con funciones pico unimodales, los algoritmos del Grupo I son los que peor rendimiento, incluso significativamente peores que sus variantes determinísticas mQJDE y mQSspDE. Por su parte, el algoritmo mSQDE (Grupo II) posee un rendimiento similar que los mejores mSQJDE y mSQSspDE (Grupo III). En el caso de los problemas con funciones multimodales 5.13 b), existe una pequeña variación a favor de los del Grupo I, en especial para el algoritmo mSspDE que alcanza un rendimiento similar al de los del Grupo III, sin embargo, significativamente inferior al de mSQDE (el mejor en estas instancias). De manera global, Figura 5.13 c), se puede ver que los algoritmos del Grupo II y III son significativamente mejores que los del Grupo I. Esto indica que, al menos para estas instancias de problema, la auto-adaptación aplicada solo a nivel de paradigma no resulta suficiente para garantizar un rendimiento adecuado en ambientes dinámicos. Note que esta conclusión es en cierta medida consistente con las obtenidas en algunas investigaciones consideradas como *Estudios básicos* en la Sección 5.1.1. Como consecuencia, se puede ver que la auto-adaptación posee un mayor impacto cuando es aplicada a los enfoques para lidiar con la dinámica del problema.

Con el objetivo de explicar con más detalles el comportamiento de los algoritmos en función de los esquemas auto-adaptativos presentes en ellos, en este apartado se hace un análisis de la adaptabilidad de los algoritmos. Para ello se han seleccionado dos instancias de problemas basadas en las funciones F1 (Rotation peaks) y F3 (Composition Rastrigin), las cuales emplean el tipo de cambio T2 (Paso largo). La intención es analizar problemas con diversas características. Además, los algoritmos seleccionados son: mSQDE (auto-adaptación quantum), mJDE y mSspDE (con auto-adaptación solo a nivel de paradigma). Resulta de interés entonces, conocer en qué medida son efectivos los modelos auto-adaptativos estudiados. En este caso conviene definir qué objetivo tienen estos modelos en el contexto de la optimización en ambientes dinámicos. Se puede ver que el papel de los modelos tratados en esta investigación es influir inteligentemente en el balance de explotación y exploración del algoritmo.

En el contexto de los algoritmos poblacionales, lo anterior se traduce en un mantenimiento adecuado de la diversidad de los individuos de la población. Idealmente, la diversidad de la población debería estar en correspondencia con el estado de la búsqueda. Por ejemplo, debe haber un nivel bajo de diversidad si el óptimo del problema está cerca y viceversa. De manera que se puede decir que los modelos auto-adaptativos influyen en la diversidad de la población con el objetivo de disminuir la distancia al óptimo del problema. Basados en este último razonamiento, en este trabajo proponemos la siguiente medida para cuantificar el nivel de *adaptabilidad* de los algoritmos:

$$adaptabilidad = relDistOpt - relDivPob \tag{5.9}$$

donde $relDistOpt$ y $relDivPob$ son a su vez, magnitudes normalizadas correspondientes a la distancia al óptimo del problema, y la diversidad de la población, respectivamente (véase Deb y Beyer (2001) y Weicker (2002)). Ambas magnitudes varían en el intervalo $[0,1]$ por tanto, $adaptabilidad \in [-1;1]$. En este sentido, si la $adaptabilidad$ se acerca a 1 indica que el algoritmo es incapaz de generar la suficiente diversidad para alcanzar al óptimo. Por el contrario, un valor cercano a -1, indica que el algoritmo genera mucho más diversidad que la necesaria. De manera que el valor ideal sería 0, el cual significa que el algoritmo genera una diversidad acorde a la distancia del óptimo del problema.

Teniendo en cuenta estos aspectos, considérese la Figura 5.14 donde se muestra la adaptabilidad de los algoritmos mSQDE, mJDE y mSspDE. En particular las gráficas 5.14-a) y c) muestran la adaptabilidad en función de las generaciones del algoritmo, mientras que las gráficas 5.14-b) y d) exhiben el promedio acumulado de estos valores. Estas últimas gráficas han sido empleadas para visualizar la tendencia general de esta medida.

Como se aprecia, el comportamiento de los algoritmos es consistente con el análisis estadístico previo. Por ejemplo, el algoritmo mSQDE presenta una mejor adaptabilidad que el resto en el escenario con función F1 (véase la gráfica 5.14-b). No obstante, a juzgar por la gráfica 5.14-a), los tres algoritmos presentan comportamientos auto-adaptativos en general, ya intentan generar diversidad después de cada cambio. Los cambios en el ambiente se encuentran en las gráficas cada 10 generaciones del algoritmo. Sin embargo, de los tres, solo mSQDE logra una adaptación más rápida y efectiva que el resto. Note como el valor acumulado de la adaptabilidad de este algoritmo tiende a 0 en menor tiempo que el resto. En el caso del escenario con función F3, este comportamiento auto-adaptativo deja de ser tan visible como en el caso anterior (gráfica 5.14-c). Indudablemente, la complejidad de la función *Composition Rastrigin* afecta de forma negativa la adaptabilidad de todos los algoritmos de manera similar, de ahí que los tres exhiban tendencias similares (gráfica 5.14-d). Pero, en todos los casos con tendencias que nunca se aproximan a 0 (al menos para

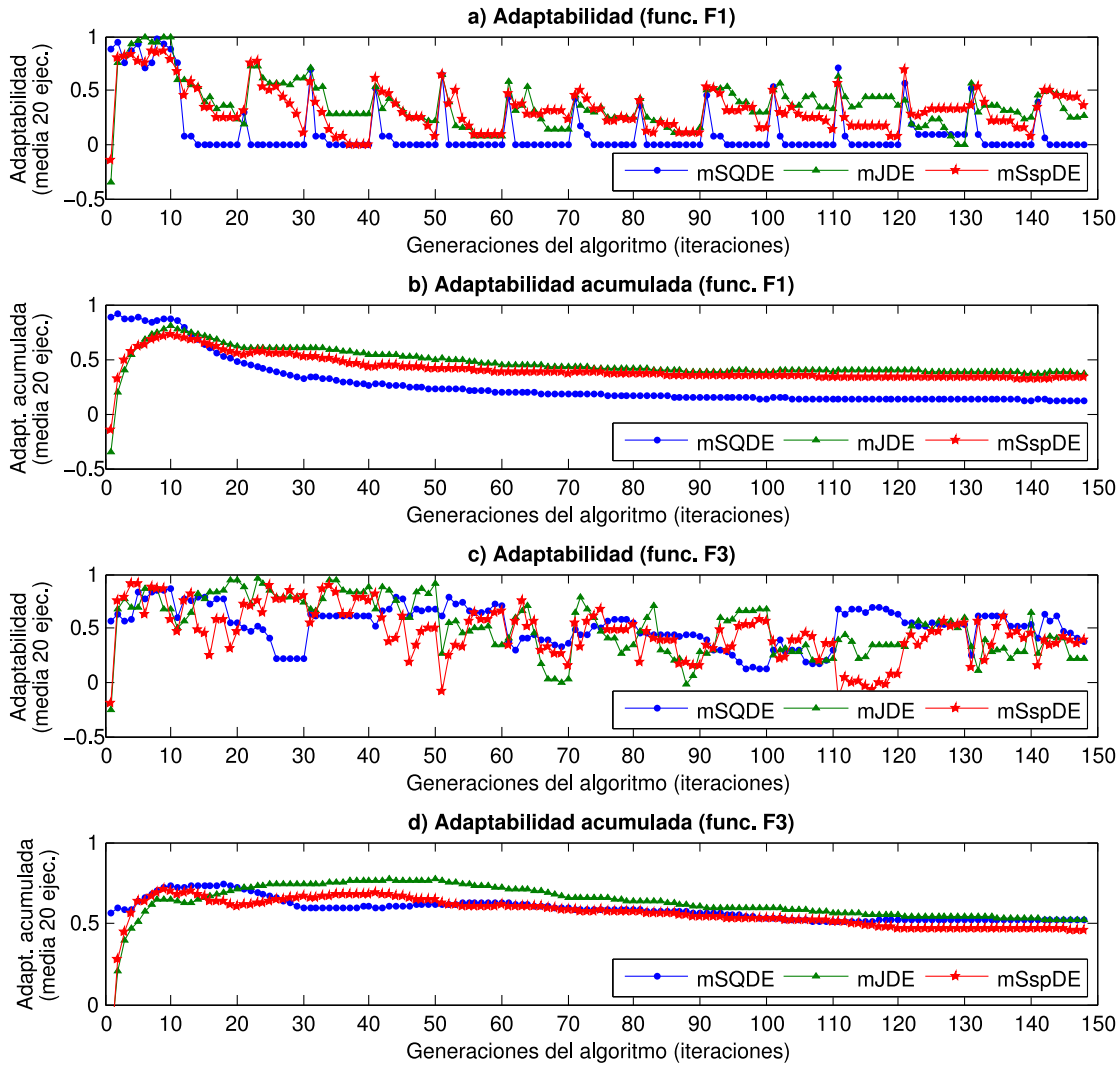

Figura 5.14 Evolución en el tiempo de la *adaptabilidad* de los algoritmos mSQDE, mJDE, y mSspDE, en instancias de problema con función F1 y F3, y tipo de cambio T2.

los 15 cambios representados en la gráfica 5.14-d).

Competitividad de los algoritmos auto-adaptativos

No obstante analizadas las cuestiones anteriores, resulta también de interés estudiar cuán competitivos son los algoritmos auto-adaptativos (mSQDE, mSQJDE y mSQSspDE) en relación a otros de la literatura. En ese sentido, hemos realizado las comparaciones con los algoritmos: jDE Brest et al (2009), DynDE (Mendes y Mohais, 2005), mQSO (Blackwell y Branke, 2006), CPSO (Li y Yang, 2009), y SACDEEA (Halder et al, 2011). A excepción de mQSO (que se ha implementado), el resto posee resultados reportados en el generador GDBG. En particular, los valores utilizados para DynDE provienen del trabajo (Halder et al,

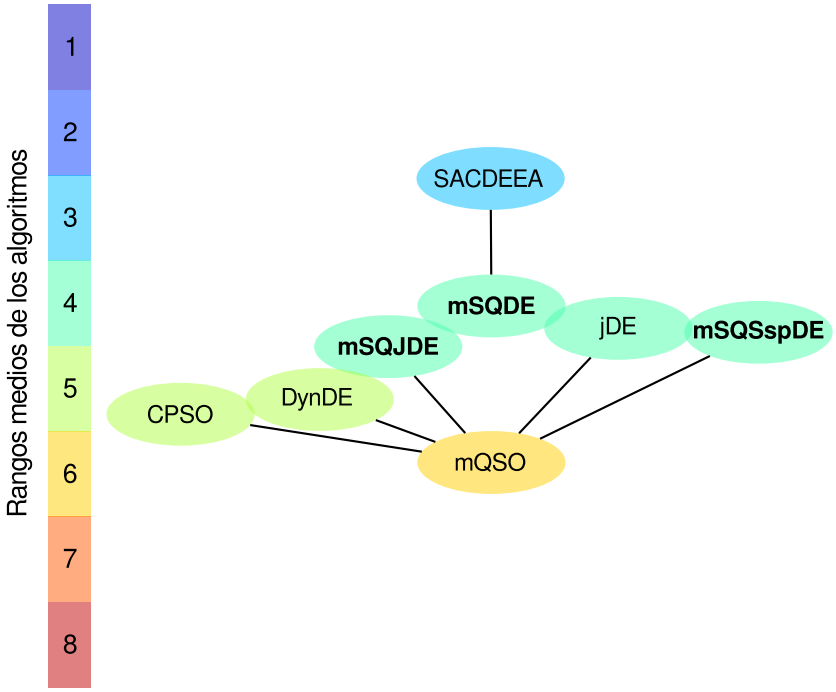

Figura 5.15 Posiciones relativas de los algoritmos auto-adaptativos (mSQDE, mSQJDE, mSQSspDE) vs. otros de la literatura, de acuerdo a sus rangos medios (test de Friedman). Los arcos directos indican, según los *p*-valores ajustados de la prueba de Holm, que no existen diferencias entre los algoritmos involucrados en relación al mejor (SACDEEA) y al peor (mQSO).

2011). Con el objetivo de hacer justa la comparación, se han eliminado del análisis aquellos resultados derivados de instancias de problema con tipo de cambio T7, debido a que (Halder et al, 2011) no la incluye entre sus resultados.

A diferencia de los análisis anteriores, en esta ocasión no se dividió el análisis por tipo de función, esto es, se consideraron todas las instancias de problema al mismo tiempo. En este contexto, las pruebas de Friedman e Iman-Davenport dieron como resultado p-valores inferiores al nivel de significación (0.05). De manera que se procedió con las pruebas post-hoc para determinar las diferencias entre los algoritmos. La Figura 5.15 resume esta prueba y los rangos medios derivados de la prueba de Friedman.

Como se aprecia, los algoritmos propuestos en nuestra investigación alcanzan posiciones competitivas frente a los de la literatura. Aunque en sentido general, los tres son inferiores comparados con SACDEEA, es de notar en ese sentido que mSQDE no lo es significativamente (aunque sí lo es en relación al peor, en este caso mQSO). Los tres son además similares en rendimiento a CPSO, DynDE, y JDE.

Tabla 5.25 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones y desviación estándar) de los algoritmos mQDE, mSQDE, mJDE, y mQJDE, en instancias del generador GDBG con funciones F1–F2.

Func.	Cambio	mQDE	mSQDE	mJDE	mQJDE
F1(10)	T1	4.46E−01(9.70E−01)	1.14E−01(7.80E−01)	3.59E+01(1.80E+01)	1.70E−01(7.82E−01)
	T2	2.51E+00(5.79E+00)	1.06E+00(4.12E+00)	1.38E+01(1.63E+01)	1.32E+00(4.53E+00)
	T3	2.36E+00(5.34E+00)	9.09E−01(3.52E+00)	1.34E+01(1.59E+01)	1.24E+00(4.19E+00)
	T4	9.40E−01(1.26E+00)	1.12E−01(5.08E−01)	3.60E+01(2.80E+01)	2.36E−01(4.18E−01)
	T5	2.23E+00(3.12E+00)	**5.83E−01(2.07E+00)**	7.42E+00(9.32E+00)	9.57E−01(2.48E+00)
	T6	1.71E+00(2.29E+00)	**1.44E−01(6.35E−01)**	2.03E+01(2.38E+01)	3.40E−01(9.78E−01)
	T7	2.37E+00(5.27E+00)	1.11E+00(4.03E+00)	7.64E+00(1.05E+01)	1.41E+00(4.71E+00)
F1(50)	T1	**2.20E+00(2.83E+00)**	2.78E+00(4.21E+00)	3.84E+01(1.98E+01)	2.76E+00(4.13E+00)
	T2	3.67E+00(4.50E+00)	2.31E+00(3.90E+00)	1.42E+01(1.53E+01)	2.48E+00(4.03E+00)
	T3	3.41E+00(4.70E+00)	2.03E+00(3.65E+00)	1.42E+01(1.63E+01)	2.41E+00(4.11E+00)
	T4	4.27E+00(5.81E+00)	3.39E+00(6.13E+00)	5.79E+01(2.73E+01)	**3.28E+00(5.99E+00)**
	T5	1.53E+00(1.78E+00)	6.12E−01(1.28E+00)	3.57E+00(4.90E+00)	7.16E−01(1.32E+00)
	T6	2.73E+00(2.51E+00)	1.27E+00(2.23E+00)	3.82E+01(3.09E+01)	1.35E+00(2.30E+00)
	T7	3.43E+00(4.53E+00)	1.99E+00(3.66E+00)	7.05E+00(7.99E+00)	2.16E+00(3.92E+00)
F2	T1	3.92E+00(4.83E+00)	1.40E+00(4.64E+00)	2.05E+02(1.25E+02)	1.89E+00(4.08E+00)
	T2	3.67E+01(9.22E+01)	4.82E+00(1.51E+01)	2.68E+02(2.15E+02)	1.06E+01(3.93E+01)
	T3	3.68E+01(9.16E+01)	5.93E+00(2.19E+01)	2.83E+02(2.12E+02)	1.27E+01(5.05E+01)
	T4	4.02E+00(3.60E+00)	5.86E−01(1.36E+00)	1.16E+02(1.48E+02)	1.36E+00(1.50E+00)
	T5	6.38E+01(1.11E+02)	1.02E+01(3.01E+01)	2.65E+02(1.69E+02)	2.01E+01(5.46E+01)
	T6	7.23E+00(6.62E+00)	9.01E−01(1.74E+00)	1.03E+02(1.49E+02)	2.04E+00(2.59E+00)
	T7	1.17E+01(1.14E+01)	1.95E+00(4.64E+00)	8.57E+01(1.27E+02)	4.09E+00(6.72E+00)

5.4 Conclusiones

Este Capítulo estuvo dedicado al análisis del rol de la auto-adaptación en ambientes dinámicos. A partir de la revisión realizada sobre el estado-del-arte de esta área del conocimiento se organizaron las investigaciones existentes de acuerdo a dos categorías: 1) tipo de investigación (Estudios básicos, Trabajos teóricos, y Trabajos avanzados), y 2) qué elemento del algoritmo es objeto de auto-adaptación (Paradigma, enfoque de adaptación, o ambos). En ese sentido, se pudo concluir que la mayoría de los trabajos existentes pertenecen a la categoría *Estudios básicos*, los cuales se caracterizan por la aplicación de modelos auto-adaptativos a nivel de paradigma, esto es, tal y como fueron concebidos para la optimización estacionaria. Particularmente, estas investigaciones poseen opiniones divididas en relación a sí este tipo de auto-adaptación tiene sentido o no en ambientes dinámicos.

La revisión también reveló que las investigaciones teóricas necesitan más atención, debido a que solo existen tres trabajos de esta naturaleza y sus resultados aunque importantes, tienen un alcance limitado a determinados paradigmas y problemas.

Tabla 5.26 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones y desviación estándar) de los algoritmos mSQJDE, mSspDE, mQSspDE, y mSQSspDE, en instancias del generador GDBG con funciones F1–F2.

Func.	Cambio	mSQJDE	mSspDE	mQSspDE	mSQSspDE
F1(10)	T1	**7.99E−02(8.07E−01)**	8.80E+00(1.16E+01)	2.07E−01(1.36E+00)	1.06E−01(9.31E−01)
	T2	**9.47E−01(3.94E+00)**	5.39E+00(1.07E+01)	1.37E+00(4.66E+00)	1.01E+00(4.18E+00)
	T3	1.01E+00(3.82E+00)	4.94E+00(9.37E+00)	1.23E+00(4.30E+00)	**8.44E−01(3.20E+00)**
	T4	7.68E−02(4.09E−01)	4.91E+00(8.76E+00)	1.62E−01(5.31E−01)	**7.60E−02(5.19E−01)**
	T5	6.82E−01(2.26E+00)	1.94E+00(4.09E+00)	7.71E−01(2.39E+00)	6.41E−01(2.22E+00)
	T6	1.75E−01(8.40E−01)	3.09E+00(7.86E+00)	2.58E−01(1.02E+00)	1.76E−01(9.11E−01)
	T7	1.13E+00(4.13E+00)	2.64E+00(6.49E+00)	1.32E+00(4.48E+00)	**1.08E+00(4.13E+00)**
F1(50)	T1	2.62E+00(4.20E+00)	6.12E+00(7.24E+00)	3.03E+00(4.50E+00)	2.73E+00(4.14E+00)
	T2	2.26E+00(3.75E+00)	5.64E+00(8.04E+00)	2.35E+00(3.87E+00)	**2.10E+00(3.58E+00)**
	T3	2.10E+00(3.74E+00)	5.24E+00(7.47E+00)	2.38E+00(4.13E+00)	**1.98E+00(3.74E+00)**
	T4	3.66E+00(7.38E+00)	1.10E+01(1.41E+01)	3.59E+00(6.39E+00)	3.66E+00(6.51E+00)
	T5	**5.33E−01(1.04E+00)**	1.18E+00(2.22E+00)	6.97E−01(1.56E+00)	5.44E−01(1.11E+00)
	T6	1.27E+00(2.51E+00)	2.75E+00(5.81E+00)	**1.25E+00(2.16E+00)**	1.28E+00(2.61E+00)
	T7	1.77E+00(3.52E+00)	3.47E+00(5.38E+00)	2.02E+00(3.65E+00)	**1.77E+00(3.50E+00)**
F2	T1	**9.88E−01(3.82E+00)**	2.26E+01(3.20E+01)	1.65E+00(4.48E+00)	1.01E+00(4.07E+00)
	T2	5.07E+00(2.20E+01)	1.95E+02(2.21E+02)	3.08E+01(9.46E+01)	**4.39E+00(1.04E+01)**
	T3	**4.71E+00(1.67E+01)**	2.08E+02(2.24E+02)	3.81E+01(1.07E+02)	5.39E+00(2.06E+01)
	T4	4.18E−01(1.25E+00)	9.66E+00(2.00E+01)	8.89E−01(1.45E+00)	**3.09E−01(1.10E+00)**
	T5	**9.69E+00(2.87E+01)**	1.66E+02(1.85E+02)	2.08E+01(5.90E+01)	9.95E+00(3.29E+01)
	T6	6.00E−01(1.49E+00)	1.21E+01(4.19E+01)	1.38E+00(2.78E+00)	**5.72E−01(2.23E+00)**
	T7	1.63E+00(4.66E+00)	4.26E+01(1.10E+02)	4.64E+00(2.24E+01)	**1.50E+00(5.04E+00)**

Además, se pudo ver que la tendencia actual en esta área es la propuesta de *Trabajos avanzados*, esto es, algoritmos más sofisticados que incorporan enfoques específicos para lidiar con la dinámica del problema. Sin embargo, estas investigaciones son similares a los *Estudios básicos* pues en su mayoría aplican la auto-adaptación a nivel de paradigma. De manera que, hasta donde se conoce, no existen investigaciones que empleen modelos auto-adaptativos en los enfoques de adaptación dinámica.

Teniendo en cuenta la limitaciones encontradas en la revisión anterior, en la presente investigación se propuso una estrategia adaptativa para controlar la diversidad del algoritmo en tiempo de ejecución, en el contexto de la evolución diferencial multipoblacional. Concretamente, la estrategia propuesta estuvo orientada a controlar el enfoque de adaptación basado en la generación de individuos quantum (Blackwell y Branke, 2006). Los experimentos realizados sobre diversos escenarios dinámicos permitieron concluir que la auto-adaptación tiene sentido en ambientes dinámicos cuando es aplicada explícitamente en los enfoques de adaptación dinámica.

No obstante los resultados derivados de la estrategia propuesta, se realizó un estudio experimental para analizar el rol de la auto-adaptación en diversas partes del algoritmo. En

Tabla 5.27 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones y desviación estándar) de los algoritmos mQDE, mSQDE, mJDE, y mQJDE, en instancias del generador GDBG con funciones F3–F6.

Func.	Cambio	mQDE	mSQDE	mJDE	mQJDE
F3	T1	7.54E+02(8.40E+01)	**1.98E+01(1.12E+02)**	7.87E+02(2.48E+02)	8.39E+02(6.64E+01)
	T2	1.01E+03(9.27E+01)	9.86E+02(8.01E+01)	9.35E+02(3.63E+02)	1.10E+03(7.73E+01)
	T3	1.00E+03(1.11E+02)	9.79E+02(8.73E+01)	8.78E+02(3.32E+02)	1.10E+03(7.13E+01)
	T4	9.48E+02(2.56E+02)	6.01E+02(5.36E+02)	8.93E+02(4.46E+02)	1.02E+03(2.66E+02)
	T5	1.01E+03(9.70E+01)	9.58E+02(9.48E+01)	1.07E+03(2.91E+02)	1.08E+03(8.83E+01)
	T6	1.08E+03(2.89E+02)	8.60E+02(4.72E+02)	1.24E+03(4.18E+02)	1.18E+03(2.88E+02)
	T7	8.33E+02(1.90E+02)	8.61E+02(2.50E+02)	4.60E+02(3.47E+02)	9.72E+02(1.41E+02)
F4	T1	4.21E+00(4.76E+00)	9.41E−01(3.13E+00)	3.33E+02(1.40E+02)	6.33E+00(2.69E+01)
	T2	7.11E+01(1.55E+02)	6.83E+00(1.98E+01)	3.49E+02(2.52E+02)	2.12E+02(2.64E+02)
	T3	6.47E+01(1.46E+02)	6.84E+00(1.93E+01)	3.51E+02(2.55E+02)	2.04E+02(2.62E+02)
	T4	4.88E+00(5.45E+00)	6.02E−01(1.30E+00)	2.37E+02(2.46E+02)	5.72E+01(1.54E+02)
	T5	1.73E+02(2.03E+02)	1.75E+01(5.42E+01)	3.71E+02(1.81E+02)	2.56E+02(2.41E+02)
	T6	1.02E+01(2.33E+01)	1.21E+00(3.20E+00)	1.78E+02(2.36E+02)	1.05E+02(2.11E+02)
	T7	5.42E+01(1.21E+02)	3.08E+00(6.91E+00)	1.50E+02(1.83E+02)	1.67E+02(2.34E+02)
F5	T1	8.16E+00(4.83E+00)	**1.32E+00(2.56E+00)**	1.67E+03(4.01E+02)	6.37E+02(8.03E+02)
	T2	4.30E+01(1.75E+02)	2.97E+00(5.30E+00)	1.92E+03(3.06E+02)	1.92E+03(1.65E+02)
	T3	3.37E+01(1.22E+02)	2.76E+00(4.53E+00)	1.88E+03(3.87E+02)	1.90E+03(2.32E+02)
	T4	9.71E+01(3.57E+02)	**1.28E+00(2.25E+00)**	1.82E+03(3.09E+02)	1.49E+03(6.88E+02)
	T5	6.91E+02(8.54E+02)	2.13E+01(1.30E+02)	1.91E+03(3.01E+02)	1.92E+03(1.58E+02)
	T6	7.05E+02(8.82E+02)	**4.75E+00(1.81E+01)**	1.91E+03(3.49E+02)	1.91E+03(3.05E+02)
	T7	2.26E+01(4.50E+01)	2.71E+00(5.18E+00)	3.57E+01(6.72E+01)	8.48E+02(8.37E+02)
F6	T1	1.17E+01(2.13E+01)	**3.13E+00(5.46E+00)**	4.28E+02(2.49E+02)	2.20E+01(7.16E+01)
	T2	2.09E+02(3.24E+02)	**1.95E+01(6.39E+01)**	2.40E+02(2.61E+02)	7.48E+02(3.22E+02)
	T3	2.45E+02(3.39E+02)	**2.67E+01(8.91E+01)**	2.65E+02(2.73E+02)	7.37E+02(3.10E+02)
	T4	4.36E+01(1.25E+02)	**6.90E+00(9.65E+00)**	3.07E+02(3.00E+02)	4.04E+02(4.27E+02)
	T5	3.46E+02(3.60E+02)	**6.03E+01(1.64E+02)**	7.24E+02(5.12E+02)	8.78E+02(1.95E+02)
	T6	8.32E+01(2.09E+02)	**1.30E+01(5.59E+01)**	7.90E+02(6.10E+02)	7.79E+02(4.42E+02)
	T7	8.07E+01(1.73E+02)	**1.55E+01(3.92E+01)**	1.66E+02(2.06E+02)	4.68E+02(3.90E+02)

este sentido, se consideraron un total de 8 algoritmos obtenidos a partir de la combinación de distintos niveles de auto-adaptación en el paradigma y en el enfoque de adaptación dinámica. Los resultados en diversos escenarios dinámicos se resumen de la manera siguiente:

1. La auto-adaptación a nivel de paradigma no es suficiente en sentido general en los escenarios dinámicos considerados.
2. La auto-adaptación resultó más eficaz cuando es aplicada explícitamente en el enfoque de adaptación quantum, orientado a lidiar con la dinámica del problema.
3. Los algoritmos con auto-adaptación tanto a nivel de paradigma como de enfoque de adaptación resultaron superiores en rendimiento que las variantes determinísticas, y similares a la variante con auto-adaptación a nivel de enfoque.

Tabla 5.28 Error de la mejor solución antes del cambio (promedio de 20 ejecuciones y desviación estándar) de los algoritmos mSQJDE, mSspDE, mQSspDE, y mSQSspDE, en instancias del generador GDBG con funciones F3–F6.

Func.	Cambio	mSQJDE	mSspDE	mQSspDE	mSQSspDE
F3	T1	1.52E+02(2.70E+02)	2.50E+02(2.91E+02)	7.97E+02(5.98E+01)	6.54E+02(2.46E+02)
	T2	1.05E+03(6.95E+01)	**6.24E+02(3.59E+02)**	1.02E+03(6.77E+01)	1.07E+03(6.29E+01)
	T3	1.04E+03(7.02E+01)	**6.44E+02(3.47E+02)**	1.01E+03(6.49E+01)	1.06E+03(6.32E+01)
	T4	7.49E+02(5.04E+02)	**2.40E+02(3.43E+02)**	9.77E+02(2.73E+02)	9.62E+02(3.32E+02)
	T5	1.03E+03(7.01E+01)	**5.51E+02(3.56E+02)**	1.01E+03(7.61E+01)	1.05E+03(6.80E+01)
	T6	1.09E+03(3.08E+02)	**2.27E+02(3.25E+02)**	1.04E+03(2.69E+02)	1.12E+03(2.84E+02)
	T7	9.96E+02(1.59E+02)	**2.26E+02(3.11E+02)**	9.10E+02(2.03E+02)	1.02E+03(1.54E+02)
F4	T1	6.80E−01(2.50E+00)	1.19E+02(1.30E+02)	6.28E+00(1.98E+01)	**4.90E−01(2.07E+00)**
	T2	**6.17E+00(2.01E+01)**	3.00E+02(2.72E+02)	1.71E+02(2.45E+02)	7.36E+00(2.66E+01)
	T3	**6.06E+00(1.20E+01)**	3.05E+02(2.75E+02)	1.62E+02(2.40E+02)	7.77E+00(2.81E+01)
	T4	4.95E−01(1.39E+00)	6.75E+01(1.39E+02)	3.38E+01(1.04E+02)	**4.56E−01(1.42E+00)**
	T5	1.66E+01(4.92E+01)	3.02E+02(2.19E+02)	2.03E+02(2.29E+02)	**1.55E+01(4.72E+01)**
	T6	9.11E−01(2.73E+00)	4.66E+01(1.14E+02)	3.77E+01(1.14E+02)	**8.11E−01(2.56E+00)**
	T7	**2.99E+00(8.07E+00)**	9.15E+01(1.73E+02)	7.89E+01(1.62E+02)	3.06E+00(7.58E+00)
F5	T1	1.44E+00(3.33E+00)	9.86E+02(6.66E+02)	4.21E+02(6.68E+02)	1.35E+00(3.73E+00)
	T2	3.44E+00(6.76E+00)	2.12E+02(5.48E+02)	6.94E+02(8.21E+02)	**1.85E+00(4.53E+00)**
	T3	3.27E+00(6.13E+00)	2.04E+02(5.45E+02)	7.06E+02(8.18E+02)	**1.54E+00(3.79E+00)**
	T4	1.96E+00(4.96E+00)	7.54E+02(8.08E+02)	7.73E+02(8.26E+02)	1.48E+00(5.29E+00)
	T5	4.16E+01(2.24E+02)	5.22E+01(2.93E+02)	8.25E+02(8.82E+02)	**1.72E+01(1.32E+02)**
	T6	5.65E+01(2.97E+02)	4.37E+02(7.24E+02)	8.21E+02(8.50E+02)	4.30E+01(2.72E+02)
	T7	3.08E+00(6.83E+00)	4.52E+00(9.45E+00)	4.20E+02(6.92E+02)	**2.24E+00(5.97E+00)**
F6	T1	5.10E+00(8.20E+00)	2.96E+02(2.85E+02)	1.71E+01(5.70E+01)	5.84E+00(9.16E+00)
	T2	3.97E+01(1.31E+02)	7.99E+01(1.81E+02)	5.65E+02(3.85E+02)	5.54E+01(1.68E+02)
	T3	5.43E+01(1.62E+02)	1.02E+02(2.12E+02)	5.77E+02(3.77E+02)	7.84E+01(2.04E+02)
	T4	9.77E+00(1.50E+01)	1.57E+02(2.32E+02)	2.38E+02(3.61E+02)	1.11E+01(1.72E+01)
	T5	1.78E+02(3.02E+02)	1.10E+02(2.18E+02)	6.57E+02(3.21E+02)	2.19E+02(3.35E+02)
	T6	3.12E+01(1.31E+02)	9.45E+01(1.96E+02)	2.80E+02(3.80E+02)	3.05E+01(1.26E+02)
	T7	3.47E+01(1.14E+02)	4.99E+01(1.20E+02)	3.32E+02(3.78E+02)	4.29E+01(1.34E+02)

4. Los algoritmos con auto-adaptación aplicada al enfoque de adaptación son competitivos en relación a otros métodos de la literatura.

Finalmente nos gustaría destacar que esta es un área aún en desarrollo cuyo objetivo final es la obtención de algoritmos cada vez más inteligentes en escenarios dinámicos. Por tanto, se necesita más trabajo en esta dirección.

Parte III

Conclusiones

6 Conclusiones y trabajos futuros

6.1 Conclusiones

Esta investigación estuvo enfocada al estudio, diseño, y desarrollo de estrategias avanzadas en el contexto de la optimización dinámica mediante algoritmos evolutivos multipoblacionales. En ese sentido, los objetivos trazados fueron los siguientes:

1. Proponer un herramienta para gestionar la experimentación en ambientes dinámicos, a partir de una organización adecuada de las medidas de rendimiento en ambientes dinámicos, y la inclusión de pruebas estadísticas no paramétricas.
2. Proponer estrategias adaptativas que mejoren el rendimiento del enfoque PSO multi-enjambre en ambientes dinámicos.
3. Investigar si la auto-adaptación, como técnica de control de parámetros, tiene sentido en ambientes dinámicos.
4. Analizar el rol de la auto-adaptación en ambientes dinámicos cuando es aplicada en distintas partes del algoritmo.

En relación al objetivo 1, se desarrolló la herramienta DynOptLab sobre la tecnología Java. Esta herramienta está compuesta por dos parte fundamentales: 1) un framework orientado a objetos que permite la inclusión de problemas, algoritmos y medidas de rendimiento, y 2) una interfaz gráfica de usuario orientada a la gestión y ejecución de experimentos computacionales. En este contexto, se propuso un marco de trabajo para organizar las medidas de rendimientos en ambientes dinámicos basado en cuatro categorías: información del problema, información del algoritmo, tiempo de medición, y función de agregación. Este marco de trabajo fue aplicado para organizar los progresos actuales, y en el diseño de las medidas de rendimiento en el framework de DynOptLab. En relación a la interfaz gráfica se diseñó de manera que el investigador pudiera establecer de manera intuitiva, los factores objeto

de estudio (problemas y algoritmos) y las variables de respuesta (medidas de rendimiento). Asimismo, la interfaz gráfica de DynOptLab permite ejecutar los experimentos de manera paralela, y analizar los resultados a partir de: la visualización directa de los resultados, comparaciones entre algoritmos, y la aplicación pruebas estadísticas no paramétricas. Con el objetivo de verificar las funcionalidades de la herramienta propuesta, se diseñó un caso de estudio basado en una investigación de la literatura. Los resultados de esta prueba fueron satisfactorios y permiten concluir que la herramienta propuesta permite una gestión eficiente de la experimentación en ambientes dinámicos. Los resultados alcanzados en este objetivo 1 fueron presentados en el Seminario Internacional Programa de Doctorado en Soft Computing, organizado en la Universidad Central de las Villas, Cuba (Novoa-Hernández et al, 2009) y en el Congreso Internacional de Computación y Matemática (COMPUMAT 2013), (Novoa-Hernández et al, 2013c).

Con respecto al objetivo 2, se desarrolló una revisión a fondo de las propuestas actuales que aplican el paradigma PSO a problemas dinámicos de optimización. A partir de este estudio se pudo comprobar que las variantes multipoblaciones (multi-enjambre) se encuentran entre las más efectivas para lidiar con problemas dinámicos complejos. En ese sentido, se destaca el algoritmo mQSO propuesto por Blackwell y Branke (2006), el cual incluye múltiples enjambres y una estrategia para la generación de diversidad durante la ejecución denominada enfoque quantum. No obstante los éxitos demostrados por este algoritmo en escenarios dinámicos, en la presente investigación se propusieron dos estrategias adicionales para mejorar su rendimiento. Por un lado, se diseñó una estrategia de diversidad similar al enfoque quantum pero aplicada después de la ocurrencia de un cambio en el ambiente, y por otro, una regla de control difusa para decidir en tiempo de ejecución los enjambres que pueden contribuir al proceso de optimización. Los resultados de los experimentos computacionales demostraron los beneficios de las propuestas. En particular, la estrategia de diversidad resultó efectiva en problemas con funciones pico multimodales, mientras que la regla de control difusa en problemas con funciones pico unimodales. Los resultados de este objetivo 2 fueron publicados en, un capítulo de libro de la serie *Studies in computational intelligence* (Novoa-Hernández et al, 2010) y en un artículo de la revista internacional *Memetic Computing* (Novoa-Hernández et al, 2011).

Para cumplir con el objetivo 3 de la investigación, se desarrolló un estudio en profundidad de los progresos actuales relacionados con la aplicación de la auto-adaptación en ambientes dinámicos. A partir de esta revisión se organizaron los trabajos existentes de acuerdo al tipo y alcance de la investigación: trabajos básicos, trabajos teóricos, y trabajos avanzados. En el primer caso se encuentran principalmente las primeros trabajos, y se caracterizan por el estudio de paradigmas auto-adaptativos en ambientes dinámicos. En el segundo caso aparecen los trabajos que analizan teóricamente el comportamiento de mode-

los auto-adaptativos. Finalmente, en el tercer grupo se encuentran los trabajos más recientes, caracterizados por la inclusión de enfoques de adaptación dinámica en conjunto con algún modelo auto-adaptativo. De manera general, se observó que existen conclusiones divididas en relación al éxito o no de la auto-adaptación en ambientes dinámicos. Asimismo, que son prácticamente inexistentes las investigaciones que apliquen modelos auto-adaptativos directamente a los enfoques de adaptación dinámica.

En este contexto, se propuso una estrategia auto-adaptativa para la generación de diversidad durante la ejecución, con el objetivo de investigar si en este caso, la auto-adaptación tiene sentido en ambientes dinámicos. En esencia, la estrategia permite la auto-adaptación de uno de los parámetros del enfoque quantum de Blackwell y Branke (2006), y fue aplicada en algoritmos multipoblacionales basados en el paradigma Evolución Diferencial (DE). Los experimentos computacionales desarrollados permitieron identificar: 1) la mejor combinación de parámetros que definen a la estrategia, y 2) el alto nivel de competitividad de los algoritmos propuestos (comparables y en ocasiones superiores a los de literatura). A partir de estos resultados, la principal conclusión a la que se arribó fue que la auto-adaptación si tiene sentido en ambientes dinámicos. Esta contribución fue publicada en un artículo de la revista internacional *Soft computing – a fusion of foundations, methodologies and applications* (Novoa-Hernández et al, 2013a).

No obstante los resultados del objetivo anterior, resultó de interés analizar con más profundidad el rol de la auto-adaptación en ambientes dinámicos. Este aspecto fue desarrollado a través del objetivo 4. En este sentido, se diseñaron experimentos computacionales orientados a estudiar el rendimiento de 8 algoritmos multipoblacionales con paradigmas computacionales basados en DE. Estos algoritmos se caracterizaron por la inclusión o no de distintos modelos auto-adaptativos tanto a nivel de paradigma como de enfoque de adaptación dinámica. Los resultados indicaron que la auto-adaptación posee un mayor impacto cuando es aplicada al enfoque de adaptación dinámica considerado. Asimismo, los algoritmos con presencia de modelos auto-adaptativos en ambos niveles resultaron superiores a los que solo aplican auto-adaptación solo a nivel de paradigma. Adicionalmente, con el objetivo de entender mucho mejor estas conclusiones, se propuso una medida que permite cuantificar el grado de adaptabilidad de los algoritmos. Finalmente, la competitividad de los mejores algoritmos estudiados fue analizada en relación a otros de la literatura. En relación a este aspecto, se pudo concluir que nuestros algoritmos alcanzan en general, niveles similares de rendimiento con respecto al resto. Los resultados de este estudio experimental fueron publicados en el Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'13) (Novoa-Hernández et al, 2012; Novoa-Hernández et al, 2013b).

6.2 Trabajos futuros

Como trabajos futuros nos proponemos los siguientes:

- Incorporar nuevos problemas, algoritmos y medidas de rendimiento de la literatura en la herramienta DynOptLab.
- Incorporar nuevas funcionalidades a la herramienta DynOptLab relacionadas con el análisis estadísticos y visualización de los resultados. Concretamente nos referimos a pruebas post-hoc y a la visualización de las medidas de rendimiento en forma de series temporales.
- Aplicar nuevos modelos auto-adaptativos en otros enfoques de adaptación dinámica.
- Extender los resultados obtenidos en los algoritmos estudiados a otros paradigmas computacionales.
- Estudiar los algoritmos propuestos en escenarios dinámicos reales.

6.3 Publicaciones derivadas de la investigación

Durante el desarrollo de la presente investigación se obtuvieron los siguientes resultados:

Artículos en revistas:

- **Novoa-Hernández, P.**, Cruz C., Pelta, D.A. *Efficient multi-swarm pso algorithms for dynamic environments*. Memetic Computing, Volume 3, Number 3, 163–174, 2011.
- **Novoa-Hernández, P.**, Cruz C., Pelta, D.A. *Self-adaptive, multipopulation differential evolution in dynamic environments.* Soft computing – a fusion of foundations, methodologies and applications. DOI 10.1007/s00500-013-1022-x .2013

Capítulos de libro:

- **Novoa, P.**, Pelta, D., Cruz, C., del Amo, I. *Controlling Particle Trajectories in a Multi-swarm Approach for Dynamic Optimization*. Problems Methods and Models in Artificial and Natural Computation. A Homage to Professor Mira's Scientific Legacy, Springer, Berlin / Heidelberg, 5601, pp 285–294, 2009.
- **Novoa-Hernández, P.**, , Pelta, D.A., Cruz C. *Improvement strategies for multi-swarm pso in dynamic environments*. Nature inspired cooperative strategies for optimization (NICSO 2010). Studies in computational intelligence, vol 284. Springer, Berlin, pp 371–383. 2010.

Trabajos enviados a congresos:

- **Novoa, P.**, Cruz C. *Control de fallos en PSO Multi-enjambre y su aplicación en ambientes dinámicos*. VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'09, Febrero, Málaga (artículo aceptado en la sesión de Metaheurísticas para Problemas Dinámicos de Optimización). 2009
- **Novoa, P.**, Pelta, D.A., Cruz, C. *DynOptLab: una herramienta para la experimentación en ambientes dinámicos*. Seminario Internacional Programa de Doctorado en Soft Computing. Universidad Central de las Villas, Abril de 2009.
- **Novoa, P.**, Pelta, D.A. *Control de fallos y PSO multi-enjambre en ambientes dinámicos*. IV Conferencia Científica Internacional, Universidad de Holguín, 15 de abril de 2009.
- **Novoa, P.**, Pelta, D.A., Cruz, C. *Una regla difusa en PSO multi-enjambre para ambientes dinámicos*. Segundo Taller de Descubrimiento de Conocimiento, Gestión del Conocimiento y Toma de Decisiones, Hotel Royal Decamerón Playa Blanca, Ciudad de Panamá, Panamá, noviembre de 2009.
- **Novoa-Hernández, P.**, Cruz C., Pelta, D.A. *Alcance de la evolución diferencial en ambientes dinámicos: un análisis empírico*. Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB 2012, 2012.
- **Novoa-Hernández, P.**, Cruz C., Pelta, D.A. *Sobre el rol de la auto-adaptación en ambientes dinámicos*. Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB 2013, 2013.
- **Novoa Hernández, P.**, Pelta, A. D, Cruz Corona, C., Novoa Hernández, M.A. *Medidas de rendimiento en ambientes dinámicos: revisión y una propuesta de clasificación*. Congreso Internacional Compumat'13. Universidad de Ciencias Informáticas, Cuba, 2013.

Otros trabajos como colaborador:

- García del Amo, I., Pelta, D., González, J., **Novoa, P.** *An Analysis of Particle Properties on a Multi-swarm PSO for Dynamic Optimization Problems*. Current Topics in Artificial Intelligence, Springer Berlin / Heidelberg, 5988, pp 32–41 2010.
- González, J.R., **Novoa-Hernández, P**., Cruz C., Castellanos, D., Pelta, D.A. Aplicación de la Soft Computing en el mundo real: entornos dinámicos. Compumat 2012.

Bibliografía

Ahmed T, Kamangar F (2005) Adaptive particle swarm optimizer: Response to dynamic systems through rank-based selection. In: FLAIRS Conference, pp 258–263

Alba E, Sarasola B (2010) Measuring fitness degradation in dynamic optimization problems. In: EvoApplications 2010, vol I, Springer-Verlag Berlin Heidelberg, pp 572–581

Alba E, Luque G, Arias D (2009) Impact of frequency and severity on non-stationary optimization problems. In: et al MG (ed) EvoWorkshops 2009, no. 5484 in Lecture Notes On Computer Science, Springer-Verlag Berlin Heidelberg, pp 755–761

Angeline P (1997) Tracking extrema in dynamic environments. In: Angeline PJ, Reynolds RG, McDonnell JR, Eberhart R (eds) Proceedings of the 6th International Conference on Evolutionary Programming, Springer Verlag, Indianapolis, Indiana, USA, vol 1213

Angeline PJ, Fogel DB, Fogel LJ (1996) A comparison of self-adaptation methods for finite state machines in dynamic environments. In: L J Fogel TB P J Angeline (ed) Evolutionary Programming V: Proceedings of the Fifth Annual Conf. on Evolutionary Programming, Cambridge, MA: MIT Press., pp 441–449

Arnold DV, Beyer HG (2002) Random dynamics optimum tracking with evolution strategies. In: Guervós JJM, Adamidis P, Beyer HG, nas JLFV, Schwefel HP (eds) Parallel problem solving from nature PPSN VII, Springer, Berlin, pp 3–12

Arnold DV, Beyer HG (2006) Optimum tracking with evolution strategies. Evolutionary Computation 14(3):291–308

Bäck T (1997) Self-adaptation. In: Bäck T, Fogel D, Michalewicz Z (eds) Handbook of Evolutionary Computation, Oxford University Press, New York

Bäck T (1998) On the behavior of evolutionary algorithms in dynamic environments. In: Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, pp 446–451, DOI 10.1109/ICEC.1998.699839

Bäck T (1999) Self-adaptive genetic algorithms for dynamic environments with slow dynamics. In: Wu A (ed) GECCO Workshops, pp 142–145

Bäck T, Schütz M (1996) Intelligent mutation rate control in canonical genetic algorithms. In: Foundation of intelligent systems, 9th international symposium, ISMIS '96, Springer, pp 158–167

Bäck T, Schwefel HP (1993) An overview of evolutionary algorithms for parameter optimization. Evol Comput 1:1–23

Banks A, Vincent J, Anyakoha C (2007) A review of particle swarm optimization. part i: background and development. Natural Computing: an international journal 6:467–848

Banks A, Vincent J, Anyakoha C (2008) A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Natural Computing: an international journal 7:109–124

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation: The New Experimentalism. Springer

Bartz-Beielstein T, Preuss M (2006) Considerations of budget allocation for sequential parameter optimization (spo). In: Proceedings of the Workshop on empirical methods for the analysis of algorithms, Reykjavik, Iceland, pp 35–40

Bartz-Beielstein T, Lasarczyk C, M P (2005) Sequential parameter optimization. In: McKay B ea (ed) Proceedings of the IEEE congress on evolutionary computation – CEC, IEEE Press, vol 1, pp 773–780

Beveridge GSG, Schechter RS (1977) Optimization: theory and practice. Chemical Engineering, McGraw-Hill Book Company

Beyer HG, Deb K (2001) On self-adaptive features in real-parameter evolutionary algorithms. Evolutionary Computation, IEEE Transactions on 5(3):250 –270

Bird S, Li X (2007) Informative performance metrics for dynamic optimization problems. In: 9th Conf. on Genetic and Evolutionary Computation, pp 18–25

Blackwell T (2003) Swarms in dynamic environments. In: Cantu-Paz E (ed) Genetic and Evolutionary Computation Conference, Springer, vol 2723 of LNCS, pp 1–12

Blackwell T (2005) Particle swarms and population diversity. Soft Computing - A Fusion of Foundations, Methodologies and Applications 9:793–802

Blackwell T, Bentley P (2002a) Don't push me! collision-avoiding swarms. In: CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress, IEEE Computer Society, Washington, DC, USA, pp 1691–1696

Blackwell T, Bentley PJ (2002b) Dynamic search with charged swarms. In: GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 19–26

Blackwell T, Branke J (2004) Multi-swarm optimization in dynamic environments. In: Raidl G (ed) Applications of Evolutionary Computing, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, vol 3005, pp 489–500

Blackwell T, Branke J (2006) Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Transactions on Evolutionary Computation 10(4):459–472

Blum C, Li X (2008) Swarm intelligence in optimization. pp 43–85, DOI 10.1007/978-3-540-74089-6_2

Boumaza A (2005) Learning environment dynamics from self-adaptation – a preliminary investigation. In: GECCO'05, Washington, DC, USA.

Branke J (1999a) Evolutionary approaches to dynamic optimization problems - a survey -. pp 134–137

Branke J (1999b) Memory enhanced evolutionary algorithms for changing optimization problemscorc. In: Angeline PJ, Michalewicz Z, Schoenauer M, Yao X, Zalzala A (eds) Proceedings of the Congress on Evolutionary Computation, IEEE Press, Mayflower Hotel, Washington D.C., USA, vol 3, pp 1875–1882

Branke J (2002) Evolutionary Optimization in Dynamic Environments, vol 3. Kluwer Academic Publishers, Boston, MA

Branke J, Kaußler T, Schmidt C, Schmeck H (2000) A multi-population approach to dynamic optimization problems. Adaptive Computing in Design and Manufacturing pp 299–308

Brest J, Greiner S, Boskovic M B Mernik, Zumer V (2006) Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. Transactions on Evolutionary Computation 10(6):646–657

Brest J, Boskovic B, Greiner S, Zumer V, Maucec M (2007) Performance comparison of self-adaptive and adaptive differential evolution algorithms. Soft Computing - A Fusion of Foundations, Methodologies and Applications 11:617–629

Brest J, Zamuda A, Boskovic B, Maucec MS, Zumer V (2009) Dynamic optimization using self-adaptive differential evolution. In: CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, pp 415–422

Broyden C (1970) The convergence of a class of double-rank minimization algorithms. i: General considerations. J Inst Math 6:76–90

Carlisle A, Dozier G (2000) Adapting particle swarm optimization to dynamic environments. In: In Proc. the International Conference on Artificial Intelligence (ICAI2000), pp 429–434

Carlisle A, Dozier G (2001) Tracking changing extrema with particle swarm optimizer. Tech. rep., Auburn University Technical Report CSSE01-08.

Carlisle A, Dozier G (2002) Tracking changing extrema with adaptive particle swarm optimizer. In: ISSCI, 2002 World Automation Congress, Orlando t USA

Charon I, Hudry O (1993) The noising method: A new method for combinatorial optimization. Operations Research Letters 14:133–137

Chun-Kit A, Ho-Fung L (2012) An empirical comparison of cma-es in dynamic environments. In: Coello Coello Cea (ed) PPSN 2012, Part I, LNCS 7491, Springer-Verlag Berlin Heidelberg, pp 529–538

Clerc M (2006) Particle swarm optimization. ISTE, Londres, Reino Unido (INGLATERRA), DOI 978-1-905209-04-0

Clerc M, Kennedy J (2002) The particle swarm - explosion, stability, and convergence in a multidimensional complex space. Evolutionary Computation, IEEE Transactions on 6(1):58–73

Cobb HG (1990) An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. AIC-90-001, Naval Res. Lab., Washington, DC

Coello Coello C (2000) Use of a self-adaptive penalty approach for engineering optimization problems. Computers in Industry 41(2):113–127

Cruz C, González JR, Pelta D (2011) Optimization in dynamic environments: a survey on problems, methods and measures. Soft Computing 15(7):1427–1448

Cui X, Potok T (2007) Distributed adaptive particle swarm optimizer in dynamic environment. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pp 1–7, DOI 10.1109/IPDPS.2007.370434

Dantzig GB (1951) Maximization of a linear function of variables subject to linear inequalities. In: Activity Analysis of Production and Allocation, Wiley

Dantzig GB (1963) Linear Programming and Extensions. Princeton University Press, Princeton, NJ

Das S, Suganthan PN (2011) Differential evolution – a survey of the state-of-the-art. IEEE Transactions on Evolutionary Computation 15(1):4–31

Dasgupta D, Mcgregor D (1992) Nonstationary function optimization using the structured genetic algorithm. In: Parallel Problem Solving From Nature, Elsevier, pp 145–154

Davidon W (1991) Variable metric methods for minimization. SIAM Journal Optimization 1:1–17

De Jong K (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, Ann Arbor, MI, USA

Deb K, Beyer H (2001) Self-adaptive genetic algorithms with simulated binary crossover. Evolutionary computation 9(2):197–221

Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7:1–30

Dong D, Jie J, Zeng J, Wang M (2008) Chaos-mutation-based particle swarm optimizer for dynamic environment. In: Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on, vol 1, pp 1032–1037, DOI 10.1109/ISKE.2008.4731081

Dorigo M (1992) Optimization, learning and natural algorithms (in italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy

Dorigo M, Maniezzo V, Colorni A (1996) The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B 26:29–41r

Dorigo M, Bonabeau E, Theraulaz G (2000) Ant algorithms and stigmergy. Future Gener Comput Syst 16(9):851–871

Du W, Li B (2008) Multi-strategy ensemble particle swarm optimization for dynamic optimization. Information Sciences 178(15):3096 – 3109, DOI 10.1016/j.ins.2008.01.020, nature Inspired Problem-Solving

Duech G, Scheuer T (1990) Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 90:161–175

Dueck G (1993) New optimization heuristics: The great deluge algorithm and the record-to- record travel. Journal of Computational Physics 104(1):86–92

Duhain J, Engelbrecht A (2012) Towards a more complete classification system for dynamically changing environments. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 2742–2749

Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science MHS95, IEEE Press, pp 39–43

Eberhart R, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimization. vol 1, pp 84–88 vol.1

Eberhart R, Shi Y (2001) Tracking and optimizing dynamic systems with particle swarms. In: Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, vol 1, pp 94–100 vol. 1

Edmonds J (1971) Matroids and the greedy algorithm. Mathematical Programming 1:127–136

Eiben A, Michalewicz Z, Schoenauer M, Smith J (2007) Parameter control in evolutionary algorithms. In: Lobo F, Lima C, Michalewicz Z (eds) Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, vol 54, Springer Berlin / Heidelberg, pp 19–46

Eriksson R, Olsson B (2002) On the behavior of evolutionary global-local hybrids with dynamic fitness functions. In: Parallel Problem Solving from Nature — PPSN VII, Springer, Springer, Granada

Eriksson R, Olsson B (2004) On the performance of evolutionary algorithms with life-time adaptation in dynamic fitness landscapes. In: Congress on Evolutionary Computation, CEC2004

Esquivel S, Coello Coello C (2004) Particle swarm optimization in non-stationary environments. In: Advances in Artificial Intelligence- IBERAMIA 2004, Springer-Verlag, Springer-Verlag, DOI 10.1007/b102591

Farmer JD, Packard N, Perelson A (1986) The immune system, adaptation and machine learning. Physica D 2:187–204

Fayad M, Schmidt D, Johnson R (1999) Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Wiley & Sons; 1 edition, cap Application Frameworks, p 638

Feng W, Brune T, Chan L, Chowdhury M, Kuek CK, Li Y (1997) Benchmarks for testing evolutionary algorithms (tech. rep. no. csc-97006). Tech. rep., Glasgow, UK: Center for System and Control, University of Glasgow.

Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8:67–71

Fernandes C, Lima C, Rosa A (2008) Umdas for dynamic optimization problems. In: Proceedings of the genetic and evolutionary computation conference, ACM, New York, pp 399–406

Fernández M (2004) Diseño de experimentos. digital, departamento de Matemática Aplicada. 2004, Instituto Superior Politécnico José Antonio Echeverría: Cuba.

Fogarty TC (1989) Varying the probability of mutation in the genetic algorithm. In: Inc MKP (ed) Proceedings of the 3rd international conference on genetic algorithms, San Francisco, pp 104–109

Fogel DB, Fogel LJ, Atmar JW (1991) Meta-evolutionary programming. In: Chen R (ed) Proc. of the 25th Asilomar Conference on Signals, Systems and Computers, Maple Press, San Jose, CA, pp 540–545

Fogel LJ (1962) Toward inductive inference automata. In: Proceedings of the International Federation for Information Processing Congress, Munich, pp 395–399

Olivetti de França F, Zuben FJV (2009) A dynamic artificial immune algorithm applied to challenging benchmarking problems. In: CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, pp 423–430

García S, Molina D, Lozano M, 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. J Heuristics 15:617–644

García Del Amo I, Pelta D, González J, Novoa P (2010) An analysis of particle properties on a multi-swarm pso for dynamic optimization problems. vol 5988 LNAI, pp 32–41

Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: Harmony search. Simulation 2(76):60–68

Glover F (1977) Heuristics for integer programming using surrogate constraints. Decision Sciences 8:156–166

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13(5):533–549

Glover F, McMillan C (1986) The general employee scheduling problem: An integration of ms and ai. Computers and Operations Research 13(5):563–573,

Goldberg DE, Smith RE (1987) Nonstationary function optimization using genetic algorithm with dominance and diploidy. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp 59–68

Grefenstette J (1989) Optimization of control parameters for genetic algorithms. IEEE Trans Syst Man Cybernetics 16:122–128

Grefenstette J (1999) Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In: Proceedings of the 1999 Congress on Evolutionary Computation., Piscataway, NJ: IEEE Service Center., vol 3

Grefenstette JJ (1992) Genetic algorithms for changing environments. In: Maenner R, Manderick B (eds) Parallel Problem Solving from Nature

Griva I, Nash SG, Sofer A (2009) Linear and nonlinear optimization, 2do edn. Society for Industrial and Applied Mathematics

Halder U, Maity D, Dasgupta P, Das S (2011) Self-adaptive cluster-based differential evolution with an external archive for dynamic optimization problems. In: Panigrahi B, et al (eds) SEMCCO 2011, LNCS, vol Part I, Springer-Verlag Berlin Heidelberg, pp 19–26

Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proc. of the 1996 IEEE Int. Conf. on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, pp 312–317

Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2):159–195

Herdy M (1992) Reproductive isolation as strategy parameter in hierarchically organized evolution strategies. In: Proceedings of the 10th conference on parallel problem solving from nature – PPSN II, pp 207–217

Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42(1):228–234

Holland JH (1962) Outline for a logical theory of adaptive systems. Journal of the ACM 3:297–314

Hu J, Zeng J, Tan Y (2007) Bio-Inspired Computational Intelligence and Applications, Springer, cap A Diversity-Guided Particle Swarm Optimizer for Dynamic Environments, pp 239–247

Hu X, Eberhart R (2002) Adaptive particle swarm optimization: Detection and response to dynamic systems. In: In Proceedings of the IEEE Congress on Evolutionary Computation, CEC2002. IEEE, Press, pp 1666–1670

Hui S, Suganthan PN (2012) Ensemble differential evolution with dynamic subpopulations and adaptive clearing for solving dynamic optimization problems. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 2629–2636

Igel C, Toussaint M (2003) On classes of functions for which no free lunch results hold. Information Processing Letters 86(6):317–321

Igel C, Suttorp T, Hansen N (2006) A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO 2006, pp 453–460

Janson S, Middendorf M (2003) A hierarchical particle swarm optimizer. In: Congress on Evolutionary Computation (CEC-2003), IEEE Press, pp 770–776

Janson S, Middendorf M (2004) A hierarchical particle swarm optimizer for dynamic optimization problems. In: Applications of Evolutionary Computing, Springer Berlin / Heidelberg, Springer Berlin / Heidelberg, Lecture Notes in Computer Science, vol 3005, p 513–524, DOI 10.1007/b96500

Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. Evolutionary Computation, IEEE Transactions on 9(3):303–317

Joines J, Houck C (1994) On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In: DB F (ed) Proceedings of the 1st IEEE conference on evolutionary computation, IEEE, Orlando, Florida, pp 579–584

Karaman A, Uyar S, Eryigit G (2005) The memory indexing evolutionary algorithm for dynamic environments. In: Applications on evolutionary computing. Lecture notes in computer science, vol 3449, Springer, Berlin, pp 563–573

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, vol 4, pp 1942–1948 vol.4

Kennedy J, Eberhart R (2001) Swarm intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on, vol 2, pp 1671–1676

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science (4598):671–680

Köppen M, Wolpert DH, Macready WG (2001) Remarks on a recent paper on the no free lunch theorems. IEEE Transactions on Evolutionary Computation 5(3):295–296

Korosec P, Silc J (2009) The differential ant-stigmergy algorithm applied to dynamic optimization problems. In: CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, pp 407–414

Korosec P, Silc J (2012) The continuous differential ant-stigmergy algorithm applied to dynamic optimization problems. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 447–454

Koza JR (1992) Genetic Programming. MIT Press, Cambridge, MA

Kramer O (2010) Evolutionary self-adaptation: A survey of operators and strategy parameters. Evolutionary Intelligence 3(2):51–65

de Landgraaf W, Eiben A, Nannen V (2007) Parameter calibration using meta-algorithms. In: Proceedings of the IEEE congress on evolutionary computation – CEC, pp 71–78

Lepagnot J, Nakib A, Oulhadj H, Siarry P (2012) A dynamic multi-agent algorithm applied to challenging benchmark problems. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 2621–2628

Lewis R, Torczon V, Trosset M (2000) Direct search methods: Then and now. Journal of Computational and Applied Mathematics 124((1-2)):191–207

Li C, Yang S (2008a) Fast multi-swarm optimization for dynamic optimization problems. In: Fourth International Conference on Natural Computation, IEEE Computer Society

Li C, Yang S (2008b) A generalized approach to construct benchmark problems for dynamic optimization. In: Simulated Evolution and Learning, Springer Berlin / Heidelberg, Springer Berlin / Heidelberg, Lecture Notes in Computer Science, vol 5361, p 391–400

Li C, Yang S (2009) A clustering particle swarm optimizer for dynamic optimization. In: CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, pp 439–446

Li C, Liu Y, Zhou A, Kang L, Wang H (2007) Advances in Computation and Intelligence, cap A Fast Particle Swarm Optimization Algorithm with Cauchy Mutation and Natural Selection Strategy, pp 334–343

Li C, Yang S, Nguyen TT, Yu EL, Yao X, Jin Y, Beyer HG, Suganthan PN (2008) Benchmark generator for cec'2009 competition on dynamic optimization. Tech. rep., Department of Computer Science, University of Leicester, U.K.

Li C, Yang S, Yang M (2012) Maintaining diversity by clustering in dynamic environments. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 2734–2741

Li X, Dam K (2003) Comparing particle swarms for tracking extrema in dynamic environments. In: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, vol 3, pp 1772–1779 Vol.3

Liu L, Wang D, Yang S (2008) Compound particle swarm optimization in dynamic environments. In: In EvoWorkshops 2008: Applications of Evolutionary Computing, LNCS 4974, Springer-Verlag, Berlin, LNCS, pp 617–626

Martin O, Otto S, Felten EW (1991) Large-step markov chains for the traveling salesman problem. Complex Systems 5(3):299–326

Mattfeld D, Bierwirth C (2004) An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 155(3):616–630

Melián B, Moreno Pérez J, Moreno Vega J (2003) Metaheurísticas: Una visión global. Revista Iberoamericana de Inteligencia Artificial 19(2):7–28

Mendes R, Mohais AS (2005) Dynde: A differential evolution for dynamic optimization problems. In: Proc. IEEE Congr. Evol. Comput., vol 2, pp 2808–2815

Mendes R, Kennedy J, Neves J (2004) The fully informed particle swarm: simpler, maybe better. Evolutionary Computation, IEEE Transactions on 8(3):204–210

Meyer-Nieberg S, Beyer HG (2007) Self-adaptation in evolutionary algorithms. In: Lobo F, Lima C, Michalewicz Z (eds) Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, vol 54, Springer Berlin / Heidelberg, pp 19–46

Mezura-Montes E, Palomeque-Ortiz A (2009) Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. Studies in Computational Intelligence 198:95–120

Mladenovic N (1995) A variable neighborhood algorithm–a new metaheuristic for combinatorial optimization. In: Abstracts of Papers Presented at Optimization Days

Montgomery D (2000) Design and Analysis of Experiments. Wiley, 5 edition

Mori N, Kita H, Nishikawa Y (1996) Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In: Voigt HM (ed) Parallel Problem Solving from Nature, no. 1141 in LNCS, pp 513–522

Morrison R (2003) Performance measurement in dynamic environments. In: Branke J (ed) GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems

Morrison R, De Jong K (1999) A test problem generator for non-stationary environments. In: Proceedings of the 1999 Congress on Evolutionary Computation, vol 3, pp –2053

Morrison RW (2004) Designing Evolutionary Algorithms for Dynamic Environments. Springer

Moser I, Chiong R (2010) Dynamic function optimisation with hybridised extremal dynamics. Memetic Computing 2(2):137–148

Nelder J, Mead R (1964) A simplex method for function minimization. The Computer Journal 7:308–13

Neri F, Tirronen V (2010) Recent advances in differential evolution: a survey and experimental analysis. Artif Intell Rev 33:61–106

Nguyena TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation 6:1–24

Novoa P, Cruz C (2009) Control de fallos en pso multi-enjambre y su aplicación en ambientes dinámicos. In: VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'09, Málaga, Spain.

Novoa P, Pelta D, Cruz C, Garcia del Amo I (2009) Controlling particle trajectories in a multi-swarm approach for dynamic optimization problems. In: 3rd International Work-conference on the Interplay between Natural and Artificial Computation, IWINAC'09, Santiago de Compostela, Spain

Novoa-Hernández P, Pelta D, Cruz C (2009) Dynoptlab: una herramienta para la experimentación en ambientes dinámicos. In: Seminario Internacional Programa de Doctorado en Soft Computing. Universidad Central de las Villas

Novoa-Hernández P, Pelta D, Corona C (2010) Improvement strategies for multi-swarm pso in dynamic environments. In: González J, Pelta D, Cruz C, Terrazas G, Krasnogor N (eds) Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), Studies in Computational Intelligence, vol 284, Springer Berlin / Heidelberg, pp 371–383

Novoa-Hernández P, Corona C, Pelta D (2011) Efficient multi-swarm pso algorithms for dynamic environments. Memetic Computing 3(3):163–174

Novoa-Hernández P, Pelta DA, Corona CC (2012) Alcance de la evolución diferencial en ambientes dinámicos: un análisis empırico. In: X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'12

Novoa-Hernández P, Cruz Corona C, Pelta DA (2013a) Self-adaptive, multipopulation differential evolution in dynamic environments. Soft Computing - A Fusion of Foundations, Methodologies and Applications DOI 10.1007/s00500-013-1022-x

Novoa-Hernández P, Cruz Corona C, Pelta DA (2013b) Sobre el rol de la auto-adaptación en ambientes dinámicos. In: X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'13, Málaga, Spain.

Novoa-Hernández P, Pelta DA, Cruz Corona C, Novoa-Hernández M (2013c) Medidas de rendimiento en ambientes dinámicos: revisión y una propuesta de clasificación. In: Congreso Internacional COMPUMAT 2013

Oltean M (2004) Searching for a practical evidence of the no free lunch theorems. In: Biologically Inspired Approaches to Advanced Information Technology, Revised Selected Papers of the First International Workshop. BioADIT 2004, volume 3141/2004, Lecture Notes in Computer Science (LNCS), Springer Berlin / Heidelberg

Omran MG, Engelbrecht AP, Salman A (2009) Bare bones differential evolution. European Journal of Operational Research 196(1):128–139

Pan G, Dou Q, Liu X (2006) Performance of two improved particle swarm optimization in dynamic optimization environments. In: ISDA '06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications, IEEE Computer Society, Washington, DC, USA, pp 1024–1028

Pan QK, Suganthan P, Wang L, Gao L, Mallipeddi R (2011) A differential evolution algorithm with self-adapting strategy and control parameters. Computers & Operations Research 38:394–408

Parrott D, Li X (2004) A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. vol 1, pp 98–103

Parsopoulos K, Vrahatis M (2001) Particle swarm optimizer in noisy and continuously changing environments. In: M.H. Hamza (Ed.), Arti cial Intelligence and Soft Computing, IASTED/ACTA, IASTED/ACTA Press, pp 289–294

Parsopoulos K, Vrahatis M (2004) Upso: A unified particle swarm optimization scheme. In: Lecture Series on Computer and Computational Sciences, Vol. 1, Proc. Int. Conf. Comput. Meth. Sci. Eng. (ICCMSE 2004), VSP International Science Publishers, Zeist, The Netherlands, pp 868–873

Parsopoulos K, Vrahatis M (2005) Unified particle swarm optimization in dynamic environments. In: Lecture Notes in Computer Science 3449, Springer Verlag, pp 590–599

Pelta D, Sancho-Royo A, Cruz C, Verdegay JL (2006) Using memory and fuzzy rules in a co-operative multi-thread strategy for optimization. Information Sciences 176(13):1849–1868

Pelta D, Cruz C, Verdegay J (2009) Simple control rules in a cooperative system for dynamic optimization problems. International Journal of General Systems 38(7):701–717

Peng B, Reynolds R (2004) Cultural algorithms: Knowledge learning in dynamic environments. In: Congress on Evolutionary Computation, CEC2004

Pettit E, Swigger K (1983) An analysis of genetic based pattern tracking and cognitive based component tracking models of adaptation. In: National Conference on AI, AAAI-83

du Plessis MC, Engelbrecht AP (2011) Self-adaptive competitive differential evolution for dynamic environments. In: IEEE Symposium on Differential Evolution - SDE , 2011

du Plessis MC, Engelbrecht AP (2012) Using competitive population evaluation in a differential evolution algorithm for dynamic environments. European Journal of Operational Research 218:7–20

Polya G (1945) How to Solve It. Princeton University Press, Princeton, NJ, 1945. Princeton University Press, Princeton, NJ.

Powell M (1964) An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal 7(2):155–162

Prat Bartés A, Tort-Martorell Llabrés X (2004) Métodos estadísticos. Control y mejora de la calidad. Ediciones UPC

Preuss M, Bartz-Beielstein T (2007) Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms. In: Parameter setting in evolutionary algorithms, Studies in computational intelligence, Springer, pp 91–119

Price K (1994) Genetic annealing. Dr Dobb's Journal pp 127–132

Rao SS (2009) Engineering optimization : theory and practice, 4to edn. John Wiley & Sons, Inc.

Rechenberg I (1965) Cybernetic solution path of an experimental problem. technical report no. 1112. Tech. rep., Royal Aircraft Establishment Library Translation Farnborough, UK

Rechenberg I (1973) Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. In: Frommann-Holzboog, Stuttgart

Reynolds RG (1994) An introduction to cultural algorithms. In: Sebald AV, Fogel LJ (eds) 3rd Annual Conference on Evolutionary Programming, World Scientific, River Edge, NJ, pp 131–139

Richter H (2009) Detecting change in dynamic fitness landscapes. In: IEEE Congress on Evolutionary Computation, CEC, 2009, pp 1613–1620

Ros R, Hansen N (2008) A simple modification in cma-es achieving linear time and space complexity. In: Rudolph G, Jansen T, Lucas S, Poloni C, Beume N (eds) PPSN 2008. LNCS, Springer, Heidelberg, vol 5199, pp 296–305

Rosenbrock H (1960) An automatic method for finding the greatest or least value of a function. The Computer Journal 3(3):175–184

Saleem S, Reynolds R (2000) Cultural algorithms in dynamic environments. In: Proceedings of the Congress on evolutionary computation, vol 2, pp 1513–1520

Salman EAOM A (2007) Empirical analysis of self-adaptive differential evolution. European Journal of Operational Research 183(2):785–804

Salomon R, Eggenberger P (1997) Adaptation on the evolutionary time scale: A working hypothesis and basic experiments. In: Hao JK, Lutton E, Ronald E, Schoenauer M, Snyders D (eds) Artificial Evolution: Third European Conf., AE'97, Springer, Berlin, pp 251–262

Schaffer J, Caruana R, Eshelman L, Das R (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization. In: Proceedings of the 3rd international conference on genetic algorithms – ICGA 1989., pp 51–60

Schoeman I, Engelbrecht A (2006) Simulated Evolution and Learning, Lecture Notes in Computer Science, vol Volume 4247/2006, Springer Verlag, cap Niching for Dynamic Environments Using Particle Swarm Optimization, pp 134–141

Schönemann L (2007) Evolution strategies in dynamic environments. Studies in Computational Intelligence (SCI) 51:51–77

Schwefel HP (1981) Numerical Optimization of Computer Models. John Wiley, Chichester, UK

Schwefel HP (1995) Evolution and optimum seeking. In: Sixth-Generation Computer Technology, Wiley Interscience, New York

Sharifi A, Noroozi V, Bashiri M, Hashemi AB, Meybodi MR (2012) Two phased cellular PSO: A new collaborative cellular algorithm for optimization in dynamic environments. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 350–357

Shilane D, Martikainen J, Ovaska SJ (2009) Time-dependent performance comparison of evolutionary algorithms. In: et al MK (ed) ICANNGA 2009, no. 5495 in Lecture Notes On Computer Science, Springer-Verlag Berlin Heidelberg, pp 223–232

Smith J (2008) Self-adaptation in evolutionary algorithms for combinatorial optimisation. Studies in Computational Intelligence 136:31–57

Stanhope SA, Daida JM (1999) (1 + 1) genetic algorithm fitness dynamics in a changing environment. In: 1999 Congress on Evolutionary Computation, Piscataway, NJ: IEEE Service Center., pp 1851–1858

Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11:341–359

ghazali Talbi E (2009) Metaheuristics : from design to implementation. John Wiley & Sons

Tinos R, Yang S (2008) Evolutionary programming with q-gaussian mutation for dynamic optimization problems. In: Proceedings of the IEEE Congress on evolutionary computation, pp 1823–1830

Trojanowski K, Michalewicz Z (1999) Searching for optima in non-stationary environments. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol 3, p (2348), DOI 10.1109/CEC.1999.785498

Trojanowski K, Wierzchon ST (2009) Immune-based algorithms for dynamic optimization. Information Sciences 179(10):1495 – 1515, DOI 10.1016/j.ins.2008.11.014, including Special Issue on Artificial Imune Systems

Verdegay J, Yager RR, Bonissone PP (2008) On heuristics as a fundamental constituent of soft computing. Fuzzy Sets and Systems 159:846–855

Voudouris C, Tsang E (1995) Guided local search. technical report csm-247. Tech. rep., University of Essex, UK

Wan S, Xiong S, Liu Y (2012) Prediction based multi-strategy differential evolution algorithm for dynamic environments. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 640–647

Wang H, Wang D, Yang S (2007) Triggered memory-based swarm optimization in dynamic environments. In: Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog, Springer-Verlag, Berlin, Heidelberg, pp 637–646

Weicker K (2002) Performance measures for dynamic environments. In: Guervós JJM, Adamidis P, Beyer HG, nas JLFV, Schwefel HP (eds) Parallel problem solving from nature, Berlin, Springer, vol VII, pp 64 – 73

Weicker K (2003) Evolutionary algorithms and dynamic optimization problems. PhD thesis, Institut für Formale Methoden der Informatik der Universität Stuttgart

Weicker K, Weicker N (1999) On evolution strategy optimization in dynamic environments. In: Congress on Evolutionary Computation, vol 3, pp 2039–2046

Wolpert D, Macready W (1995) No free lunch theorems for search. JPL pp 1–38

Wolpert D, Macready W (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1):67–82

Xiao L, Zuo X (2012) Multi-DEPSO—-a DE and PSO based hybrid algorithm in dynamic environments. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 1601–1607

Yang S (2003) Non-stationary problems optimization using the primal-dual genetic algorithm. In: Proc. Congr. Evol. Comput., vol 3, pp 2246–253

Yang S (2005) Memory-based immigrants for genetic algorithms in dynamic environments. In: ACM (ed) Proceedings of the genetic and evolutionary computation conference, New York, pp 1115–1122

Yang S (2006) Associative memory scheme for genetic algorithms in dynamic environments. In: Applications of evolutionary computing. Lecture notes in computer science, vol 3907, Springer, Berlin., pp 788–799

Yang S, Cheng H, Wang F (2010) Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. IEEE Trans Syst Man Cybernet C: Appl Rev 40(99):52–63

Yao X, Liu Y (1996) Fast evolutionary programming. In: Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press, pp 451–460

Yazdani D, Akbarzadeh-Totonchi MR, Nasiri B, Meybodi MR (2012) A new artificial fish swarm algorithm for dynamic optimization problems. In: Li X (ed) Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, pp 1538–1545

Yonezawa Y, Kikuchi T (1996) Ecological algorithm for optimal ordering used by collective honey bee behavior. In: 7th International Symposium on Micro Machine and Human Science, pp 249–256

Yu EL, Suganthan PN (2009) Evolutionary programming with ensemble of explicit memories for dynamic optimization. In: CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, pp 431–438

Zadeh LA (1994) Fuzzy logic and soft computing: issues, contentions and perspectives. In: Proc. IIZUKA'94: 3rd Internat. Conf. on Fuzzy Logic, Neural Nets and Soft Computing, Iizuka, Japan, pp 1–2

Zou X, Wang M, Zhou A, Mckay B (2004) Evolutionary optimization based on chaotic sequence in dynamic environments. In: IEEE International Conference on Networking, Sensing and Control