UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación
e Inteligencia Artificial

Soft Computing en Problemas de Optimización Dinámicos

MEMORIA QUE PRESENTA

Jenny Fajardo Calderín

PARA OPTAR AL GRADO DE DOCTOR EN INFORMÁTICA

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

DIRECTORES:

David Alejandro Pelta
Antonio David Masegosa Arredondo

Editor: Universidad de Granada. Tesis Doctorales
Autora: Jenny Fajardo Calderín
ISBN: 978-84-9125-477-5
URI: http://hdl.handle.net/10481/42206

Agradecimientos

En las siguientes líneas agradezco a todas las personas e instituciones que han hecho posible esta tesis.

Dedico este trabajo a mi familia, que son mi razón de ser. A mis padres por estar siempre apoyándome incondicionalmente, por confiar en mí. Mami gracias por ser tan fuerte ante todas las cosas que hemos vivido en los últimos años, y papi aunque no estés este trabajo también es tuyo. A mis cuatro hermanos por todo su cariño, Tammy-Erne y Andy-Damiana, gracias por sus consejos y estar siempre cuando los necesito, sin ustedes nunca hubiera llegado hasta aquí. A mis dos niños, Eimy y Andysito, por hacerme tan feliz. En especial a Damián, porque siempre me impulsó a seguir adelante con este trabajo, por todo su amor y cariño, en muchos momentos difíciles tus palabras siempre me daban tranquilidad, gracias por estar ahí. Los amo mucho.

Agradezco a mis directores de tesis, David y Antonio, por darme la oportunidad de trabajar con ustedes, muchas gracias por todo su apoyo, por motivarme a seguir adelante, por todos sus consejos y conocimientos que han compartido conmigo. Les agradezco por haber dedicado tanto tiempo a las revisiones que tanto me han enseñado.

Agradezco a la Asociación Universitaria Iberoamericana de Estudios de Posgrado (AUIP) por auspiciar el programa de doctorado, así como la posibilidad que me brindó de realizar estancias de investigación en la Universidad de Granada, en los años 2012, 2013 y 2014. Agradecer al programa EurekaSD de Erasmus Mundus, porque me brindó la posibilidad de realizar una estancia de investigación en la Universidad de Granada en el año 2015.

Muchas gracias a los miembros del grupo de trabajo Modelos de Decisión y Optimización (MODO) del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, muchas gracias, Maite, Curro, Carlos, Blanca, Virgilio y Pablo, por todo el apoyo, por recibirme como un miembro más del grupo, y siempre estar dispuestos a ayudarme.

A mi compañero y amigo, Alejandro Rosete, gracias por todo su apoyo, por sus consejos, por guiarme e impulsarme en la investigación.

A mi familia del casino deportivo, por acogerme como parte de la familia, gracias por

todo su apoyado, preocupación y poder siempre contar con ustedes.

A mis amigos, muchas gracias porque siempre he podido contar con ustedes, porque tantos momentos alegres en todo este tiempo me animaron a seguir adelante con la tesis, sin dudas ustedes son lo máximo!!!.

A los amigos de Granada y a las niñas de Málaga, agradecerles toda su ayuda y apoyo, gracias a ustedes mis estancias han sido más agradables y se ha aliviado mi nostalgia. Me han hecho sentir como en casa y se han convertido en parte de mi familia.

A la Facultad de Informática de la CUJAE y a todos mis compañeros del departamento DIAISI y del grupo de investigación, muchas gracias por darme la oportunidad de formar parte de este programa de doctorado, y por toda su ayuda. Agradezco a Orestes Llanes, Vicerrector de Investigaciones de la CUJAE, por motivarnos a participar en este programa de doctorado y porque durante todo este tiempo siempre ha estado pendiente de los avances de esta investigación.

MUCHAS GRACIAS A TODOS!!!

Resumen

La presente investigación se centra en el estudio, diseño y evaluación de esquemas de portafolio basados en metaheurísticas para abordar problemas de optimización dinámicos de tipo combinatorio. Se tuvieron en cuenta las ideas de adaptación, cooperación y aprendizaje que consideramos indispensables en este contexto. En ese sentido los objetivos planteados fueron:

- Realizar un estudio en profundidad en el ámbito de la Soft Computing, de cara a identificar las técnicas que se utilizan para resolver problemas de optimización dinámicos, incluyendo las diferentes posibilidades de hibridación de técnicas.
- Diseñar un portafolio de algoritmos que integre técnicas de adaptación, cooperación y aprendizaje, para resolver problemas de optimización dinámicos.
- Validar el funcionamiento del portafolio de algoritmos con respecto a algoritmos del estado del arte sobre problemas de optimización dinámicos combinatorios de prueba y reales.
- Proponer una librería de software para facilitar la reutilización y aplicación de metaheurísticas en la resolución de problemas de optimización estacionarios y dinámicos.

Para alcanzar estos objetivos se propusieron, por un lado un portafolio de algoritmos que incorpora mecanismos de aprendizaje, y por otro, una batería de experimentos sobre problemas dinámicos de optimización, para validar su funcionamiento.

La primera contribución de la tesis consiste en un portafolio de algoritmos, que dispone de un conjunto de metaheurísticas. En cada iteración selecciona cuál de ellas aplicar mediante un enfoque basado en créditos que actúa como un esquema de aprendizaje. Los experimentos se realizaron sobre cinco problemas "artificiales" a los que se les indujo dinamismo mediante el operador XOR, para diferentes escenarios de severidad y frecuencia de cambio. Los objetivos de la experimentación fueron evaluar el rendimiento del portafolio con diferentes esquemas de aprendizaje; evaluar el rendimiento de los algoritmos individuales que componen el portafolio, frente al propio portafolio; y finalmente, comparar la mejor

variante de aprendizaje del portafolio con dos algoritmos de la literatura que han mostrado un muy buen rendimiento en estos problemas. Los resultados del método propuesto fueron significativamente mejores en la mayoría de los problemas, lo que demuestra que el enfoque del portafolio de algoritmos con un esquema de aprendizaje simple, proporciona buenos resultados para resolver PODs.

Por otro lado, se aplicó el portafolio al problema de máxima cobertura dinámico (DMCLP). Dicho problema tiene como objetivo encontrar la ubicación de instalaciones que maximiza la demanda cubierta, es decir, la demanda que se encuentra dentro de un determinado tiempo o distancia de cobertura. Partiendo de la variante clásica de DMCLP se definieron dos variantes nuevas: DMCLP-AC y DMCLP-LocF, las cuales consisten en incorporar costos por apertura/cierre de instalaciones de un periodo a otro y en fijar la ubicación de las instalaciones a lo largo del tiempo, respectivamente.

Para la resolución de estos problemas se empleó un portafolio compuesto por tres métodos de Recocido Simulado propuestos en la literatura, y que habían reportado muy buenos resultados. En la experimentación, se comparó el portafolio con los algoritmos individuales que lo componen y de forma general, se pudo concluir que el portafolio fue mejor que los métodos individuales en cada una de las instancias de prueba utilizadas. Cabe destacar que los resultados del portafolio de algoritmos para las tres variantes del DMCLP muestran que este tipo de esquemas son capaces de crear sinergias entre los métodos que lo componen.

Por último, todos los algoritmos, métodos y problemas utilizados durante el desarrollo de la tesis se incorporaron en una librería de software llamada BiCIAM, que incluye diferentes algoritmos para resolver problemas de optimización tanto mono-objetivo como multi-objetivo, y tanto estáticos como dinámicos. BiCIAM le permite a los investigadores centrarse en el diseño del problema y el análisis de los resultados, reduciendo el esfuerzo de programación. Además dispone de una gran cantidad de algoritmos ya implementados, por lo que disminuye el nivel de conocimiento requerido a la hora de realizar un estudio en esta temática.

Índice

Introducción

A Planteamiento

Una amplia variedad de problemas del contexto socio-tecnológico actual tanto en países desarrollados como en vías de desarrollo se pueden plantear como problemas de optimización. Los problemas de optimización tienen como objetivo encontrar la solución que maximice o minimice una determinada función objetivo, dependiendo de diferentes variables de decisión. Algunas de las áreas donde podemos encontrar problemas de optimización son: distribución de mercancías, planificación de tareas, gestión de recursos, localización de servicios y recursos humanos, entre otras.

En muchos problemas de optimización como los mencionados encontramos elementos que presentan algún tipo de dinamismo o incertidumbre. Por ejemplo los tiempos de viaje en problemas de distribución de mercancías; o la demanda para los problemas de localización de servicios o instalaciones. Con el fin de modelar este tipo de situaciones se propusieron los denominados Problemas de Optimización Dinámicos (PODs). Los elementos del modelo de un POD que pudieran variar en el tiempo son: la función objetivo del problema, la posición del óptimo, las restricciones, el dominio de las variables, etc. Además en la literatura se han reportado diversos tipos de cambios. Entre ellos se pueden citar: con correlación con el cambio anterior, caótico, recurrente y recurrente con ruido. Entre los ejemplos clásicos de PODs se encuentran: el viajante de comercio dinámico (se añaden o se cancelan entregas en determinadas ciudades), la planificación de tareas que varían en el tiempo, la variación de la demanda de clientes que se deben satisfacer, entre otros [24].

Una de las herramientas más exitosas a la hora de abordar problemas con incertidumbre, imprecisión y dinamismo, es la Soft Computing o Computación Inteligente/Flexible [131]. Las componentes de la Soft Computing son: la lógica difusa, las redes neuronales, la computación evolutiva (las metaheurísticas en general) y el razonamiento probabilístico. Cada uno de estos componentes aborda diferentes problemas, la lógica difusa se ocupa principalmente de la imprecisión y el razonamiento aproximado, las redes neuronales del

aprendizaje y la adaptación, las metaheurísticas de la búsqueda de soluciones a problemas optimización, y el razonamiento probabilístico de la incertidumbre.

Las metaheurísticas son el componente de la Soft Computing que se ha aplicado principalmente para resolver PODs. Se pueden entender como algoritmos de propósito general, en el sentido que no tienen conocimiento a priori del problema a resolver. Según Talbi [121], se clasifican en dos grupos principales: los métodos basados en trayectoria, y los basados en poblaciones de soluciones. Las metaheurísticas han alcanzado un alto prestigio, como demuestran la amplia gama de problemas a los que se han aplicado con éxito en la literatura, y al gran número de revistas, libros y conferencias dedicados a este tema. Se han aplicado en diferentes contextos, tales como: minería de datos, física, biología, logística, control y procesamiento de imágenes, entre otros.

Cuando se afronta la resolución de un POD, se deben tener en cuenta los siguientes aspectos: los cambios son graduales, la información actual debe ser útil para lograr una adaptación más rápida cuando se produce un cambio, y resolver el problema desde cero después de un cambio sería como enfrentarse a un nuevo problema. Es sabido que las versiones clásicas de los algoritmos metaheurísticos tienen algunas carencias a la hora de abordar PODs. Por ejemplo, algunos presentan problemas de convergencia, que ocurren cuando el algoritmo ha convergido a un óptimo obsoleto del problema y es incapaz de encontrar o rastrear la nueva posición del óptimo tras la ocurrencia de un cambio. Además el ajuste de parámetros en ambientes dinámicos se hace más complejo que en ambientes estáticos, ya que las características del problema varían con el tiempo. También si se reinicia el ajuste de los parámetros después de un cambio, se perdería información. Por ello, para ser utilizados en escenarios dinámicos, los métodos clásicos se han extendido con enfoques que permitan tratar con estos problemas. Así, se encuentran enfoques basados en memoria (almacenar información de las mejores soluciones antes de cada cambio), multipoblacionales (utilizar subpoblaciones que manejen soluciones de diferentes áreas del espacio de búsqueda), mantenimiento de la diversidad durante la ejecución y después de cada cambio (aplicable a los algoritmos poblacionales, con el objetivo de introducir individuos aleatorios en la población actual o seleccionar en la próxima generación soluciones menos buenas pero más diferentes).

La aplicación de metaheurísticas a PODs presentan las mismas dificultades que en problemas estáticos: 1) dado un problema o una instancia desconocida es muy difícil saber cual va a ser el comportamiento de una determinada metaheurística, y 2) cual es el ajuste de parámetros que llevará a un mejor resultado. Para abordar el primer problema, se han desarrollado métodos que se complementan en fortalezas y debilidades para crear una sinergia entre diferentes metaheurísticas y mejorar la calidad de las soluciones obtenidas. Estos métodos se conocen como metaheurísticas híbridas. Para abordar el segundo problema, se han diseñado mecanismos de aprendizaje, para adaptar la configuración de parámetros en función de las características de la instancia o del estado actual de la búsqueda.

Entre las variantes de hibridación que se encuentran en la literatura (con resultados prometedores sobre los PODs) y que utilizan los enfoques comentados anteriormente, tenemos: los portafolios de algoritmos, las estrategias cooperativas y las hiperheurísticas. Las estrategias cooperativas se componen de un conjunto de agentes cooperativos y autónomos que se ejecutan simultáneamente intercambiando información entre ellos [76]. En el caso de las hiperheurísticas, estas se definen como métodos de búsqueda o mecanismos de aprendizaje para la selección o generación de heurísticas específicas, distinguiéndose por trabajar con el espacio de búsqueda de las heurísticas (o componentes heurísticos) en lugar de directamente en el espacio de búsqueda de soluciones [15]. Por otra parte, los portafolios de algoritmos se conocen como una colección de diferentes algoritmos y/o diferentes copias del mismo algoritmo que se ejecutan en diferentes procesadores o en el mismo procesador, y pueden retroalimentar información entre los algoritmos durante el proceso de búsqueda [62].

En esta tesis centramos nuestra atención en el diseño de portafolios de algoritmos, para resolver PODs combinatorios. Tiene un gran interés para este campo impulsar el estudio de los portafolios de algoritmos ya que han demostrado ser muy competitivos en los problema de optimización de ruteo de vehículos [115], abastecimiento de cadenas de suministros [137], funciones continuas [89], entre otros. Por esta razón, en esta tesis nos proponemos profundizar en este tema, para mostrar que los portafolios de algoritmo son una buena alternativa para solucionar PODs combinatorios.

B Objetivos

Dada la necesidad de abordar la naturaleza dinámica de los problemas del contexto socio tecnológico actual y las evidencias existentes sobre la adecuación de los métodos tipo portafolio para resolverlos, se plantea como objetivo general de esta tesis: estudiar, diseñar y evaluar esquemas de portafolio basados en metaheurísticas para abordar PODs de tipo combinatorio, teniendo en mente las ideas de adaptación, cooperación y aprendizaje que consideramos indispensables en este contexto.

A grandes rasgos, para alcanzar este objetivo, se realizará en primer lugar una evaluación de los métodos propuestos utilizando PODs sintéticos. En segundo lugar, se abordará la resolución del problema de máxima cobertura dinámico para estudiar el comportamiento de los métodos desarrollados sobre escenarios más reales.

Los objetivos específicos en los que se desglosa el objetivo general son:

- Realizar un estudio en profundidad en el ámbito de la Soft Computing, de cara a identificar las técnicas que se utilizan para resolver problemas de optimización dinámicos,

incluyendo las diferentes posibilidades de hibridación de técnicas.

- Diseñar un portafolio de algoritmos que integre técnicas de adaptación, cooperación y aprendizaje, para resolver problemas de optimización dinámicos.
- Validar el funcionamiento del portafolio de algoritmos con respecto a algoritmos del estado del arte sobre problemas de optimización dinámicos combinatorios de prueba y reales.
- Proponer una librería de software para facilitar la reutilización y aplicación de metaheurísticas en la resolución de problemas de optimización estacionarios y dinámicos.

C Resumen

Las tareas desarrolladas para alcanzar los objetivos planteados se describen en cuatro capítulos, una sección de comentarios finales y dos apéndices. La estructura de cada una de estas partes se introduce brevemente a continuación.

En el Capítulo 1, se presentan los fundamentos teóricos de la investigación. Inicialmente introducimos algunas definiciones básicas y los fundamentos relacionados con la Soft Computing, enfatizando el papel que juegan las metaheurísticas. Además se comentan algunas metaheurísticas clásicas, que son utilizadas en las contribuciones de la memoria. Luego se definen los portafolios de algoritmos, las estrategias cooperativas y las hiperheurísticas, que se han utilizado para resolver PODs. También se definen los conceptos sobre optimización en ambientes dinámicos a través de problemas, métodos y métricas que evalúan el rendimiento de los algoritmos. Finalmente, se presenta un breve resumen del capítulo.

En el Capítulo 2, se propone y evalúa un portafolio de algoritmos para resolver PODs, se describe el modelo, su esquema de aprendizaje y las diferentes variantes de dicho esquema. Luego se define el marco de experimentación, donde se comentan los detalles de los algoritmos de referencia utilizados para las comparaciones, y los test estadísticos que se aplicaron para la evaluación de los resultados. La siguiente parte del capítulo está dedicada a analizar los resultados de los experimentos. Se analizan los resultados de diferentes variantes del portafolio. Luego se comparan los resultados de la variante del portafolio que mejores resultados obtuvo con los algoritmos individuales que lo componen y con los algoritmos de referencia del estado del arte. Por último, se presenta un breve resumen del capítulo.

En el Capítulo 3, se presenta la aplicación de los métodos desarrollados al problema de máxima cobertura dinámico. En primer lugar, se realiza una revisión de la literatura donde se comenta cómo se ha abordado el problema y los principales métodos que se han utilizado

para resolverlo. A partir de esta revisión, se definen dos variantes nuevas que logran modelar el problema de una forma más cercana al mundo real. En la primera variante propuesta, se incorporan al modelo costes de apertura/cierre de instalaciones. En el segundo, se considera que el número de instalaciones abiertas permanece fijo a lo largo de todos los periodos. Para cada una de las versiones del problema, se realizan un conjunto de experimentos computacionales y se analizan los resultados obtenidos. Finalmente, se presenta un resumen del capítulo.

En el Capítulo 4, se propone una librería de software (framework), denominada BiCIAM, para resolver problemas de optimización, tanto dinámicos como estáticos, y que obtuvo la mención de "Premio Relevante" en el Concurso Nacional de Computación de Cuba del año 2015. Se comentan las principales características de BiCIAM y los algoritmos que tienen disponibles. Además se presenta su arquitectura, lo cual tiene como objetivo identificar y describir la estructura del framework a alto nivel de abstracción, las piezas principales que lo conforman, sus relaciones y formas de interacción. Se muestra un diseño detallado con los elementos significativos identificados, se describe cada elemento interno con un nivel de detalle suficiente para comprender la implementación, utilizando diagramas de clases de los paquetes principales. Seguidamente, se describen los patrones y mecanismos de diseño utilizados en la implementación. Luego se presenta un caso de estudio que ilustra el funcionamiento del framework para resolver un problema dinámico en particular. Finalmente, se presenta un breve resumen del capítulo.

Se ha incluido una sección de "Conclusiones y trabajos futuros", que resume los resultados obtenidos en esta memoria, presentando las principales conclusiones que se pueden extraer de éstos y se comentan algunos aspectos sobre trabajos futuros.

Finalmente, se incluyen dos apéndices que recopilan tablas de resultados complementarias a los resultados presentados en los capítulos anteriores. El primer apéndice se dedica a mostrar los resultados experimentales obtenidos para evaluar el rendimiento de los algoritmos presentados en el Capítulo 2. En el último apéndice se muestran los resultados experimentales obtenidos sobre las tres variantes del problema de máxima cobertura dinámico presentado en el Capítulo 3. La memoria termina con una recopilación bibliográfica que recoge las contribuciones más destacadas en la materia estudiada, y en las que se han apoyado las contribuciones de esta tesis.

Capítulo 1

Soft Computing y problemas de optimización dinámicos combinatorios

El objetivo de este capítulo es introducir los principales conceptos relacionados con la presente investigación. Inicialmente en la Sección 1.1 se realiza una descripción del concepto de Soft Computing, según la definición dada en [131], y sus principales componentes. En este sentido, en la Sección 1.2 se presenta especial atención a uno de los componentes principales que son las metaheurísticas, revisando con detalle los conceptos principales y algunos algoritmos clásicos del estado del arte. Luego en la Sección 1.3 se comentan los fundamentos de los problemas de optimización dinámicos, incluyendo los problemas artificiales y algunas aplicaciones de problemas reales. Además, se describen los portafolios de algoritmos, estrategias cooperativas e hiperheurísticas, que se han desarrollado integrando componentes de aprendizaje para guiar la búsqueda de forma adaptativa, logrando explorar y explotar mejor el espacio de búsqueda. También se describen un conjunto de medidas o métricas de rendimiento que permiten evaluar el comportamiento de las metaheurísticas frente a los PODs. Finalmente en la Sección 1.4 se concluye con un resumen de los aspectos teóricos analizados.

1.1. Soft Computing

En las últimas décadas, se han propuesto diferentes enfoques para desarrollar sistemas que sean capaces de aprender y comportarse inteligentemente, lo cual representa uno de los principales objetivos de la Inteligencia Artificial. Entre los enfoques más recientes se encuentra el de Soft Computing, definido por Zadeh en 1994, donde se hacía referencia a conceptos de forma aislada y se indicaba el empleo de la lógica difusa. Este término ha ido

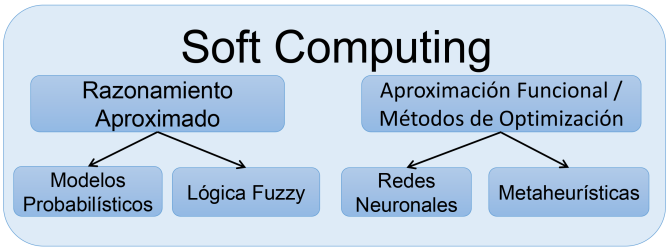

Figura 1.1: Métodos y técnicas que se relacionan para integrar la Soft Computing.

evolucionando y ha quedado situado como la base teórica del área de los Sistemas Inteligentes. Esto se encuentra sustentado en la definición de Verdegay en [131] estableciendo que:

Se trata de considerarla como antítesis de lo que podríamos denominar "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.

Entre los componentes principales de la Soft Computing se encuentran: razonamiento probabilístico, lógica y conjuntos fuzzy, redes neuronales y metaheurísticas. Se conoce como el resultado de la integración, cooperación, asociación o la hibridación de sus componentes. El marco definitorio de las principales metodologías que integran la Soft Computing se describe en la Figura 1.1.

De entre las diferentes herramientas que integran la Soft Computing, las encargadas de abordar los problemas de optimización son las metaheurísticas. Estas tienen una dimensión cada vez mayor ya que se consideran herramientas muy valiosas para proporcionar soluciones que los algoritmos exactos no son capaces de alcanzar cuando los recursos o el tiempo disponible son limitados. En la siguiente sección, se aborda este concepto con mayor profundidad porque es el que usaremos en nuestra investigación.

1.2. Metaheurísticas

Antes de comentar las principales características de las metaheurísticas es importante introducir los problemas de optimización. La optimización consiste en hallar la mejor solución posible a un determinado problema, ya sea con el propósito de maximizar o minimizar una

función objetivo específica. La variante más simple de optimización representa un problema donde no existen restricciones y si existen son de igualdad, con menor o igual número de variables que la función objetivo [30, 7], por ejemplo el problema de optimización clásico del viajante de comercio [98].

En el mundo de la optimización, los investigadores han clasificado los tipos de problemas según modelos matemáticos. Los problemas que pueden ser resueltos por algoritmos que necesitan un tiempo polinomial para llegar al estado óptimo pertenecen a la clase P, o sea, los problemas de complejidad polinómica tratable, que en la práctica se pueden resolver en un tiempo razonable. Además se conoce la clase NP, donde se incluyen aquellos que pudieran ser resueltos por un algoritmo no determinístico en un tiempo polinomial de resolución. En la teoría de complejidad computacional se encuentra también la clase NP-Completo (NPC), a cuyos problemas no se le conoce que exista un algoritmo que encuentre el estado óptimo en un tiempo polinomial; pero no se ha probado que pueda existir para al menos un problema. Además, se conoce la clase NP-Duro, que son aquellos problemas NPC que no han podido ser reducidos a problemas NP. Estos problemas se consideran muy difíciles de resolver [30].

Los problemas de optimización se pueden dividir en dos grupos, de acuerdo al número de objetivos que se tratan de optimizar: mono-objetivo y multiobjetivo [10]. La diferencia principal es el número de funciones objetivo a analizar, donde los mono-objetivo buscan obtener el mejor diseño o decisión, el cual es regularmente un máximo o mínimo global, según sea el caso de maximizar o minimizar. En cambio, en la optimización multiobjetivo puede no existir una solución que sea la mejor con respecto a todas las funciones objetivo y lo que se obtiene no es una única solución sino un conjunto de soluciones no dominadas o soluciones óptimas de Pareto.

Formalmente, un problema de optimización se puede definir de la siguiente forma: dado el espacio de búsqueda $\Omega \subseteq \mathbb{R}^D$, la función objetivo $f : \Omega \to \mathbb{R}$ y la relación de comparación $\succeq \in \{\leq; \geq\}$, la resolución de un problema de optimización consiste en encontrar el conjunto X de soluciones óptimas definido como:

$$X = \{x^* \in \Omega \mid f(x^*) \succeq f(x), \forall x \in \Omega\} \tag{1.1}$$

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

La existencia de una gran cantidad y variedad de problemas difíciles, que aparecen en la práctica y que necesitan ser resueltos de forma eficiente, impulsó el desarrollo de procedimientos capaces de llegar a buenos estados o soluciones, aunque no fuesen óptimos. Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan heurísticos o aproximados. Las heurísticas están hechas a medida y diseñadas para resolver un problema y/o instancia específica, por lo cual surgen las

metaheurísticas que son algoritmos de propósito general que se pueden aplicar para resolver cualquier problema de optimización [7]. También es importante tener en cuenta que según el teorema *No free lunch* propuesto por Wolpert y Macready en 1995 [136], no existe un algoritmo de optimización ideal, que su rendimiento fuese superior a cualquier otro sobre todas las funciones de optimización posibles, por lo cual dependiendo del problema hay que analizar el funcionamiento de los algoritmos.

Las metaheurísticas se pueden aplicar en diferentes contextos: aprendizaje automático, minería de datos, física, biología, logística, control y procesamiento de imágenes, entre otros. Se clasifican en dos grupos principales: basados en un punto o de trayectoria, y basados en poblaciones de puntos [121]. Los métodos de búsqueda basados en un punto son aquellos que partiendo de un punto, buscan la vecindad y actualizan la solución actual en función de esta, formando una trayectoria, de punto a punto. Los algoritmos basados en poblaciones, trabajan con un conjunto de soluciones en cada iteración, y su resultado debe ser el mejor individuo de la población o sea el individuo mejor adaptado.

En el conjunto de los algoritmos poblacionales se encuentran los Algoritmos Evolutivos, los cuales comprenden una serie de técnicas que tienen sus bases en modelos biológicos, y están inspiradas en la teoría darwiniana de la evolución natural. Se basan en el intercambio de información entre los individuos de una población o de distintas poblaciones, características comunes, comportamiento social, etc. El intercambio se produce como resultado de aplicar el proceso de selección, competición y recombinación de los individuos.

Se han desarrollado una gran variedad de metaheurísticas con diferentes criterios, teniendo en cuenta: uso o no de memoria, determinísticas o estocásticas, inspiradas en la evolución de la naturaleza, entre otras. A continuación se describen algunos de los métodos más conocidos y sus características esenciales, agrupados en basados en trayectoria y basados en poblaciones. Para su definición, y sin perder generalidad, supondremos que estamos abordando problemas de maximización.

1.2.1. Metaheurísticas basadas en trayectoria

Entre las metaheurísticas clásicas basadas en trayectoria se conocen: Búsqueda Aleatoria, Escalador de Colinas, Búsqueda Tabú y Recocido Simulado, en esta sección se describen las características más importantes de cada una de ellas.

1.2.1.1. Búsqueda Aleatoria

La Búsqueda Aleatoria (BA) es un algoritmo metaheurístico basado en un punto o en trayectoria [121]. Propone como estrategia la generación aleatoria, en cada iteración, de un

Algoritmo 1: Pseudocódigo del algoritmo Búsqueda Aleatoria

```
1: Entradas: N (número máximo de iteraciones)
2: Salida: x_best (mejor solución encontrada)
3: x_best ← generarSolucionInicial()
4: repeat
5:     x_c ← generarSolucionCandidataAleatoria()
6:     if f(x_c) > f(x_best) then
7:         x_best ← x_c
8:     end if
9:     n = n + 1
10: until se alcanza el criterio de parada, n > N
11: return x_best
```

elemento del espacio de estados. El esquema del funcionamiento general de la BA se describe en el Algoritmo 1. Inicialmente se le asigna al estado actual o mejor solución encontrada x_{best} un estado inicial generado aleatoriamente. Mientras no se cumpla una condición de parada, que podría ser un número máximo de iteraciones N, en cada iteración, se genera un estado candidato x_c de forma aleatoria, si x_c es mejor que x_{best} en términos de evaluación en la Función Objetivo, se actualiza x_{best} con el valor de x_c. Finalmente se devuelve el mejor estado encontrado x_{best} [1].

La BA puede ser lenta para encontrar un estado óptimo o cercano a él, ya que navega a ciegas por estados tanto mejores, peores, o repetidos, sin control alguno. Por lo tanto, su convergencia a buenas soluciones puede ser lenta, aunque su ejecución puede ser rápida gracias a su simpleza.

1.2.1.2. Escalador de Colinas

El algoritmo Escalador de Colinas (o Hill Climbing por su nombre en inglés) [1], tiene su inspiración en una analogía con una situación imaginaria en la que un alpinista busca escalar la cima de una montaña. Debido a la niebla el alpinista solo conoce si la superficie inmediata a él es ascendente o descendente [104]. El planteamiento está enfocado para maximizar; pero el algoritmo también es válido para minimizar.

El esquema del funcionamiento general del EC se describe en el Algoritmo 2. Los Escaladores de Colinas (EC) son métodos de propósito general y su limitación principal es la convergencia al máximo local más cercano. Por ello, en caso que la evaluación de la solución candidata x_c no sea mejor que la de la solución actual x_{best}, indica que la solución actual es un máximo local y el EC nunca saldrá de este punto. El tamaño de la vecindad se ha demostrado que influye significativamente en el comportamiento del EC, puesto que mientras más soluciones son vecinas de la actual, habrá menos probabilidad de caer en un máximo

Algoritmo 2: Pseudocódigo del algoritmo Escalador de Colinas Clásico

```
1: Entradas: N (número máximo de iteraciones), T (tamaño de la vecindad del estado actual)
2: Salida: x_best (mejor solución encontrada)
3: x_best ← generarSolucionInicial()
4: repeat
5:     x_c ← mejorVencino(x_best, T)
6:     if f(x_c) > f(x_best) then
7:         x_best ← x_c
8:     end if
9:     n = n + 1
10: until se alcanza el criterio de parada, n > N
11: return x_best
```

local. Otro punto interesante en la mejora del algoritmo se consigue aceptando soluciones candidatas cuya evaluación sea igual que la actual.

El EC Clásico explora la vecindad de manera exhaustiva para seleccionar el x_c. En caso de que x_c no sea mejor que x_{best}, x_{best} es un máximo local, lo que implica que el EC nunca saldrá de ese estado y por tanto la búsqueda se detiene.

Existen otras variantes del EC que se comentan a continuación [57, 104]:

- Escalador de Colinas Estocástico con Mejor Ascenso: esta variante es una adaptación del algoritmo anterior donde el tamaño de la vecindad es generalmente muy grande, haciéndose muy costosa realizar una iteración, por lo que sustituye la exploración exhaustiva de la vecindad por la exploración de una parte de la misma, seleccionada aleatoriamente.

- Escalador de Colinas Estocástico con Primer Ascenso o Primera Opción: la diferencia es que no se hace una exploración exhaustiva de una parte de la vecindad, sino se escoge aleatoriamente un x_c. Si el estado seleccionado es mejor que x_{best}, éste se actualiza. El algoritmo escoge el primero con mejor evaluación, de ahí se origina su nombre primer ascenso o primera valoración.

- Escalador de Colinas con Reinicio: cada cierta cantidad de iteraciones o cada vez que se compruebe que la búsqueda se ha detenido por un máximo local, comienza la búsqueda en un x_{best} generado aleatoriamente.

1.2.1.3. Recocido Simulado

El Recocido Simulado (o Simulated Annealing por su nombre en inglés) [16, 69], surge con el propósito de simular el proceso de recocido físico de metales. El esquema del

Algoritmo 3: Pseudocódigo del algoritmo Recocido Simulado

1: Entradas: N (número máximo de iteraciones), T (tamaño de la vecindad del estado actual), t_0 (temperatura inicial), t_f (temperatura final), m_t (número de iteraciones con la misma temperatura), $e(t)$ (función de reducción de temperatura)
2: Salida: *mejor* (mejor solución encontrada)
3: $x_{best} \leftarrow$ **generarSolucionInicial()**
4: **repeat**
5: **for** $(i = 0; i < m_t; i++)$ **do**
6: $x_c \leftarrow$ **generarVecino**(x_{best}, T)
7: **if** $f(x_c) > f(x_{best})$ **then**
8: $x_{best} \leftarrow x_c$
9: **else**
10: $random \leftarrow$ **generarNumeroAleatorio()**
11: **if** $random < exp^{-\frac{f(x_c)-f(x_{best})}{t_n}}$ **then**
12: $x_{best} \leftarrow x_c$
13: **end if**
14: **end if**
15: $n = n + 1$
16: **if** $f(x_{best}) > f(mejor)$ **then**
17: $mejor \leftarrow x_{best}$
18: **end if**
19: **end for**
20: $t_n \leftarrow e(t)$
21: **until** se alcanza el criterio de parada, $n > N$
22: **return** *mejor*

funcionamiento general del Recocido Simulado (RS) se describe en el Algoritmo 3. Es una de las metaheurísticas más antiguas y fue uno de los primeros algoritmos que contenían una estrategia explícita para escapar de óptimos locales, basándose en la idea fundamental de permitir movimientos que no producían mejoras, o sea aceptar soluciones peores que la solución actual x_{best}. La probabilidad de aceptación de soluciones peores disminuye a lo largo de la búsqueda, y depende de la evaluación en la función objetivo de la solución actual $f(x_{best})$ y un parámetro de control que regula la proporción de malas soluciones.

En el RS se establecen un conjunto de parámetros de entrada, entre ellos t_0 (temperatura inicial) y t_f (temperatura final), se debe establecer que la t_0 sea alta y t_f baja. Inicialmente se genera una solución inicial, que puede ser aleatoriamente o construida por alguna heurística específica. En cada iteración se genera una vecindad de la solución actual x_{best}, y se selecciona aleatoriamente una solución candidata x_c. Se acepta x_c como nueva x_{best} en función de su evaluación en la función objetivo $f(x)$, y la temperatura actual t_n. De esta forma, x_c se acepta si es mejor que x_{best}, o de lo contrario con una cierta probabilidad que depende de t_n y $f(x_c) - f(x_{best})$, esta probabilidad se calcula por lo general haciendo uso de la distribución de Bolztman ($exp^{-\frac{f(x_c)-f(x_{best})}{t_n}}$) [10].

La temperatura t_0 se reduce a medida que la búsqueda avanza, de esta manera la probabilidad de permitir movimientos que no mejoran es alta al principio y va disminuyendo gradualmente, de este modo la estrategia converge hacia el Escalador de Colinas. Además la selección de una función de reducción de temperatura es un aspecto muy importante en el funcionamiento del algoritmo, ya que hay algunas funciones que han obtenido mejores resultados que otras en determinados problemas.

1.2.1.4. Búsqueda Tabú

La Búsqueda Tabú (o Tabu Search por su nombre en inglés), fue propuesta por Glover [50, 51], es un algoritmo metaheurístico que tiene como característica especial el empleo de memoria adaptativa y estrategias especiales. La memoria adaptativa le permite restringir el entorno de búsqueda e introducir mecanismos de reiniciación a través de intensificación o diversificación en el espacio de soluciones. Su funcionamiento se divide en 3 fases: búsqueda preliminar, intensificación y diversificación.

El esquema del funcionamiento general de la BT se describe en el Algoritmo 4. Inicialmente la Búsqueda Tabú (BT) explora la vecindad de la solución actual x_{best}, y selecciona a la mejor solución de la vecindad y acepta esta solución como x_{best}, aunque sea peor. Este tipo de movimientos por el espacio de búsqueda puede conducir a la ocurrencia de ciclos entre dos soluciones [21]. Para evitar esta situación, BT prohíbe los últimos movimientos. Este mecanismo se basa en una memoria a corto plazo o lista tabú, que almacena la información que permite guiar la búsqueda de forma inmediata y organizar el modo en que se explora el espacio de soluciones. Los elementos se insertan y se retiran en un orden de "cola" , es decir, siguiendo un esquema FIFO.

En la etapa de intensificación la lista tabú está vacía y la búsqueda se inicia desde la mejor solución encontrada, procediendo como en la búsqueda preliminar para un cierto número de iteraciones. Luego tiene lugar la fase de diversificación, donde la lista tabú contiene los movimientos más frecuentes hasta ese momento y se selecciona la mejor solución de la vecindad pero que no se encuentre en la lista tabú, lo cual puede que esa solución esté distante de x_{best}. De esta forma, la BT logra explorar regiones prometedoras encontradas por la fase de intensificación y explorar nuevas regiones en la etapa de diversificación.

En el manejo de la lista tabú, el almacenamiento de soluciones completas puede ser ineficiente, desde el punto de vista de costo computacional, por lo que la lista tabú por lo general contiene atributos de las soluciones. Los atributos pueden ser componentes de la solución, movimientos o diferencias entre dos soluciones. Sin embargo, el almacenamiento de características de la solución tiene un inconveniente asociado, la pérdida de información. El mismo atributo se puede encontrar en más de un punto del espacio de búsqueda, el hecho de prohibir uno de ellos puede evitar buscar regiones inexploradas. Para solucionar este

Algoritmo 4: Pseudocódigo del algoritmo Búsqueda Tabú

Entradas: N (número máximo de iteraciones), T (tamaño de la vecindad del estado actual)
Salida: *mejor* (mejor solución encontrada)
$x_{best} \leftarrow$ **generarSolucionInicial()**
inicializarListaTabu()
repeat
  solucionesVecinas() $\leftarrow \{x_c \in T(x_{best}) \mid$ **estadoTabu**$(x_c) = false$
  OR criterioAspiracion$(x_c) = true\}$
  $x_{best} \leftarrow$ **mejorVecino**$(solucionesVecinas(x_{best}))$
  if $f(x_{best}) > f(mejor)$ **then**
    $mejor \leftarrow x_{best}$
  end if
  $n = n + 1$
until se alcanza el criterio de parada, $n > N$
return *mejor*

problema, algunos criterios de aspiración se pueden establecer, permitiendo la inclusión de determinadas soluciones en el conjunto permitido, incluso si están prohibidas por las condiciones tabú. Permitir que las soluciones que son mejores que la mejor solución encontrada, es un criterio aspiración común. En general se permitirán movimientos tabú si el criterio de aspiración determina que pueden ser factible.

1.2.2. Metaheurísticas basadas en poblaciones

Entre las metaheurísticas clásicas basadas en poblaciones se conocen: Estrategia Evolutiva, Algoritmo Genético y Algoritmo de Estimación de Distribuciones. A continuación en esta sección se describen las características más importantes de cada una de ellas.

1.2.2.1. Estrategia Evolutiva

Las Estrategias Evolutivas fueron ideadas por un grupo de estudiantes encabezado por Rechenberg en Alemania hacia el año 1964 [96], para resolver problemas hidrodinámicos de alto grado de complejidad. La versión original $(1 + 1) - EE$ usaba un solo padre y con él se generaba un solo hijo, este hijo se mantenía si era mejor que el padre, de lo contrario se eliminaba. Hasta este punto este algoritmo es muy similar al EC. Rechenberg introdujo el concepto de población, al proponer una Estrategia Evolutiva (EE) llamada $(\lambda + 1) - EE$, en la cual existían λ padres, y se genera un solo hijo, el cual puede reemplazar al peor padre de la población.

Posteriormente, se introdujo el uso de múltiples hijos en las denominadas $(\mu + \lambda) - EEs$, donde sobreviven los μ mejores individuos obtenidos de la unión de padres e hijos; y

$(\mu, \lambda) - EEs$, donde sólo los μ mejores hijos de la siguiente generación sobreviven [52, 93].

El esquema del funcionamiento general de la EE se describe en el Algoritmo 5. Inicialmente la EE parte de generar una población inicial aleatoria de individuos. Luego se va evolucionando la población actual como resultado de aplicar los operadores: selección, mutación, y reemplazo, hasta que no se cumpla la condición de parada que puede ser un número máximo de generaciones, y finalmente se devuelve la mejor solución o individuo mejor adaptado de la población actual.

Existen diversos métodos para llevar a cabo la selección de individuos que se van a reproducir, entre los que se encuentran [132]:

- Torneo: se selecciona el individuo con mejor adaptación de un grupo de individuos, elegidos aleatoriamente de la población.
- Truncamiento: se ordenan los individuos según su evaluación en la función objetivo, el umbral T indica la proporción de la población que va a ser seleccionada como padres.
- Ruleta: se asigna a cada individuo una probabilidad de selección proporcional a su evaluación en la función objetivo, la selección se realiza mediante lanzamientos independientes de la ruleta.

Luego los individuos obtenidos del proceso de selección se les puede aplicar el operador de mutación, y la ocurrencia de que esto se controla con un parámetro llamado probabilidad de mutación p_m. Existen varios métodos para aplicar la mutación, entre los más conocidos se encuentran [132]:

- Mutación en un punto: es utilizado para representaciones numéricas, cambiando el gen a mutar por un valor generado aleatoriamente y uniformemente en el dominio de la variable.
- Mutación por Intercambio: se intercambian dos elementos seleccionados aleatoriamente de una permutación.

Por último, se procede a determinar si se aceptan los hijos generados aplicando un operador de reemplazo, para ello existen varias técnicas, entre las que se encuentran [97]:

- Reemplazo Generacional: los individuos hijos sustituyen a los padres, o sea en la nueva generación no se tienen en cuenta los padres.
- Reemplazo Estado-Estable: se ordena la lista de individuos de mayor a menor adaptación, y la nueva generación serían los mejores entre padres e hijos, o sea se reemplazan los mejores hijos por los peores padres.

Algoritmo 5: Pseudocódigo de la Estrategia Evolutiva

1: Entradas: N (número máximo de generaciones), pc (probabilidad de cruzamiento), pm (probabilidad de mutación), m (tamaño de la población)
2: Salida: *mejor* (mejor solución encontrada)
3: $P \leftarrow$ **generarPoblacionInicial(m)**
4: **evaluarIndividuos**(P)
5: **repeat**
6: $P\prime \leftarrow$ **Seleccion**(P)
7: $P\prime\prime \leftarrow$ **Mutacion**($P\prime$, p_m)
8: **evaluarIndividuos**($P\prime\prime$)
9: $P \leftarrow$ **Reemplazo**($P\prime\prime, P$)
10: $n = n + 1$
11: **until** se alcanza el criterio de parada, $n > N$
12: *mejor* $\leftarrow$ **mejorPoblacion** (P)
13: **return** *mejor*

1.2.2.2. Algoritmo Genético

Los Algoritmos Genéticos (o Genetic Algorithm por su nombre en inglés) surgen entre los años 60 y 70 por el investigador John Holland de la Universidad de Michigan. Inicialmente desarrolló una técnica de búsqueda basada en mecanismos de supervivencia, planteados en la teoría de la evolución de Darwin. Esta técnica fue llamada originalmente planes "reproductivos", pero el nombre Algoritmo Genético se hizo popular tras la publicación de su libro en 1975 [60]. Otra de las referencia del tema ha sido el libro publicado por un discípulo de Holland, llamado David Goldberg en 1989 [52].

Los Algoritmos Genéticos (GA) simples se basan en un principio básico de la evolución, donde los mejores individuos tienen una mayor probabilidad de reproducirse y sobrevivir que otros individuos menos adaptados al entorno. Para implementar este principio, los GA mantienen una población que evoluciona a través del tiempo y que al final converge a una única solución. Los individuos de la población se representan mediante un cromosoma que codifica las variables del problema que se quiere resolver. Cada individuo o solución tiene una evaluación en la función objetivo, que mide la calidad de la solución, la cual es la herramienta que permite simular el concepto de individuos mejor adaptados [52, 97, 132].

El esquema del funcionamiento general del AG se describe en el Algoritmo 6. La estructura del AG es muy similar al de la EE, con la única diferencia que incorporan el proceso de recombinación de individuos o operador de cruzamiento, entre selección y mutación. Los individuos seleccionados se recombinan para intercambiar parte de su información genética. A partir de este intercambio se obtienen nuevos individuos que tienen una codificación que posee algo de ambos padres. La ocurrencia de esto se controla con un parámetro llamado probabilidad de cruzamiento p_c. Existen varios métodos para aplicar el cruzamiento, entre

Algoritmo 6: Pseudocódigo del Algoritmo Genético

```
1: Entradas: N (número máximo de generaciones), pc (probabilidad de cruzamiento), pm (probabilidad de
   mutación), m (tamaño de la población)
2: Salida: mejor (mejor solución encontrada)
3: P ← generarPoblacionInicial(m)
4: evaluarIndividuos(P)
5: repeat
6:    P′ ← Seleccion(P)
7:    P′′ ← Cruzamiento(P′, p_c)
8:    P′′′ ← Mutacion(P′′, p_m)
9:    evaluarIndividuos(P′′′)
10:   P ← Reemplazo(P′′′, P)
11:   n = n + 1
12: until se alcanza el criterio de parada, n > N
13: mejor ← mejorPoblacion (P)
14: return mejor
```

los más conocidos se encuentran [132]:

- Cruzamiento en un punto: se cortan sus cromosomas por un punto aleatorio para generar dos segmentos: cabeza y cola; y finalmente se intercambian las colas, generándose así los descendientes. El punto de cruce puede ser cualquiera de los 2 extremos de la cadena, en cuyo caso no se realiza el cruzamiento.

- Cruzamiento en dos puntos: es una generalización del método anterior, por lo que en lugar de cortar los cromosomas en un punto se corta en dos, manteniéndose los genes de los extremos, e intercambiando los del centro.

- Cruzamiento Uniforme: cada gen en los hijos se crea copiando el correspondiente gen de un padre u otro, utilizando para ello una máscara de cruce generada aleatoriamente. Donde hay un "1" en la máscara los genes en el primer hijo se toman del primer padre y donde hay un "0" se toman del segundo padre. Los genes del segundo hijo se establecen con las decisiones inversas.

1.2.2.3. Algoritmo de Estimación de Distribuciones

Los Algoritmos de Estimación de Distribuciones (o Estimation of Distribution Algorithms (EDAs) por su nombre en inglés) fueron introducidos en el ámbito de la computación evolutiva por Mühlenbein y Paaß. El EDA no requiere de operadores de cruce ni de mutación, la nueva población de individuos se obtiene por simulación de una distribución de probabilidad, la cual es estimada a partir de los valores de las variables de los individuos mejores, seleccionados en la generación anterior [107].

Algoritmo 7: Pseudocódigo del Algoritmo de Estimación de Distribuciones

```
1: Entradas: N (número máximo de generaciones), m (tamaño de la población)
2: Salida: mejor (mejor solución encontrada)
3: P ← generarPoblacionInicial(m)
4: evaluarIndividuos(P)
5: repeat
6:     P′ ← Seleccion(P)
7:     P′′ ← modeloProbabilístico(P′)
8:     evaluarIndividuos(P′′)
9:     P ← Reemplazo(P′′, P)
10:    n = n + 1
11: until se alcanza el criterio de parada, n > N
12: mejor ← mejorPoblacion (P)
13: return mejor
```

El esquema del funcionamiento general del EDA se describe en el Algoritmo 7. El EDA es un algoritmo que elimina la inspiración biológica, ya que su visión se centra en un muestreo estadístico de variables, donde cada hijo toma sus genes de cualquiera de los padres de manera aleatoria, lo cual acelera el cruzamiento pero pudiese ser muy destructivo. En el sentido en que cada hijo puede ser una mezcla de muchos padres, pudiendo no parecerse a ninguno en particular. El UMDA (o Univariate Marginal Distribution Algorithm por su nombre en inglés) es el caso más simple de EDA, donde cada gen escoge su valor según la probabilidad (muestreo probabilístico) de ese valor entre los padres escogidos. La selección de los padres se realiza aplicando la Selección por Truncamiento.

1.2.2.4. Optimización con enjambre de partículas

Los algoritmos basados en optimización con enjambres de partículas (o Particle Swarm Optimization (PSO) por su nombre en inglés) son metaheurísticas basadas en poblaciones e inspiradas en el comportamiento social observado en animales o insectos (por ejemplo: bandadas de peces, insectos, pájaros, etc). El PSO fue propuesto por Kennedy y Eberhart en el año 1995 [67], el enfoque original fue basado en la metáfora social que se resume en: los individuos que conviven en una sociedad tienen una opinión que es parte de un conjunto de creencias (el espacio de búsqueda), compartida por todos los posibles individuos y cada individuo puede modificar su propia opinión basándose en tres factores:

- su conocimiento sobre el entorno (su evaluación).
- su conocimiento histórico o experiencias anteriores (su memoria).
- el conocimiento histórico o experiencias anteriores de los individuos situados en su vecindario.

Algoritmo 8: Pseudocódigo del Algoritmo Optimización con enjambre de partículas

```
Entradas: N (número máximo de generaciones), m (total de partículas)
Salida: mejor (mejor partícula encontrada)
for (i = 1; i ≤ m; i++) do
    inicializar p_i → x_i, v_i
    Actualizar g_best
end for
repeat
    for (i = 1; i ≤ m; i++) do
        Calcular nueva velocidad v'_i, ecuación 1.2
        Calcular nueva posición x'_i, ecuación 1.3
        Evaluar p_i
        Actualizar memoria de p_i
        Actualizar memoria global de g_best
    end for
    n = n + 1
until se alcanza el criterio de parada, n > N
mejor ← g_best
return mejor
```

El esquema del funcionamiento general del PSO se describe en el Algoritmo 8. Los individuos en PSO reciben el nombre de partículas, y cada partícula i se compone por un vector de posición x_i (coordenadas en el espacio de búsqueda), un vector de velocidad v_i que define el desplazamiento o dirección en la cual se moverá la partícula, y la memoria de la mejor solución encontrada hasta el momento por ella p_i. En el enjambre de partículas se conoce la posición de la mejor solución encontrada en el enjambre hasta el momento g_{best}.

El enjambre se inicializa generando aleatoriamente las posiciones de las partículas p_i, y se calcula la evaluación en la función objetivo de cada partícula. Como las partículas al iniciar el algoritmo no están en movimiento, las velocidades iniciales son cero. Luego de inicializar el enjambre, las partículas se deben mover dentro del proceso iterativo. Una partícula se mueve desde una posición del espacio de búsqueda hasta otra, simplemente, añadiendo el vector posición x_i al vector velocidad v_i para obtener un nuevo vector posición, se calcula como:

$$x'_i = x_i + v'_i \tag{1.2}$$

$$v'_i = v_i + c_1 * n_1 * (p_i - x_i) + c_2 * n_2 * (g_{best} - x_i) \tag{1.3}$$

Donde n_1 y n_2 son vectores de n-dimensionales formados por números aleatorios en el rango [0,1], y c_1 y c_2 son números reales denominados coeficientes de aceleración o

ratios de aprendizaje. Puesto que los vectores x_i, v_i y g_{best} son reales, las operaciones de suma, resta y multiplicación se realizan sin ningún tipo de transformación. Cada uno de los componentes tiene un significado relacionado con el comportamiento social de los enjambres en la naturaleza, por ejemplo: $c_1 * n_1 * (p_i - x_i)$ representa el componente cognitivo del individuo y $c_2 * n_2 * (g_{best} - x_i)$ representa el componente social del individuo.

1.3. Problemas de optimización dinámicos combinatorios

Luego de introducir los conceptos básicos anteriores, en el siguiente apartado nos centraremos en la optimización en ambientes dinámicos. En esta sección se define formalmente un problema de optimización dinámico, y cuales son los modelos principales que se han utilizado en la literatura. Luego se comentan las metaheurísticas en la resolución de problemas de optimización dinámicos, y las principales referencias en la literatura. Por último se hace una revisión de las métricas para evaluar el rendimiento de los algoritmos, más importantes en este contexto.

1.3.1. Problemas dinámicos

Los problemas de optimización dinámicos (PODs) son problemas en los cuales algunos de sus elementos (función objetivo, restricciones, tamaño, etc) varían o cambian en el tiempo. Se definen como:

$$POD_s = \{optimizar f(x,t); x \in F(t) \subset S; t \in T\}$$

donde:

- S es el espacio de búsqueda
- t es el tiempo
- $f : S \times T \longrightarrow R$ es la función objetivo que asigna un valor numérico a cada posible solución ($x \in S$) en el instante t.
- $F(t)$, es el conjunto de posibles soluciones factibles en el instante t, $F(t) \subset S$.

Para estudiar los PODs se han desarrollado modelos artificiales de estos problemas que permiten investigar aspectos interesantes como la severidad (en que grado cambia el problema), frecuencia (cada cuánto tiempo cambia el problema) y tipo de los cambios. Un

problema artificial es un entorno de simulación que sirve para estudiar el comportamiento de un algoritmo, y la importancia de los problemas de este tipo se soporta en dos aspectos fundamentales: se evitan las consecuencias negativas que pueden ocurrir en entornos reales (mediante la simulación), y favorecen la comparación de propuestas en la comunidad científica.

En este sentido, se han hecho muy populares los generadores de problemas dinámicos [11, 70], los cuales están disponibles en código fuente de algún lenguaje de alto nivel, y permiten mediante la configuración de sus parámetros la obtención de diferentes familias de problemas dinámicos. Entre los generadores más utilizados en la literatura se encuentra el del problema de los Picos Móviles [11, 81, 82], el cual que provee una función multimodal altamente parametrizable que varía con el tiempo. Adicionalmente, Li y Yang en el año 2008 [70], propusieron el generador de problemas dinámicos GDBG (*Generalized Dynamic Benchmark Generator*), diseñado con la idea de construir entornos dinámicos a partir de la desviación de la distribución de las soluciones en el ambiente. Por otra parte, se conocen funciones más complejas que provienen de la optimización continua, tales como: Esfera, Griewank, Rastrigin, Ackley [72].

Además, existen versiones dinámicas de los problemas combinatorios, tales como: problema de la mochila [13, 66, 103], movimiento de la parábola [3], comparación bit a bit [32, 118] y problemas de planificación [33, 27, 78, 112, 113]. Los parámetros de estos problemas varían en el tiempo a partir de un conjunto de modelos que posibilitan analizar diferentes tipos de transiciones y que controlan la forma y el grado de interacción entre ellas.

Entre los problemas combinatorios más conocidos se encuentran: combinación de bit dinámico [142], mochila dinámico [103], viajante de comercio dinámico [79], ruteo de vehículos [92], entre otros. Estos problemas tienen características diferentes, entre las que se pueden mencionar: factores que cambian, visibilidad, predictibilidad, restricciones. También tienen puntos en común, en la mayoría de los casos los factores que cambian son las funciones objetivo, y suponen cambios periódicos, es decir, los cambios se producen cada un número fijo de generaciones o evaluaciones de la función objetivo.

Gran parte de los problemas del "mundo real" se conocen como: problemas de planificación, diseño de redes de transporte, problemas de empaquetado y recorte, problemas de ruteo de vehículos, problemas de asignación, entre otros. En la Tabla 1.1 se muestran algunos trabajos interesantes sobre los problemas del "mundo real" que muestran el comportamiento de entornos dinámicos, y sus referencias.

Entre los ejemplos se pueden mencionar, Mack y colaboradores en el año 2007 [75], resolvieron un problema de diseño aeroespacial aplicando un algoritmo genético. En ese mismo año, Michalewicz, Schmidt, Michalewicz y Chiriac [80], resolvieron tres casos de estudios: sistema para el control de la contaminación, planificación de rutas de buques y sistema para

	Referencias
Aplicaciones del mundo real	Diseño aeroespacial [75]
	Sistema de distribución de autos [80]
	Robótica evolutiva [126]
	Problemas de optimización financieros [125]
	Redes móviles ad-hoc [138]
	Planificación de Rutas [36, 80]
	Control de contaminación [80]
	Redes de Sensores Inalámbricos [91]

Tabla 1.1: Problemas de optimización dinámicos aplicados al mundo real, y las referencias correspondientes.

la distribución de autos para arrendamiento; utilizando inteligencia de negocio adaptativa. Además, Tezuka, Munetomo y Akama [125], hicieron una propuesta del problema de optimización de riesgos financieros, utilizando en la solución un algoritmo genético. Por otra parte, Dam, Lokan y Abbass en [26], resuelven la tarea de realizar minería de datos online en un entorno dinámico aplicando un Sistema de clasificación de aprendizaje basado en la genética. Luego, Yang y colaboradores en el año 2010 [138], abordaron el problema de enrutamiento del camino más corto en redes móviles ad-hoc, y utilizaron el algoritmo genético como método de solución.

Como se muestra anteriormente, muchos problemas de optimización combinatorios dinámicos, reales o prueba/sintéticos, se han utilizado en la literatura. Estos problemas tienen características diferentes, y para su clasificación se siguen los siguientes criterios [139]:

- Predictibilidad: si los cambios son predecibles o no. Algunos de los generadores de problemas se pueden configurar para permitir cambios predecibles, al menos en la frecuencia y la periodicidad de los cambios.
- Visibilidad: si los cambios son visibles para el algoritmo de optimización y, si es así, si los cambios se pueden detectar mediante el uso de pocos detectores.
- Restringido: si el problema está sujeto a restricciones cuando ocurre un cambio.
- Cíclico: si los cambios son cíclicos/recurrente en el espacio de búsqueda. La mayoría de problemas asumen cambios cíclicos, es decir, los cambios se producen cada un número fijo de generaciones o evaluaciones de la función objetivo.
- Factores que cambian: si los cambios involucran a los parámetros: función objetivo, dominio de las variables, número de variables, restricciones, entre otros.

1.3.2. Metaheurísticas en problemas dinámicos combinatorios

En problemas estacionarios, por lo general, el objetivo de un algoritmo metaheurístico es encontrar o aproximarse al óptimo global tan rápido y preciso como sea posible. Sin embargo, en las metaheurísticas para PODs, el objetivo se convierte en encontrar o aproximarse al óptimo global antes de cada cambio. Esto requiere un algoritmo para detectar en primer lugar los cambios y en segundo lugar un seguimiento de los óptimos cambiantes. También se espera que los problemas antes y después de un cambio de alguna manera se correlacionen entre sí. Un algoritmo de optimización puede aprender de su experiencia previa en la búsqueda, tanto como sea posible, lo cual le permitirá avanzar y la búsqueda puede ser más efectiva. De lo contrario, el proceso de optimización después de cada cambio, simplemente se convertirá en el proceso de resolución de un problema diferente.

La selección de una metaheurística y el ajuste de parámetros para solucionar un problema estático nunca ha sido una tarea trivial. Como consecuencia se han desarrollado investigaciones que mejoran los métodos con algunas características de aprendizaje para ajustar los parámetros de los algoritmos durante la ejecución. Esto también se está utilizando en el contexto de los POD, donde hay una tendencia creciente en el uso de mecanismos de aprendizaje para cambiar los parámetros y operadores del algoritmo durante la búsqueda. En [139] se han definido un conjunto de enfoques para adaptar adecuadamente las metaheurísticas a escenarios dinámicos y mejorar su funcionamiento, entre los que se encuentran:

- Seguimiento: consiste en iniciar la búsqueda cuando ocurre un cambio desde las soluciones óptimas anteriores encontrado. El propósito de seguimiento es doble. En primer lugar, en caso de que haya alguna correlación entre los cambios, a partir de las soluciones anteriores puede aumentar la probabilidad de hallar el nuevo óptimo. En segundo lugar, a partir de las soluciones existentes se puede minimizar el riesgo de forma significativa al cambio en el sistema existente, y por lo tanto reducir el costo operacional.

- Mantenimiento de la diversidad: se aplica a los algoritmos poblacionales. Consisten en introducir individuos al azar (inmigrantes aleatorios) a la población o introducir las versiones mutadas de los individuos actuales. Otra forma puede ser desviar el proceso de selección para que las soluciones menos buenas pero más diferentes sean seleccionadas para la próxima generación.

- Uso de memoria: almacenamiento de información útil a partir de la situación actual (ya sea implícita o explícita) y volver a usarla en una etapa posterior. La memoria puede ayudar a preservar las buenas soluciones del pasado que podrían ser útiles en un momento posterior, reutilizar buenos componentes de la solución en soluciones más recientes, para evitar las regiones del espacio de búsqueda cuya evaluación empeore.

- Aprendizaje y predicción: cuando son problemas muy conocidos obtener patrones en los cambios y aprender de estos patrones para predecir los cambios futuros y como reaccionar ante ellos.
- Multipoblación: utilizar subpoblaciones al mismo tiempo en la que cada cual maneje un área diferente del espacio de búsqueda y se puedan asignar tareas diferentes a cada subpoblación.

Entre los métodos propuestos para resolver PODs se encuentran: Algoritmos Evolutivos [86, 140], Enjambres de Partículas [8, 71], métodos Multi-Swarm [2, 9], Colonias de Hormigas [59, 79], estrategias cooperativas [56, 55], híbridos [65, 117]. Existe gran diversidad en las propuestas para resolver PODs, por lo cual al plantearnos un problema nuevo todo dependerá del problema en cuestión, cómo detectar los cambios para reaccionar adecuadamente y lograr mantener diversidad para enfrentar nuevos escenarios.

1.3.3. Métricas de rendimiento en problema dinámicos combinatorios

Con el fin de comparar las metaheurísticas en PODs, los investigadores han desarrollado diferentes medidas de rendimiento de acuerdo a los intereses que se planteen. Dentro de las medidas más utilizadas se encuentran: offline error y offline performance, propuestas por Branke y Branke-Schmeck en [12, 14]. Se definen formalmente por las siguientes expresiones:

$$offlineError = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{T}\sum_{e=1}^{T}F_{B_{opt}} - F_{B_{ie}^{k}}\right) \tag{1.4}$$

$$offlinePerformance = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{T}\sum_{e=1}^{T}F_{B_{ie}^{k}}\right) \tag{1.5}$$

donde N es el número de ejecuciones, T es el número de evaluaciones en la función objetivo en una ejecución, el valor $k = \lceil e/\tau \rceil, k \in \{0 \dots m\}$ es el periodo de cambio actual, m el total de cambios, $F_{B_{ie}^{k}}$ es la evaluación de la mejor solución encontrada en k y $F_{B_{opt}}$ es la evaluación del óptimo o la mejor solución conocida.

Entre las primeras medidas se encuentra la exactitud (*acurracy* por su nombre en inglés) propuesta por Feng en [39] y fue adaptada para problemas dinámicos por Weicker en [134], la cual tiene como principal objetivo caracterizar la precisión del algoritmo de forma diferente en cada generación y antes del cambio, está definida por la siguiente expresión:

$$acurracy^{k} = \frac{best^{k} - Min^{k}}{Max^{k} - Min^{k}} \tag{1.6}$$

donde $best^k$ es la evaluación de la mejor solución candidata en el cambio k, Max^k es la evaluación de la mejor solución encontrada o óptimo en k, y Min^k la evaluación de la peor solución en el espacio de búsqueda en k. Entre las desventajas que puede tener esta métrica es que no se puede analizar el funcionamiento global del algoritmo y que se debe conocer el óptimo o mejor solución conocida en el espacio de búsqueda.

Algunos estudios también han desarrollado medidas para evaluar la capacidad de los algoritmos en restringir el deterioro de la evaluación cuando se produce un cambio, las medidas más representativas son la estabilidad (*stability*) propuesta por Weicker [134] y la robustez propuesta por Rand y Riolo [95]. La estabilidad depende de *acurracy*, y se define por la siguiente expresión:

$$stability^k = max\{0, acurracy^{k-1} - acurracy^k\} \tag{1.7}$$

La medida de la robustez es similar a la medida de la estabilidad, también determina cuánto puede disminuir la evaluación cuando ocurre un cambio, se calcula como razón entre la evaluación promedio de la función objetivo de las soluciones de una población y la evaluación promedio de la función objetivo de la generación consecutiva [139].

$$robustness^k = max\{0, \frac{mean_p^k}{mean_p^{k-1}}\} \tag{1.8}$$

Adicionalmente Morrison en [83] propone una variante generacional del offline performance por lo cual está diseñada para que la utilicen los algoritmos evolutivos, se conoce como collective fitness, y se define por la siguiente expresión:

$$collectiveFitness = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{G}\sum_{e=1}^{G}F_{B_{ie}^k}\right) \tag{1.9}$$

donde G es el número de generaciones del algoritmo.

Por último, recientemente se han propuesto métricas más sofisticadas, que permiten analizar la convergencia de los algoritmos, se conocen como: *recovery rate* y *absolute recovery rate*, ambas propuestas por Nguyen y Yao en los años 2011 y 2012, respectivamente [85]. *Recovery rate* se utiliza para analizar cuán rápido el algoritmo se puede recuperar cuando ocurre un cambio y comienza a converger a una buena solución antes de ocurrir el próximo cambio. En el caso de *absolute recovery rate* se emplea para analizar cuán rápido el algoritmo comienza a converger al óptimo global antes del próximo cambio [139].

1.3.4. Métodos híbridos con aprendizaje y cooperación

Varios estudios han demostrado que las heurística y las metaheurísticas son herramientas eficaces para proporcionar buenas soluciones (excelentes en algunos casos), utilizando un número moderado de recursos. Una breve mirada a la literatura reciente revela la gran variedad de problemas y métodos que aparecen bajo el tema general de heurísticas de optimización. Una de las tendencias actuales en este ámbito es desplegar diferentes estrategias de resolución, que se usan para resolver los problemas de una forma coordinada. Hay dos razones principales para estos enfoques: en primer lugar, los problemas con instancias más grandes pueden ser resueltos; y en segundo lugar, herramientas robustas pueden obtener otras soluciones de alta calidad, a pesar de las variaciones en las características de las instancias.

Muchas técnicas han surgido en los últimos años para mejorar la resolución de problemas y toma de decisiones, creciendo el interés en la combinación de diferentes metaheurísticas o otras técnicas de optimización. Este tipo de algoritmo de búsqueda se conoce como metaheurísticas híbridas, y pueden proporcionar un rendimiento más robusto que los algoritmos "puros", cuando se trata de problemas del mundo real y de gran escala. De hecho, muchos de los mejores resultados reportados en la literatura para problemas académicos o reales, se han obtenido por este tipo de metaheurísticas. La principal motivación de la hibridación de diferentes algoritmos es el desarrollo de mejores estrategias de rendimiento, mediante la combinación de las ventajas de métodos "puros".

Entre las taxonomías más interesantes para clasificar las metaheurísticas híbridas se conoce la propuesta por Talbi en [120]. Es una taxonomía jerárquica, donde los híbridos se clasifican por su estructura. En el primer nivel se encuentran las hibridaciones de bajo nivel y de alto nivel. Los híbridos de bajo nivel se refieren a los casos en que se sustituye un componente determinado de una metaheurística para otra metaheurística, lo que conduce a una fuerte dependencia entre ellos. En los híbridos de alto nivel, los algoritmos mantienen su identidad original y son capaces de trabajar de forma autónoma. En el siguiente nivel las hibridaciones de bajo nivel y alto nivel, se clasifican en transmisión o cooperación. En las hibridaciones de transmisión se conoce que la salida de un algoritmo se conecta a la entrada de otro. Por otra parte, las hibridaciones cooperación son modelos de optimización de cooperación, donde un conjunto de agentes cooperan mientras que cada uno de ellos realiza su propia búsqueda.

En este apartado nos centraremos en metaheurísticas híbridas que se clasifican como: alto nivel con cooperación, ya que son los métodos más relacionados con los temas abordados en esta tesis. A continuación se describen las metaheurísticas híbridas: portafolios de algoritmos, estrategias cooperativas e hiperheurísticas, que se han utilizado para resolver diferentes problemas de optimización, donde se propone integrar componentes de aprendizaje

para guiar la búsqueda de forma adaptativa, logrando explorar y explotar mejor el espacio de búsqueda.

1.3.4.1. Portafolio de Algoritmos

Los portafolio de algoritmos fueron introducido por Huberman y colaboradores en el año 1997 [62], siguiendo la práctica estándar en economía, con el objetivo de maximizar los retornos esperados y minimizar los riesgos. Propusieron cómo ejecutar dos o más algoritmos, en este caso: Las Vegas [84], acortando el tiempo de espera para encontrar una buena solución en el problema de coloreado de grafos [62].

Cuando se hace referencia a un portafolio de algoritmo el concepto que más se utiliza es: colección de diferentes algoritmos y/o diferentes copias del mismo algoritmo que se ejecutan en diferentes procesadores o en el mismo procesador [45, 53, 62]. Ejecutar diferentes algoritmos en múltiples procesadores puede mejorar el rendimiento de los algoritmos en términos de tiempo computacional esperado y calidad de la solución obtenida, pero requiere un mayor costo computacional. En la literatura se conocen tres tipos de portafolios [53]: los algoritmos se ejecutan en paralelo (todos los algoritmos se ejecutan concurrentemente en diferentes procesadores), secuencial intercalado (se intercala el funcionamiento de los algoritmos en uno o más procesadores), y secuencial con reinicio (en un solo procesador ejecutar un algoritmo seleccionado aleatoriamente para una cantidad fija de tiempo o iteraciones, el cual se puede reiniciar con una nueva semilla aleatoria o seleccionar otro algoritmo) [45, 53, 89].

La selección de los algoritmos en un portafolio se puede realizar de dos formas: estática (se realiza antes de la ejecución) y dinámica (el proceso de selección se adapta durante la ejecución real de los algoritmos). Una de las estrategias de selección dinámicas o online que más se utiliza es: "el ganador se lleva todo", es decir, se selecciona el mejor algoritmo en función de su rendimiento en el problema, el principal inconveniente es que un algoritmo puede tener el mejor rendimiento en una instancia pero no necesariamente para el resto de los casos [53].

Una propuestas de portafolio de algoritmos dinámicos con aprendizaje fue presentada por Gagliolo y Schmidhuber en el 2006 [45], la cual trabaja con dos problemas: asignación de tiempo en base a predicciones de ejecución, y la clasificación del impacto de la asignación basada en modelos. Este portafolio de algoritmos dinámicos con aprendizaje admite la ejecución de varios algoritmos a la vez, donde la prioridad de un algoritmo depende de su tiempo previsto, que está condicionado por el hecho de que aún no ha encontrado la solución o la mejor solución conocida. Luego en el 2010 se propuso otra variante de portafolio dinámico presentada por Peng y colaboradores [89]. Aplican a 27 funciones continuas un portafolio

de algoritmos poblacionales que emplea un mecanismo de migración entre subpoblaciones, se activa periódicamente para fomentar el intercambio de información entre algoritmos. El esquema depende de dos parámetros: intervalo en que ocurre la migración y tamaño de la migración, que determinan el número de generaciones entre dos migraciones posteriores y el número de emigrantes en cada migración, respectivamente, cada vez que una migración se activa, se involucran todos los algoritmos del portafolio.

Entre las referencias de portafolios estáticos, se encuentra el trabajo de Yadav y colaboradores [137], para resolver el problema de abastecimiento de cadenas de suministros. En el portafolio no aplican ningún método para seleccionar los algoritmos o sea todos se ejecutan por separado y no intercambian información. Los algoritmos que lo constituyen son poblacionales. Recientemente Shukla y colaboradores, en el 2013 [115], resuelven el problema de ruteo de vehículos dinámico con demanda estocástica. Al igual que el anterior ejecutan en paralelo los algoritmos, por independiente y sin intercambiar información, y el portafolio está formado por algoritmos de trayectoria y poblacionales. Además, en [116] se propone un portafolio estático para resolver el problema de enrutamiento de inventario (IRP) con demandas estocásticas, donde los algoritmos se ejecutan en paralelo sin retroalimentación de información, los algoritmos que utilizan son cinco variantes de Algoritmos Genéticos.

Recientemente se propuso un portafolio de algoritmos evolutivos por Yin y Zhang en el año 2015 [143], en la solución propuesta los algoritmos se ejecutan secuencialmente y la selección de los algoritmos se realiza en cada generación de forma automática. La selección depende de un ranking que se obtiene a partir de la evaluación en la función objetivo de la mejor solución de la población para cada algoritmo. Los algoritmos evolutivos que componen al portafolio son: Optimización por colonia de abejas [31], CMA-ES [10, 68], Evolución Diferencial [10] y Optimización por cúmulo de partículas [34], en este sentido se puede observar que son algoritmos diseñados para resolver problemas continuos. La experimentación se realiza con 25 funciones continuas de prueba que se obtuvieron de la sección especial de CEC2005 [119]. Estas funciones se pueden dividir en cuatro clases: unimodal, multimodal básica, multimodal ampliada y de composición híbrida.

De manera general, la mayoría de los trabajos que utilizan portafolios de algoritmos emplean metaheurísitcas poblacionales [45, 89, 116], y en el caso de utilizar de trayectoria no intercambian información entre ellos [115], por lo cual sería interesante combinar metaheurísticas de trayectoria con poblacionales. Por otro lado, para un problema que tenga definido un conjunto de heurísticas de construcción específicas se podría diseñar un portafolio de heurísticas. Tampoco se han aprovechado las potencialidades de estas propuestas para PODs, ya que casi no se han utilizado para resolver este tipo de problemas y no se ha tenido en cuenta que estas técnicas introducen componentes que promueven la diversidad en las soluciones obtenidas para la no convergencia de los algoritmos cuando se produce un cambio.

1.3.4.2. Estrategias Cooperativas

Las estrategias cooperativas son ampliamente utilizadas en el campo de las metaheurísticas, y en la literatura se definen como: una búsqueda realizada por varios agentes altamente autónomos, aplicando cada método de solución en particular y un esquema de cooperación que combina estos agentes en una sola estrategia de resolución, intercambiando información sobre los estados, modelos y otras características del espacio de búsqueda [10, 22, 128]. Además, este tipo de técnica puede tener en cuenta que los agentes cooperen en paralelo o secuencial, donde cada agente realice una búsqueda en un espacio de soluciones. Existen varios de elementos que se debe tener en cuenta para su comportamiento, entre ellos: las heurísticas que componen la estrategia, como realizan el flujo de información, la información que intercambian y el modo y frecuencia de comunicación entre los agentes.

Las estrategias cooperativas se pueden clasificar según una propuesta por Talbi en el año 2002 [120], en las siguientes categorías:

- Homogéneas o Heterogéneas: se refiere a la composición de la estrategia, cuando todos los métodos combinados utilizan la misma metaheurística la estrategia es homogénea. Por el contrario, las estrategias heterogéneas obviamente utilizan algoritmos diferentes, las combinaciones entre los métodos basados en poblaciones y búsquedas basadas en trayectoria son probablemente los casos más conocidos.
- Global o Parcial: en el caso de global se refiere a aquellos métodos en los que los algoritmos intentan explorar todo el espacio de soluciones, siendo el caso más común en la literatura. En el caso de las estrategias parciales descomponen el espacio de búsqueda del problema y cada algoritmo se le asigna un subespacio específico.
- Especializada o General: en las estrategias generales, todos los componentes funcionan con la misma función objetivo, siendo el caso común en la literatura. Por el contrario, los métodos especializados están formados por algoritmos que resuelven problemas diferentes.

En las estrategias cooperativas también se debe tener en cuenta que es lo que es hibridado, desde este punto de vista se pueden considerar tres tipos distintos de estrategias cooperativas: combinación de metaheurísticas con metaheurísticas, metaheurísticas con algoritmos específicos de un problema o metaheurísticas con otras técnicas procedentes de Investigación de Operaciones o métodos de Soft Computing. Además se debe tener en cuenta el orden de ejecución de los algoritmos, en caso de considerar la ejecución secuencial se intercalan los componentes y la información se intercambia en una o varias direcciones; y en

paralelo la ejecución de los algoritmos es al mismo tiempo y se pueden obtener diferentes formas de comunicación entre ellos [94].

Existen trabajos donde se han aplicado estrategias cooperativas para resolver problemas de optimización estáticos [25, 76, 88]. Además se han utilizado para resolver PODS, entre las referencias se encuentra el trabajo presentado por Pelta y colaboradores en el año 2009 [87], en la propuesta utilizan una red de soluciones como un mecanismo implícito de la diversidad. Debido a que hay más soluciones que agentes, se tendría como una copia de seguridad de soluciones que no se tienen en cuenta por ningún agente. En esencia, es una estrategia multiagente descentralizada que utiliza una población de agentes de cooperación y soluciones, concentrándose el estudio en los mecanismos de cooperación y diversidad; y evalúan su funcionamiento con el problema de los picos móviles.

Además los autores Lung y Dumitrescu en ese mismo año presentaron un enfoque híbrido conocido como una cooperativa [74], basado en la colaboración entre un algoritmo de optimización por enjambre de partículas y un algoritmo evolutivo. Se diseñó para hacer frente a los óptimos en movimiento de los PODs. En esencia, utiliza tres poblaciones de individuos: dos poblaciones de estrategia evolutiva y una población de enjambre de partículas, las tres poblaciones evolucionan en paralelo y utilizan las reglas de un algoritmo de optimización multimodal que proponen para generar diversidad en la búsqueda. Para la experimentación utilizan funciones continuas de referencia en PODs.

Otro de los métodos propuestos como estrategias cooperativas para resolver PODs es el trabajo de González y colaboradores en el año 2011 [56], la estrategia consiste en un conjunto de heurísticas/hilos, las heurísticas (son varios algoritmos de Búsqueda Tabú) pueden ser diferentes algoritmos dependiendo del problema en cuestión. El coordinador procesa la información recibida de los solucionadores y produce ajustes posteriores de su comportamiento mediante el envío de "órdenes". Para los experimentos utilizan el problema de los picos móviles y las funciones de prueba: Ackley, Griewank y Rastrigin [72], y compraran los resultados obtenidos con el método propuesto anteriormente en [87].

En general, los trabajos que han utilizado estrategias cooperativas como solución a PODs en su mayoría han trabajado con problemas continuos, que se encuentran muy distantes de los problemas del "mundo real".

1.3.4.3. Hiperheurísticas

Una hiperheurísticas se definen como: un método de búsqueda o mecanismo de aprendizaje para la selección o generación de heurísticas para resolver problemas de optimización [15]. También se conocen como una metodología (de alto nivel), cuando se les da un caso par-

ticular (problema), y una serie de heurísticas de bajo nivel (o sus componentes) que producen automáticamente una adecuada combinación para resolver el problema dado. Algunos autores las definen como: algoritmo con aprendizaje cuando se utiliza alguna retroalimentación del proceso de búsqueda, de acuerdo con la fuente de información durante el aprendizaje, y se puede distinguir entre online y offline. La característica distintiva de las hiperheurísticas es que operan en un espacio de búsqueda de las heurísticas (o componentes heurísticos) en lugar de directamente en el espacio de búsqueda de soluciones para el problema que se esté abordando, como es el caso de la mayoría de los enfoques metaheurísticos [15, 20].

Entre los métodos de alto nivel para la selección/generación/combinación de heurística se conocen: búsqueda con vecindad variable (permiten encontrar buenas combinaciones de heurísticas parametrizadas), técnicas de minería de datos (redes neuronales, regresión y razonamiento basado en casos para encontrar patrones globales ocultos en grandes conjuntos de datos que producen las heurísticas), algoritmo genético con heurísticas específicas y técnicas para el descubrimiento de conocimiento. En el contexto de problemas estáticos, estos métodos se han aplicado a problemas como: coloreado de grafos [106], planificación de la producción [42], empaquetado binario [124], planificación de fuerza de trabajo [99], satisfacción de restricciones [123] y ruteo de vehículos [48].

La mayoría de los métodos de selección que utilizan las hiperheurísticas se basan en generar un crédito o puntuación online para cada heurística en base a su funcionamiento. El crédito de cada heurística es procesado en cada iteración para seleccionar la heurística a aplicar a la solución candidata, lo cual depende de los siguientes componentes: crédito inicial, longitud de la memoria, estrategia de selección basada en los resultados y reglas para la actualización del crédito en caso de mejora o empeoramiento. Por lo general, los créditos iniciales se fijan en el mismo valor, en algunos casos es cero. Dado estos créditos, se aplica una estrategia de selección que pude ser: seleccionar heurística con el crédito máximo o selección por ruleta. En algunos casos la selección por máximo crédito puede generar mejores resultados que la ruleta, pero cuando el tamaño del espacio de soluciones del problema aumenta, puede que la ruleta sea mejor ya que promueve diversidad [15].

Otro de los métodos que se utilizan en las hiperheurísticas es el aprendizaje por refuerzo [63], el cual interactúa con el entorno y dado una solución mediante prueba y error el sistema intenta aprender que acción llevar a cabo mediante su evaluación. En el contexto de hiperheurísticas, premiar y penalizar cada heurística en función del funcionamiento durante la búsqueda, utilizando un mecanismo de créditos. Si una heurística mejora una solución, entonces es premiada y el crédito se actualiza de forma positiva, mientras que si un movimiento provoca empeoramiento, se penaliza la heurística disminuyendo su crédito. Se han diseñado diferentes combinaciones de operadores para premiar y penalizar.

Las hiperheurísticas se han utilizado para resolver PODs, se conoce el trabajo de

Uludag y colaboradores en el año 2012 [54], para resolver el problema OneMax de prueba utilizando el generador XOR [141]. Esta hiperheurística se basa en dos variantes de HH-PBIL que difieren en la información que utilizan para actualizar la puntuación de las heurísticas de bajo nivel. El método de selección de heurísticas que emplean es el aprendizaje por refuerzo, premiando o penalizando los créditos de las heurísticas dependiendo de su funcionamiento.

Por otra parte, Kiraz y colaboradores en el año 2013 [68] proponen varias hiperheurísticas partiendo de un conjunto de métodos de selección: simple aleatorio, descenso aleatorio, permutación aleatoria y con descenso aleatorio; y algunos métodos de selección con aprendizaje: aprendizaje por refuerzo, hyper-mutation, CMA-ES. Los métodos anteriores los combinan con diferentes tipos de aceptación: todos los movimientos, sólo el mejor, mejor e igual, Exponencial Monte Carlo [4] y Recocido Simulado; utilizan como problema de prueba los picos móviles. Además recientemente Topcuoglu y colaboradores en el año 2014 [127], utilizan hiperheurísticas para resolver el problema de asignación generalizado dinámico y los picos móviles. Proponen un framework que combina diferentes métodos de selección y aceptación de soluciones, utilizando como metodología de alto nivel un algoritmo basado en poblaciones.

De forma general, la mayoría de los trabajos que utilizan hiperheurísticas proponen metodologías para la generación de heurísticas, las cuales son específicas para los problemas. Por otra parte, la selección de las heurísticas generalmente es secuencial, o sea en cada momento de la búsqueda se decide que heurística aplicar, lo cual no quiere decir que se pudieran ejecutar de forma intercalada o en paralelo. Otro punto interesante es que las hiperheurísticas no tienen porque estar condicionadas a utilizar heurísticas de bajo nivel, pudieran disponer de un conjunto de metaheurísticas, pero no es lo común.

1.4. Sumario

En este capítulo se han descrito los conceptos necesarios para comprender la investigación realizada. A partir del estudio presentado podemos resaltar los siguientes aspectos:

- La Soft Computing está formada por un conjunto técnicas y métodos que permiten abordar problemas reales, y las metaheurísticas se consideran entre sus principales componentes para resolver problemas de optimización.
- Los algoritmos metaheurísticos encuentran soluciones de alta calidad, y según el teorema *No Free Lunch* no existe uno mejor que otro sobre todas las funciones de optimización posibles.

- Existen técnicas que mejoran el funcionamiento de los algoritmos de búsqueda, mediante la hibridación con otros métodos y la integración con componentes de aprendizaje, que guían de forma adaptativa la búsqueda, tal es el caso de los portafolio de algoritmos, estrategias cooperativas e hiperheurísticas.

- Las estrategias cooperativas se definieron como un grupo de agentes que cooperan mientras llevan a cabo su búsqueda en el espacio de soluciones, y se han utilizado para resolver problemas de optimización dinámicos reportando muy buenos resultados en la literatura, en la mayoría de los casos son problemas de prueba o sintéticos.

- En las hiperheurísticas existen metodologías propuestas para la generación de heurísticas, pero en la mayoría de los casos son específicas a problemas, y entre los problemas más estudiados se conocen: SAT, viajante de comercio, planificación de horarios y programación, corte y embalaje, programación de la producción. Tienen como definición operar sobre el espacio de búsqueda de las heurísticas produciendo automáticamente una adecuada combinación para resolver el problema dado.

- En el diseño de los portafolio de algoritmos se contemplan diferentes enfoques, en cuanto a: selección de los algoritmos, tipo de ejecución (secuencial o paralela) e intercambio de información.

- Los portafolio de algoritmos en su mayoría utilizan metaheurísticas poblacionales, y en el caso de utilizar de trayectoria no intercambian información entre ellas, por lo cual sería interesante combinar metaheurísticas de trayectoria con poblacionales.

- Los problemas de optimización dinámicos se pueden resolver como un conjunto de problemas estáticos secuencialmente, introduciendo dinamismo en la transición de un problema a otro. Se debe tener en cuenta que se pueden presentar diferentes tipos de cambios, lo cual influye notablemente en la caracterización del problema en cuestión.

- Existe diversidad en la aplicación de las metaheurísticas para resolver problemas de optimización dinámicos, por lo cual dependiendo del problema constituye un papel importante la detección de los cambios para una reacción adecuada y el mantenimiento de la diversidad frente a nuevos escenarios.

Capítulo 2

Un portafolio de algoritmos para problemas de optimización combinatorios dinámicos

La idea de que múltiples algoritmos cooperen entre ellos y aprendan unos con otros, no es nueva, y la literatura ha mostrado que puede dar buenos resultados. En este sentido, algunos investigadores han desarrollado diferentes enfoques y métodos para resolver problemas de optimización combinatorios dinámicos, debido a los retos que impone el dinamismo. Estas propuestas tienen una tendencia creciente en el uso de mecanismo de aprendizaje que mejoran los métodos clásicos, con la incorporación de características para ajustar los parámetros de los algoritmos durante el proceso de búsqueda.

Teniendo en cuenta los aspectos anteriores, en este capítulo proponemos un portafolio de algoritmos para resolver PODs, ya que dispone de metaheurísticas que trabajan sobre su propia solución o soluciones. Nuestra propuesta se basa en varios esquemas de aprendizaje, con el objetivo de evaluar si debe olvidar o no lo aprendido después de un cambio, y cuál es el esquema de refuerzo más apropiado. El modelo usa un mecanismo de aprendizaje que asigna créditos a los diferentes métodos para penalizarlos o premiarlos de acuerdo a su evolución, con el fin de seleccionar el mejor algoritmo en cada momento de la búsqueda.

En este sentido, el proceso de selección de los algoritmos en el portafolio es dinámico, o sea la selección se realiza online y se ajusta durante la ejecución de los algoritmos. En cada momento de la búsqueda, en el portafolio se selecciona que metaheurística aplicar. Los algoritmos que integran el portafolio son variantes clásicas y generales de metaheurísticas, y se ejecutan iterativamente bajo un esquema sencillo de aprendizaje.

Este capítulo se organiza como sigue. En la primera sección se describe el portafolio de

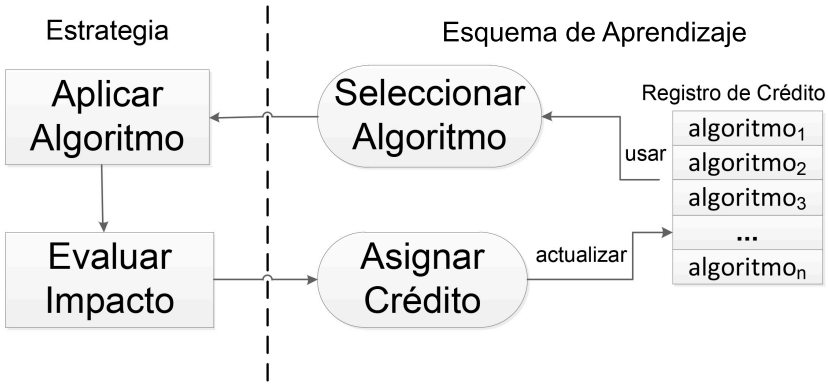

Figura 2.1: Esquema del funcionamiento general del portafolio de algoritmos.

algoritmos propuesto, su esquema de aprendizaje y las diferentes variantes. En la Sección 2.3 se comentan los detalles de la experimentación, problemas y algoritmos de referencia para las comparaciones con nuestra propuesta. Luego en la Sección 2.4 se analizan los resultados de la experimentación. Finalmente, en la Sección 2.5 se presenta un breve resumen del capítulo.

2.1. Portafolio de Algoritmos

El portafolio de algoritmos que se presenta en esta memoria está compuesto por un conjunto de metaheurísticas: A $= \{a_1, \ldots, a_n\}$, siendo cada a_i una metaheurística de trayectoria o basada en población. Cada a_i tiene una solución actual $x_{curr}^{a_i}$ (para el caso de los métodos basados en trayectoria) o una población actual $p_{curr}^{a_i}$ (para las metaherísticas poblacionales). En la Figura 2.1 se muestra un esquema del funcionamiento general del portafolio propuesto, donde se puede apreciar la relación entre sus componentes.

En el Algoritmo 9 se muestra el funcionamiento interno del portafolio de algoritmos. Luego de la inicialización de los algoritmos, el método entra en el ciclo principal. En primer lugar, si se detecta un cambio, se re-evalúan $x_{curr}^{a_i}$ y $p_{curr}^{a_i}$, y se calcula la nueva mejor solución global (x_{best}). Luego, se selecciona un a_i y se ejecuta durante una iteración (la definición de lo que consideramos como una iteración se da al final de esta sección). La estrategia de selección utilizada en esta etapa es el método de la ruleta, donde se asigna la probabilidad de selección de cada a_i de acuerdo con su crédito w_i. Esta probabilidad se calcula como:

$$P_{selec}^{a_i} = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{2.1}$$

La solución generada por el algoritmo seleccionado en una iteración es x_c y se compara a continuación con la mejor solución global, x_{best}. Si x_c es mejor que x_{best}, se actualiza el

Algoritmo 9: Pseudocódigo del Portafolio de Algoritmos

```
Entrada: A (conjunto de metaheurísticas)
Salida: x_best (mejor solución encontrada)
n = tamaño del portafolio
for (i = 1; i ≤ n; i++) do
    inicializar a_i
end for
while no condicion − parada do
    if se detecta cambio then
        for (j = 1; j ≤ n; j++) do
            re-evaluar x_curr^aj or p_curr^aj
            re-calcular x_best
        end for
    end if
    a_i ← seleccionar un algoritmo de A con una probabilidad de P_select^ai
    x_c ← ejecutar a_i por una iteración
    if f(x_c) > f(x_best) then
        x_best ← x_c
        for (j = 1; j ≤ n; j++) do
            actualizar x_curr^aj o p_curr^aj con x_best
        end for
    end if
    actualizar el crédito de a_i
    for (j = 1; j ≤ n; j++) do
        actualizar la probabilidad de selección P_selec^aj
    end for
end while
return x_best
```

portafolio de algoritmos. Concretamente, se actualiza la solución actual o población del resto de algoritmos ($a_j \neq a_i$) con el nuevo x_{best}, para retroalimentar al resto de los algoritmos con la información generada por a_i. En la última etapa del ciclo principal, el método asigna un crédito a a_i y recalcula las probabilidades de selección de los algoritmos de acuerdo con la Ecuación 2.1. El crédito asignado a a_i depende de la calidad de la solución x_c y el esquema de créditos utilizado. Los detalles del esquema de asignación de créditos del portafolio de algoritmos se describirán en el apartado siguiente.

Algunos de los pasos mencionados anteriormente se ejecutan de una manera diferente dependiendo de si el algoritmo considerado es basado en trayectoria o basado en poblaciones. En la Tabla 2.1 muestra cómo el portafolio de algoritmos lleva a cabo estos pasos en cada caso. Las diferencias más importantes aparecen en cómo se realiza una iteración, cómo se obtiene x_c y cómo se actualiza la solución actual o población actual. En relación con la ejecución de una iteración y la obtención de x_c, para algoritmos basados en trayectoria, una

Línea del Pseudocódigo en el Algoritmo 9	*basado en Trayectoria*	*basado en Población*
Inicialización de los algoritmos (5)	generar aleatoriamente una solución siguiendo una distribución uniforme	generar aleatoriamente una población siguiendo una distribución uniforme
Re-evaluar (10)	re-evaluar la solución $x_{curr}^{a_i}$	re-evaluar toda la población $p_{curr}^{a_i}$
Una iteración (15)	aplicar un operador de vecindad a $x_{curr}^{a_i}$ y evaluar dependiendo del criterio de aceptación este movimiento	ejecutar secuencialmente la aplicación de los operadores correspondiente al algoritmo poblacional
x_c (15)	solución resultado de aplicar un operador de vencindad a $x_{curr}^{a_i}$	un individuo como resultado de un ciclo evolutivo de operadores a la población $p_{curr}^{a_i}$
Actualizar $x_{curr}^{a_i}$ o $p_{curr}^{a_i}$ (19)	reemplazar $x_{curr}^{a_i}$ por x_{best}	sustituir x_{best} por el peor individuo de la población $p_{curr}^{a_i}$

Tabla 2.1: Detalles de los pasos del portafolio de algoritmos, que se ejecutan de manera diferente en función de si el algoritmo seleccionado se basa en trayectoria o basado en población.

iteración corresponde a una aplicación del operador de vecindad para $x_{curr}^{a_i}$ y la evaluación del criterio de aceptación del movimiento (por ejemplo, tabú y criterios de aspiración en la Búsqueda Tabú, probabilidad de aceptación del Recocido Simulado, etc), y x_c representa la solución resultado de la aplicación del operador de vecindad.

En los métodos basados en poblaciones, una iteración corresponde a la aplicación secuencial de sus operadores (por ejemplo: selección → cruzamiento → mutación → reemplazo, en Algoritmos Genéticos). Cuando se aplica un operador de cruzamiento, solo uno de los dos individuos obtenidos (seleccionados al azar de acuerdo con una distribución uniforme) se considera para el siguiente paso de la secuencia de aplicaciones de operadores. En caso del método de reemplazo, el individuo generado reemplaza al peor de los padres. Por tanto, x_c corresponde al individuo generado en este proceso. La actualización de $x_{curr}^{a_i}$ para los algoritmos basados en la trayectoria, en el paso de 17 del Algoritmo 9, consiste en el reemplazo de $x_{curr}^{a_i}$ por x_{best}. En cuanto a los algoritmos basados en la poblaciones, en este paso, x_{best} reemplaza el peor individuo de la población.

2.1.1. Esquema de Aprendizaje

El mecanismo de asignación de crédito implementado en el portafolio es de hecho un esquema de aprendizaje, cuyo objetivo es aprender que el mejor método o métodos ofrecen un mejor rendimiento para el problema en cuestión.

El crédito asignado a una metaheurística en el tiempo $t + 1$ se calcula como:

$$w_i(t+1) = w_i(t) + r_i(t) - l_i(t) \tag{2.2}$$

donde:

- t es el instante de tiempo actual.
- $w_i(t+1)$ es el crédito obtenido por a_i en el tiempo $t+1$.
- $w_i(t)$ es el crédito actual de a_i.
- $r_i(t)$ es el premio asignado a a_i. Es un valor mayor o igual a cero que se determina en función de la calidad de la solución generada por a_i y la calidad de x_{best}.
- $l_i(t)$ penalización asignada a a_i si la solución generada es peor que x_{best}. Se define como sigue:

$$l_i(t) = w_i(t) * Q \tag{2.3}$$

donde: Q es la constante de penalización, $Q \in [0, 1]$.

Suponemos que cada posible definición o variante para la actualización del premio, penalización y crédito, conducen a un esquema de aprendizaje diferente.

2.1.2. Variantes del Esquema de Aprendizaje

Cuando nos enfrentamos a PODs, es fundamental decidir qué hacer con el aprendizaje obtenido (crédito asignado) por el portafolio. Las principales componentes del esquema de aprendizaje son: $w_i(t)$, $r_i(t)$ y $l_i(t)$. Las diferentes combinaciones de los tres componentes conducen a las variantes del portafolio que se analizarán, con el fin de detectar las mejores alternativas. Las principales definiciones se describen a continuación:

Crédito actual (*$w_i(t)$*):

- Reinicio (RS): el crédito actual se fija en cero cuando se detecta un cambio. El fundamento de esta idea es que para enfrentar un nuevo problema necesitamos detectar si partir de cero es bueno, o sea olvidar lo aprendido.
- No Reinicio (NRS): suponemos que la nueva situación es similar a la anterior y si un método era bueno en el pasado, entonces será bueno en el futuro, o sea no se reinicia el crédito si se detecta un cambio en el problema.

Variante	$w_i(t)$	$l_i(t)$	$r_i(t)$
RS-AP-RB	RS	AP	RB
RS-AP-REB	RS	AP	REB
RS-IP-RB	RS	IP	RB
RS-IP-REB	RS	IP	REB
NRS-AP-RB	NRS	AP	RB
NRS-AP-REB	NRS	AP	REB
NRS-IP-RB	NRS	IP	RB
NRS-IP-REB	NRS	IP	REB

Tabla 2.2: Variantes de los esquemas de aprendizaje.

Penalización ***($l_i(t)$):***

- Penalización Activa (AP): $Q = 0.9$ si a_i genera soluciones peores que x_{best}, y $Q = 0$ en caso contrario. De esta manera, disminuimos el crédito de un método si no está permitiendo mejorar la mejor solución encontrada. El valor $Q = 0.9$ fue seleccionado luego de observaciones empíricas.
- Penalización Inactiva (IP): $Q = 0$ durante toda la ejecución.

Premio ***($r_i(t)$):***

- Mejor (RB): $r_i(t) = 1$ si la evaluación en la función objetivo de la solución generada $f(x_c)$ es estrictamente mejor que la evaluación de la mejor solución encontrada $f(x_{best})$.
- Mejor o Igual (REB): $r_i(t) = 1$ si la evaluación en la función objetivo de la solución generada $f(x_c)$ es mejor o igual que la evaluación de la mejor solución encontrada $f(x_{best})$.

Analizamos las ocho variantes posibles derivadas de las combinaciones de los componentes anteriores (que se muestran en la Tabla 2.2). Como comentamos anteriormente, cada combinación se puede considerar como un "esquema de aprendizaje".

2.2. Algoritmos de referencia para la comparación de los resultados

Para evaluar el rendimiento del portafolio de algoritmos se van a realizar comparaciones con algunos algoritmos de referencia. En una revisión reciente de la literatura en [24] se

muestran dos algoritmos para resolver PODs que obtienen buenos resultados, son: Adaptive Hill Climbing Memetic Algorithm (AHMA) propuesto en el año 2008 en [133] y Self Organized Random Immigrants Genetic Algorithm (SORIGA) presentado en el año 2007 en [126]. Aunque existen métodos más recientes para los mismos problemas utilizados en este trabajo [40, 129], AHMA y SORIGA todavía se utilizan en las comparaciones [29, 129, 77], puesto que hay un acuerdo general sobre su calidad, son fáciles de entender y su código fuente se encuentra disponible, por lo que ejecutar los experimentos en nuestros escenarios es mucho más fácil. Consideramos que de esta forma la comparación es más justa que tomando los valores publicados en los artículos.

Adaptive Hill Climbing Memetic Algorithm (AHMA)

El algoritmo AHMA (Adaptive Hill Climbing Memetic Algorithm) fue propuesto por H. Wang y colaboradores en el año 2009 [133]. Combina un algoritmo genético con un algoritmo de búsqueda local que utiliza dos operadores de vecindad: Greedy Crossover Hill Climbing (GCHC) y Steepest Mutation Hill Climbing (SMHC). GCHC aplica un operador de cruzamiento, utilizando la solución élite (mejor solución de la población) y un individuo de la población (elegido mediante selección por ruleta), y se devuelve el mejor hijo obtenido. SMHC selecciona al azar un número de bits de la solución élite y los invierte. Si la nueva solución (resultado de aplicar: GCHC o SMHC) es mejor que la élite, ésta se reemplaza.

Los operadores se seleccionan siguiendo una distribución de probabilidad que se ajusta durante la ejecución del algoritmo, con el objetivo de dar una mayor probabilidad al operador que mejores resultados obtiene, o sea el esquema de aprendizaje ajusta los valores de probabilidad en función de si producen o no mejoras. Después de cada cambio los valores de probabilidad se mantienen, se utiliza esa información en el transcurso del proceso de búsqueda.

AHMA también incluye dos mecanismos para gestionar la diversidad de la población durante la ejecución: Adaptable Dual Mapping (ADM) y Triggered Random Immigrants (TRI). El mecanismo ADM utiliza el concepto de individuo dual para obtener un nuevo individuo, tal como: dado $x = (x_1, ..., x_n) \in I$ de tamaño n, el individuo dual es definido como $x' = (x'_1, ..., x'_n) \in I$ donde $x'_i = 1 - x_i (i = 1, ..., n)$. El mecanismo TRI se utiliza cuando se detecta que los individuos de la población actual tienen un grado de convergencia por debajo de un umbral, por lo cual se generan individuos aleatoriamente y reemplazan a los individuos de la población actual que no cumplen con el umbral dado.

SORIGA

El algoritmo SORIGA (o Self Organized Random Immigrants Genetic Algorithm) propuesto por Tinós y Yang en el año 2007 [126], es un algoritmo genético en el que después de la inicialización, su población se divide en dos sub-poblaciones: principal y secundaria. Los mejores individuos se mueven a la subpoblación principal mientras que los peores a la subpoblación secundaria. Ambas subpoblaciones coevolucionan independientes, hasta que el peor individuo entre las dos subpoblaciones se encuentra en la población principal. Cuando eso ocurre se unen las subpoblaciones y evolucionan una generación, luego se dividen de la misma forma comentada anteriormente. El proceso anterior se repite de manera iterativa hasta que se alcanza la condición de parada, que pudiera ser el número de generaciones.

El flujo de inmigrantes entre dos poblaciones generalmente aumenta el nivel de diversidad genética de una población, lo cual puede permitir escapar de óptimos locales. Además utilizan una estrategia de reemplazo aleatorio, donde los individuos con menor adaptación de la población principal son reemplazados por individuos generados aleatoriamente, lo cual igualmente promueve la diversidad.

2.3. Marco de Experimentación

En esta sección se describen los detalles a seguir en la experimentación para evaluar el rendimiento de la propuesta. La organización de esta sección es la siguiente:

- La Sección 2.3.1 presenta los cinco problemas utilizados para los experimentos.
- La Sección 2.2 describe la composición del portafolio propuesto.
- La Sección 2.3.3 describe los métodos de comparación utilizados que se basan en test estadísticos no paramétricos.

2.3.1. Problemas

Para inducir dinamismo en algunos problemas combinatorios estáticos de prueba conocidos, se ha utilizado el generador XOR-POD [141]. El XOR-POD puede generar entornos dinámicos a partir de cualquier problema binario estático utilizando un operador o-exclusivo (XOR) bit a bit. La función objetivo cambia cada τ generaciones. Para el k-ésimo cambio, se genera incrementalmente una máscara XOR $M(k)$ de la siguiente forma:

$$M(k) = M(k-1) \oplus T(k) \tag{2.4}$$

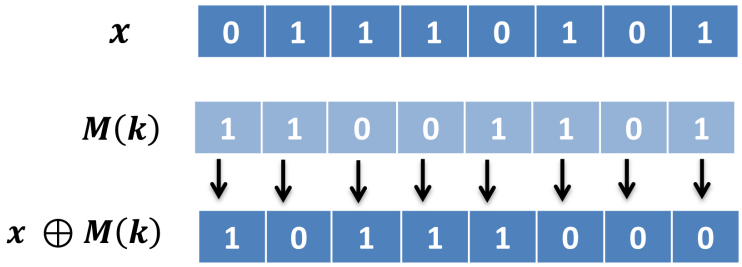

Figura 2.2: Dada una máscara $M(k)$ y una solución x, la función objetivo es aplicada como $f(x \oplus M(k))$, donde $\oplus$ es el operador XOR (OR exclusivo).

donde $\oplus$ es el operador XOR ($a \oplus b = 1 \iff a \neq b$, y $a \oplus b = 0$ en cualquier otro caso) y $T(k)$ es una máscara binaria creada aleatoriamente con $\rho \times m$ unos en el k-ésimo periodo de cambio. Para el primer cambio $k = 1$, $M(1) = \{0 \ldots 0\}$. De esta forma la solución x se evalúa en la generación t como:

$$f(x, t) = f(x \oplus M(k)) \tag{2.5}$$

donde $k = \lceil t/\tau \rceil$ es el índice de los cambios, siendo $\lceil \rceil$ el operador parte entera. De esta manera, el generador XOR va modificando la solución óptima a alcanzar cada vez que se genera un cambio en el problema. Esta modificación se consigue mediante la operación XOR sobre la solución óptima anterior y la máscara binaria aleatoria $M(k)$, de forma tal que la nueva solución óptima diferirá de la anterior en tantos bits como unos (bits con valor 1) haya en la máscara, como se ilustra en la Figura 2.2.

Una de las ventajas de este generador XOR radica en que se pueden ajustar fácilmente la frecuencia y la severidad de los cambios, mediante los parámetros τ y ρ, respectivamente. Un mayor valor de ρ lleva a cambios más severos, mientras que una menor τ significa cambios más frecuentes.

Los otros problemas de optimización combinatoria más utilizados con codificación binaria son [141]: OneMax, Plateau, RoyalRoad y Deceptive. En todos estos problemas el objetivo consiste en encontrar soluciones que tengan exactamente los mismos bits que la solución óptima, donde inicialmente y para el caso estático, se toma como aquella solución en que todos los bits son 1. Para evaluar una solución, consideramos bloques de 4 bits, donde cada bloque contribuye una cantidad dada al valor objetivo final. La contribución de cada bloque de 4 bits para cada una de las funciones consideradas se calcula como sigue:

- **OneMax**: Cada bit correcto suma 1 al valor de la función objetivo.
- **Plateau**: Si se aciertan 3 bits, se contribuye con un valor de 2 al valor objetivo y si se

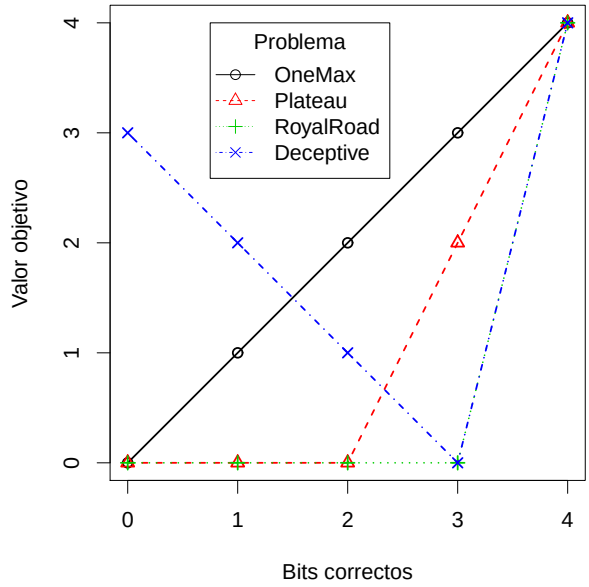

Figura 2.3: Contribución a la función de fitness (valor objetivo) para cada bloque de 4 bits de una solución, con respecto al número de bits acertados correctamente.

aciertan 4 se contribuye con 4. En cualquier otro caso, la contribución es 0.

- **RoyalRoad**: Solo se contribuye con 4 unidades al valor objetivo si se aciertan los 4 bits del bloque. En caso contrario, se contribuye 0.
- **Deceptive**: Si se aciertan los 4 bits, se contribuye con un valor de 4 a la función objetivo. En caso contrario se contribuye con la resta entre 3 y el número de bits correctos del bloque.

La Figura 2.3 muestra la contribución de cada bloque de 4 bits para cada una de las funciones base de acuerdo al número de bits acertados correctamente.

Para realizar los experimentos seleccionamos las cuatro variantes dinámicas de problemas estáticos con codificación binaria construidos utilizando el generador XOR-POD [141], comentadas anteriormente. A través de cambios en la máscara, el generador produce cambios en la posición óptima, y con el uso de diferentes valores de τ y ρ, se puede controlar la frecuencia y la severidad de los cambios, respectivamente. Altos valores de ρ implican cambios más severos, mientras que los valores bajos de τ significan cambios más frecuentes.

Se puede utilizar la idea de la máscara en cualquier problema con codificación binaria estático y se puede convertir en su versión dinámica. En la siguiente sección, se describe el otro problema seleccionado: la Mochila, que utiliza el generador XOR para introducir dinamismo en el problema.

Todas las configuraciones de los problemas se generaron con una dimensión de 100 (25 bloques de 4 bits). Se consideraron cinco frecuencias de cambio diferentes: $\tau \in \{1200, 3000, 6000, 9000, 12000\}$; y cuatro severidades: $\rho \in \{0.1, 0.2, 0.5, 0.9\}$, cada que tiempo cambia el problema y en cuanto cambia el problema, respectivamente. El τ seleccionado se puede traducir al número de evaluaciones de la función objetivo permitido entre los cambios consecutivos, es decir, representa el número de evaluaciones que el algoritmo tiene el objetivo de "alcanzar" el óptimo hasta el próximo movimiento.

Se realizaron 30 ejecuciones independientes para cada algoritmo, problema y combinación de τ y ρ, donde cada ejecución constaba de 100 cambios de la función objetivo. La métrica que se utilizó para evaluar el rendimiento de los algoritmos fue el offline performance, comentado anteriormente en el Capítulo 1 Sección 1.3.3.

Problema de la mochila

El problema de la mochila es un conocido problema de optimización combinatoria NP-Duro [66]. Dado un conjunto de m elementos el problema de la mochila se describe de la siguiente manera:

$$f(x) = \sum_{i=1}^{m} p_i x_i \tag{2.6}$$

$$\text{sujeto a} \sum_{i=1}^{m} w_i x_i \leq C \quad x_i \in \{0, 1\} \;\; i = 1 \ldots m \tag{2.7}$$

donde $x = (x_1, ..., x_m)$ y $x_i = 1$ si el objeto i es seleccionado o $x_i = 0$, en caso contrario. Los valores de p_i y w_i representan el beneficio y el peso de cada objeto i respectivamente, y C es la capacidad de la mochila. Se cree que la mochila es uno de los problemas más fáciles entre los NP. La dificultad de las instancias aumenta con la correlación entre pesos y beneficios [90].

La instancia de prueba que utilizamos tiene $m = 100$ objetos. Los pesos, beneficios y capacidad se definen como:

$$w_i = U(1, 50) \tag{2.8}$$

$$p_i = w_i + U(1, 5) \tag{2.9}$$

$$C = 0.6 * \sum_{i=1}^{m} w_i \tag{2.10}$$

donde $U(a, b)$ es una función que devuelve un valor aleatorio siguiendo una distribución

uniforme en el intervalo $[a, b]$. La definición de w_i y p_i conduce a una instancia con una fuerte correlación entre ambos valores. Como se indica en [90]:

Las instancias fuertemente correlacionados son difíciles de resolver por dos razones: *(a) están mal condicionadas en el sentido de que existe una gran diferencia entre la solución continua y entera del problema; (b) ordenar los elementos de acuerdo con la disminución de la eficiencia corresponde a un orden de acuerdo a los pesos. Por lo tanto, para cualquier intervalo pequeño de los elementos a analizar hay una variación limitada en los pesos, por lo que es difícil satisfacer la restricción de la capacidad.*

En el proceso de búsqueda pueden surgir soluciones no factibles del problema porque no cumplan con la restricción de capacidad, por lo cual se utiliza el mismo esquema de penalización propuesto por Yang y Yao en [141]:

$$f(x) = \begin{cases} \sum_{i=1}^{m} p_i x_i & \text{Si } C' \leq C \\ \\ 10^{-10} \cdot ((\sum_{i=1}^{m} w_i) - C') & \text{en otro caso} \end{cases} \tag{2.11}$$

donde $C' = \sum_{i=1}^{m} w_i x_i$

2.3.2. Composición del Portafolio

Para implementar los algoritmos que componen el portafolio de algoritmos utilizamos BiCIAM [38]. Un framework en Java que contiene versiones de las metaheurísticas más comunes y que se describieron en la Sección 1.2. Para la composición del portafolio se seleccionaron los siguientes métodos: Escalador de Colinas, Búsqueda Aleatoria, Recocido Simulado, Búsqueda Tabú, Algoritmos Genéticos, Estrategia Evolutiva y Algoritmo de Estimación de Distribuciones. Es importante señalar que seleccionamos métodos basados en trayectoria y algoritmos evolutivos, sin conocer a priori su desempeño en los PODs mostrado anteriormente. El objetivo fue tener un conjunto diverso de algoritmo con los comportamientos de búsqueda heterogéneos.

La Tabla 2.3 muestra los valores de los parámetros de los métodos que integran el portafolio de algoritmo. La versión actual de la biblioteca BiCIAM en Java, incluye las metaheurísticas mencionados anteriormente y la implementación del portafolio de algoritmos propuesto, se encuentra disponible en el siguiente enlace:

`http://modo.ugr.es/algorithmportfolio/index.html`

Método	Parámetros	Valor del Parámetro
Escalador de Colinas	tipo de ascenso	primer ascenso
	operador de vecindad	mutación en un punto
Recocido Simulado	temperatura inicial t_0	20
	temperatura final t_n	0
	número de iteraciones T	50
	α	0.93
	esquema de enfriamiento	$t_n = \alpha * t_0$
	operador de vecindad	mutación en un punto
Búsqueda Tabú	tamaño de la lista tabú	20
	contenido de la lista	soluciones
	operador de vecindad	mutación en un punto
Estrategia Evolutiva	tamaño de la población	50
	probabilidad de mutación	0.9
	operador de selección	truncamiento (20)
	operador de mutación	uniforme
Algoritmo Genético	tamaño de la población	50
	probabilidad de mutación	0.5
	probabilidad de cruzamiento	0.9
	operador de selección	truncamiento (20)
	operador de cruzamiento	uniforme
	operador de mutación	uniforme
Algoritmo de Estimación de Distribuciones UMDA	tamaño de la población	50
	operador de selección	truncamiento (20)
	modelo probabilístico	UMDA [107]

Tabla 2.3: Los métodos que integran el portafolio de algoritmos y la configuración y ajustes de sus parámetros.

2.3.3. Evaluación estadística de los resultados

En la actualidad se asume ampliamente que cualquier comparación entre un conjunto de algoritmos sobre un conjunto de problemas debe ser soportada por pruebas estadísticas. En este trabajo seguimos las directrices propuestas en [47] donde se sugieren las pruebas estadísticas no paramétricas en situaciones como la que enfrentamos: varios problemas, algoritmos y configuraciones.

En primer lugar, vamos a aplicar el test de Friedman [44] para comprobar si existen diferencias significativas entre un conjunto de algoritmos. Además el ranking promedio de Friedman permite ordenar los algoritmos en términos de rendimiento. Luego con el objetivo de evaluar si el mejor algoritmo en cada caso tiene un rendimiento significativamente diferente a los demás, se aplicaron los test post-hoc: Holm [61] y Finner [41], con un nivel de significación de $\alpha = 0.05$. Por último, si se detectan tales diferencias significativa, utilizamos el test de Wilcoxon [135] para comparaciones por pares de algoritmos.

Se utilizó el software comercial SPSS para los test: Friedman y Wilcoxon, y la herramienta KEEL [47] para los post-hoc Holm y Finner.

2.3.4. Evaluación de los resultados con el método de comparación SRCS

Los resultados para cada posible configuración de los algoritmos, instancia, severidad y frecuencia de cambio serían difícil de interpretar si se muestra como una tabla con valores numéricos. Por este motivo, además de las comparaciones con los test estádisticos no paramétricos se utilizó un esquema de ranking (SRCS) propuesto en [29], que sería otra variante para analizar los resultados.

SRCS utiliza tests no paramétricos (Kruskal-Wallis y Mann-Whitney Wilcoxon con corrección de Holm's para comparaciones múltiples) para evaluar la significación estadística de las diferencias individuales entre cada par de algoritmos en todas las configuraciones del problema. Si la prueba concluye que hay suficiente evidencia estadística de diferencia de rendimiento, el algoritmo con el mayor offline performance suma 1 a su ranking, y el otro suma -1. En caso de empate, ambos reciben un 0. El rango de los valores en el ranking de n algoritmos para cualquier problema es $[-n+1, n-1]$. Cuanto más alto sea el ranking obtenido, mejor se puede considerar un algoritmo en relación con los otros. A cada valor de ranking se asocia un color, utilizando blanco para el valor más alto del ranking $(n-1)$ y el valor más oscuro para el más bajo $(-n+1)$. Los colores para los valores intermedios se ajustan proporcionalmente.

Dada una instancia del problema, se construye un ranking para cada escenario depen-

diendo de la variación de frecuencia y severidad. Si agrupamos las gráficas de un algoritmo para un problema dado con cada combinación de frecuencia y severidad posible, podemos obtener una matriz de color, donde es fácil observar cómo el algoritmo funciona para ese problema específico. El color blanco en una celda dada indica que el algoritmo es estadísticamente mejor que todos los otros algoritmos para la configuración de un problema específico. Si el color es más oscuro significa que el algoritmo es estadísticamente igual o peor que otros algoritmos para esa configuración del problema.

2.4. Estudio Experimental

Los objetivos de la experimentación llevada a cabo son los siguientes:

- Analizar si el portafolio de algoritmos obtiene mejores resultados cuando usa aprendizaje, que variante de esquema de aprendizaje conduce a mejores resultados, y cómo influye el esquema de aprendizaje en la selección de los algoritmos. Los resultado se muestran en la Subsección 2.4.1.
- Analizar los resultados de la variante de portafolio que mejores resultados obtuvo frente a los algoritmos individuales que lo componen. Los resultado se muestran en la Subsección 2.4.2.
- Evaluar el rendimiento del portafolio de algoritmos con respecto a los algoritmos referencia: AHMA y SORIGA. Los resultado se muestran en la Subsección 2.4.3

Cada algoritmo se analiza para 5 problemas (OneMax, Plateau, RoyalRoad, Deceptive y Mochila), 4 niveles de severidad (ρ) y 5 de frecuencia de cambio (τ), es decir, que se hacen experimentos con un total de 100 escenarios.

2.4.1. Análisis del esquema de aprendizaje del portafolio

En este apartado vamos a analizar los mecanismos de aprendizaje que se presentaron en la Sección 2.1.1, con el objetivo de comprobar si su uso tiene sentido (si conducen a mejores resultados que un portafolio algoritmo sin aprendizaje), cuál de ellos obtiene el mejor rendimiento y cómo influye el esquema de aprendizaje en la selección de los algoritmos. Para simplificar el análisis de los resultados, aquí nos centraremos únicamente en los resultados de los test no paramétricos.

El lector interesado puede encontrar los valores de offline performance obtenidos por los diferentes métodos en la Tabla A.1 del Apéndice A.

	AP-NoLearn					
Esquema de Aprendizaje	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
RS-AP-RB	>	>	>	>	>	>
RS-AP-REB	–	>	>	>	–	–
RS-IP-RB	>	>	>	>	–	>
RS-IP-REB	<	–	–	<	<	>
NRS-AP-RB	<	<	<	–	–	–
NRS-AP-REB	>	>	>	>	–	<
NRS-IP-RB	>	>	>	>	–	>
NRS-IP-REB	<	<	<	<	<	>

Tabla 2.4: Aprendizaje vs. no aprendizaje. Los resultados de las comparaciones por pares, utilizando el test no paramétrico de Wilcoxon con un nivel significativo de α=0.05, entre los esquemas de aprendizaje propuestos y la variante del portafolio de algoritmo sin aprendizaje (AP-NoLearn). La primera columna indica el esquema de asignación de crédito previsto. La segunda columna indica el resultado de la comparación en todos los problemas y configuraciones. El resto de las columnas se corresponden con los resultados en cada problema. El signo '>' indica que el esquema considerado es estadísticamente mejor que la AP-NoLearn, '<' significa lo contrario, y '−' indica que no hay diferencias significativas.

2.4.1.1. Aprendizaje vs. No Aprendizaje

El objetivo de este primer análisis es comprobar si los esquemas de aprendizaje propuestos conducen al portafolio a obtener mejores resultados que una estrategia sin aprendizaje (AP-NoLearn), es decir, un portafolio en el que los métodos que lo constituyen tienen la misma probabilidad de selección a lo largo de la ejecución.

En primer lugar comparamos mediante el test de Wilcoxon con un nivel significativo de α=0.05, todos los esquemas de aprendizaje vs. AP-NoLearn. Los resultados se muestran en la Tabla 2.4. La primera columna indica el esquema de aprendizaje en consideración. La segunda columna "Global", es el resultado de la comparación en todos los problemas y configuraciones. El resto de las columnas se corresponden a los resultados de cada problema. El signo '>' indica que el esquema considerado es estadísticamente mejor que el AP-NoLearn, '<' significa lo contrario, y '−' indica que no hay diferencias significativas.

Considerando la columna "Global", sólo la mitad de los esquemas de aprendizaje tienen un desempeño significativamente mejor que AP-NoLearn, por lo cual es relevante la necesidad de ser cuidadoso en el diseño de un mecanismo de aprendizaje. En otras palabras, un mecanismo de aprendizaje malo pude conducir a peores resultados que un portafolio sin aprendizaje. Además no queda claro qué componente del esquema de aprendizaje tiene un mayor impacto en el rendimiento del portafolio, a pesar de que los esquemas que usan 'RB' (recompensa cuando la solución generada es estrictamente mejor que la solución referencia) proporciona mejores resultados.

Considerando los resultados desglosados por problema, para el caso de OneMax, Pla-

teau y RoyalRoad, son bastante similares al comportamiento de "Global". Además, los signos de la Tabla son casi iguales entre los tres problemas. Probablemente, lo que significa es que el portafolio se comporta de manera similar en estos tres problemas. Deceptive es quizás el problema más complejo, lo cual tiende a "confundir" el esquema de aprendizaje. Sólo la variante RS-AP-RB es capaz de obtener mejores resultados que la AP-NoLearn.

Para el caso del problema de la Mochila, tal vez el más cercano a los problemas de la vida real, el esquema de aprendizaje comienza a ser muy relevante, sólo en un caso de los ocho (NRS-AP-REB), el uso del aprendizaje obtuvo peores resultados que AP-NoLearn, mientras que otros siete casos permitieron obtener resultados mejores o iguales que el AP-NoLearn. Es importante señalar que RS-AP-RB es la única variante que supera a AP-NoLearn sobre todos los casos considerados.

2.4.1.2. Análisis de los esquemas de aprendizaje

En esta sección se van a comparar los diferentes esquemas de aprendizajes propuestos con el objetivo de determinar cual tiene un mejor comportamiento. Se va a analizar el mejor esquema tanto a nivel global (sobre todos los problemas), como en cada problema específico. Inicialmente se utilizó el test de Friedman donde se compararon todos los esquemas sobre todos los problemas y en cada problema. Para todos los casos, el test devolvió un $p - value = 0$, lo significa que se rechaza la hipótesis nula por lo cual existen diferencias significativas entre las variantes del portafolio cuando se utilizan diferentes esquemas de aprendizaje.

Como se comentó anteriormente, el test de Friedman propone un ranking promedio de los algoritmos que se comparan. Dicho ranking se muestra en la Figura 2.4 donde cada serie representa el ranking promedio para cada variante sobre cada uno de los problemas mencionados anteriormente. Con el fin de evaluar si el mejor esquema de aprendizaje en cada caso tiene un rendimiento significativamente diferente con los demás, se utilizaron los test post-hoc: Holm y Finner, con un nivel significativo de α=0.05.

La Tabla 2.5 muestra los resultados de los test de Holm y Finner, donde el símbolo '*' indica la variante de esquema de aprendizaje considerado como método de control (método con el mejor ranking promedio de acuerdo con el test de Friedman), mientras que '>' y '−' indican la existencia o no, respectivamente, de diferencias significativas entre el método de control y el mecanismo de aprendizaje correspondiente. En caso de que ambos test no proporcionen el mismo resultados (por ejemplo, el test de Holm no rechaza la hipótesis nula y Finner si lo hace), el resultado del test de Finner se muestra entre paréntesis.

La Figura 2.4 muestra que el esquema con mejor ranking de forma global es RS-AP-RB (reinicio de los créditos cuando ocurre un cambio, penalización activa, premio en el

Learning scheme	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
RS-AP-RB	*	*	*	*	*	−
RS-AP-REB	>	>	>	>	>	>
RS-IP-RB	>	>	>	>	−(>)	*
RS-IP-REB	>	>	>	>	>	>
NRS-AP-RB	>	>	>	>	>	>
NRS-AP-REB	>	>	>	>	>	>
NRS-IP-RB	>	>	>	>	−	−
NRS-IP-REB	>	>	>	>	>	−

Tabla 2.5: Resultados obtenidos por los test Holm y Finner, con un nivel significativo de α=0.05, para el mejor esquema de aprendizaje global (para todos los problemas) y en cada problema en concreto se compara con los demás. El símbolo '*' indica la variante de esquema de aprendizaje considerado como método de control (método con el mejor ranking promedio de acuerdo con el test de Friedman), mientras que '>' y '−' indican la existencia o no, respectivamente, de diferencias significativas entre el método de control y el mecanismo de aprendizaje correspondiente. En caso de que ambos test no proporcionen el mismo resultados (por ejemplo, el test de Holm no rechaza la hipótesis nula y Finner si lo hace), el resultado del test de Finner se muestra entre paréntesis.

crédito si se genera soluciones estrictamente mejores que la mejor solución disponible hasta ahora). Por otra parte, la diferencia en rendimiento con respecto a las otras variantes es significativo (Tabla 2.5). Por lo tanto, RS-AP-RB es estadísticamente el mejor esquema de aprendizaje sobre todos los problemas.

Si consideramos cada problema por separado, RS-AP-RB es la mejor variante del portafolio en cuatro de los cinco problemas, es decir, en todos menos en el problema de la Mochila. Volviendo a la Tabla 2.5, el algoritmo mejora significativamente al resto de los métodos en tres de estos cuatro problemas (OneMax, Plateu y RoyalRoad), mientras que en Deceptive la hipótesis nula no puede ser rechazada para los esquemas de aprendizaje, RS-IP-RB y NRS-IP-REB (o solo NRS-IP-REB si tenemos en cuenta el test post-hoc de Finner). En el problema de la Mochila, a pesar de no ser el mejor esquema de aprendizaje, RS-AP-RB no es significativamente peor que RS-IP-RB, el método de control en este caso.

Por último, se hizo un análisis más detallado de cada escenario en cada instancia utilizando la técnica SRCS que se describe en la Sección 2.3.4. Los resultados de las diferentes variantes del portafolio se muestran en la Figura 2.5. Para analizar los resultados, se debe comparar el color de la misma celda en las matrices de cada método para un determinado problema.

Por ejemplo, si queremos ver cuál es el mejor método para el problema OneMax con severidad 0.9 y frecuencia 1200, tenemos que comparar el color de la casilla correspondiente (la de la esquina superior izquierda) en las ocho matrices que muestran los resultados de cada método en la columna del problema OneMax. En este caso RS-AP-RB tiene el color blanco,

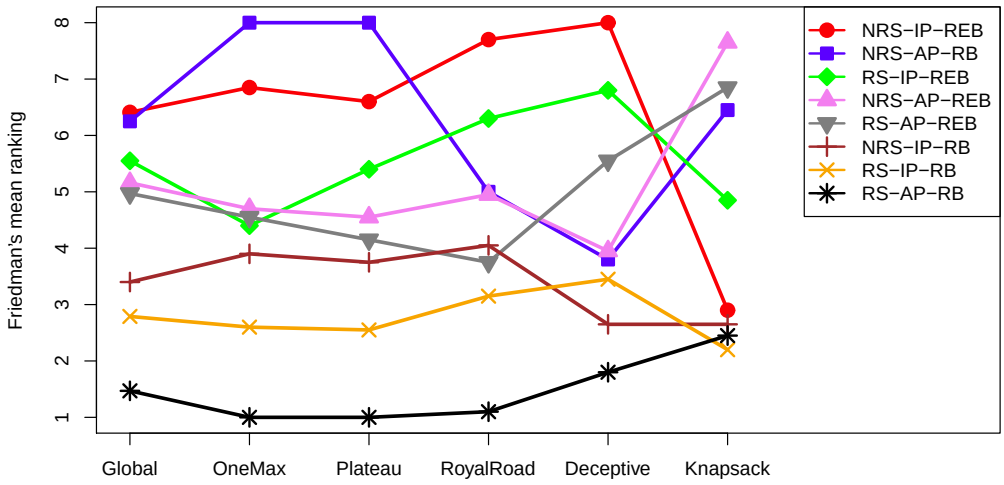

Figura 2.4: Ranking promedio proporcionado por el test de Friedman cuando se comparan todos los esquemas de aprendizaje. Cada serie representa el ranking promedio para cada esquema de aprendizaje en específico (eje Y), cuando se consideran todos los problemas (Global) y para cada problema (OneMax, Plateau, RoyalRoad, Deceptive y Mochila).

seguido por RS-IP-REB, RS-IP-RB, etc; de acuerdo a la escala de colores ($blanco \rightarrow rojo$), esto significa que para esa configuración de OneMax, RS-AP-RB es el mejor método, seguido por RS-IP-REB y el resto de las variantes según su escala de los colores, siendo NRS-AP-RB el que tiene un peor rendimiento.

Se puede apreciar en los resultados que muestra el ranking y los test, la variante RS-AP-RB supera o iguala al resto en los problemas: OneMax, Plateau y RoyalRoad. En el problema Deceptive tiene un buen comportamiento en promedio, igualado en la mayoría de las combinaciones con bajas frecuencias de cambio por las variantes NRS-AP-REB y NRS-IP-RB. En el caso de la Mochila también tiene un comportamiento bueno en promedio, siendo igualado en algunos casos por las variantes: RS-IP-RB, NRS-IP-REB y NRS-IP-RB. De forma general analizando los resultados en función de la frecuencia y severidad de cambio no se muestra claramente en la mayoría de los escenarios como podrían influir estos componentes en los diferentes algoritmos.

En resumen, este último análisis vuelve a confirmar que el rendimiento de RS-AP-RB es mejor o similar que los otros esquemas de aprendizaje, por lo que podemos concluir que en estos casos es la mejor variante del portafolio.

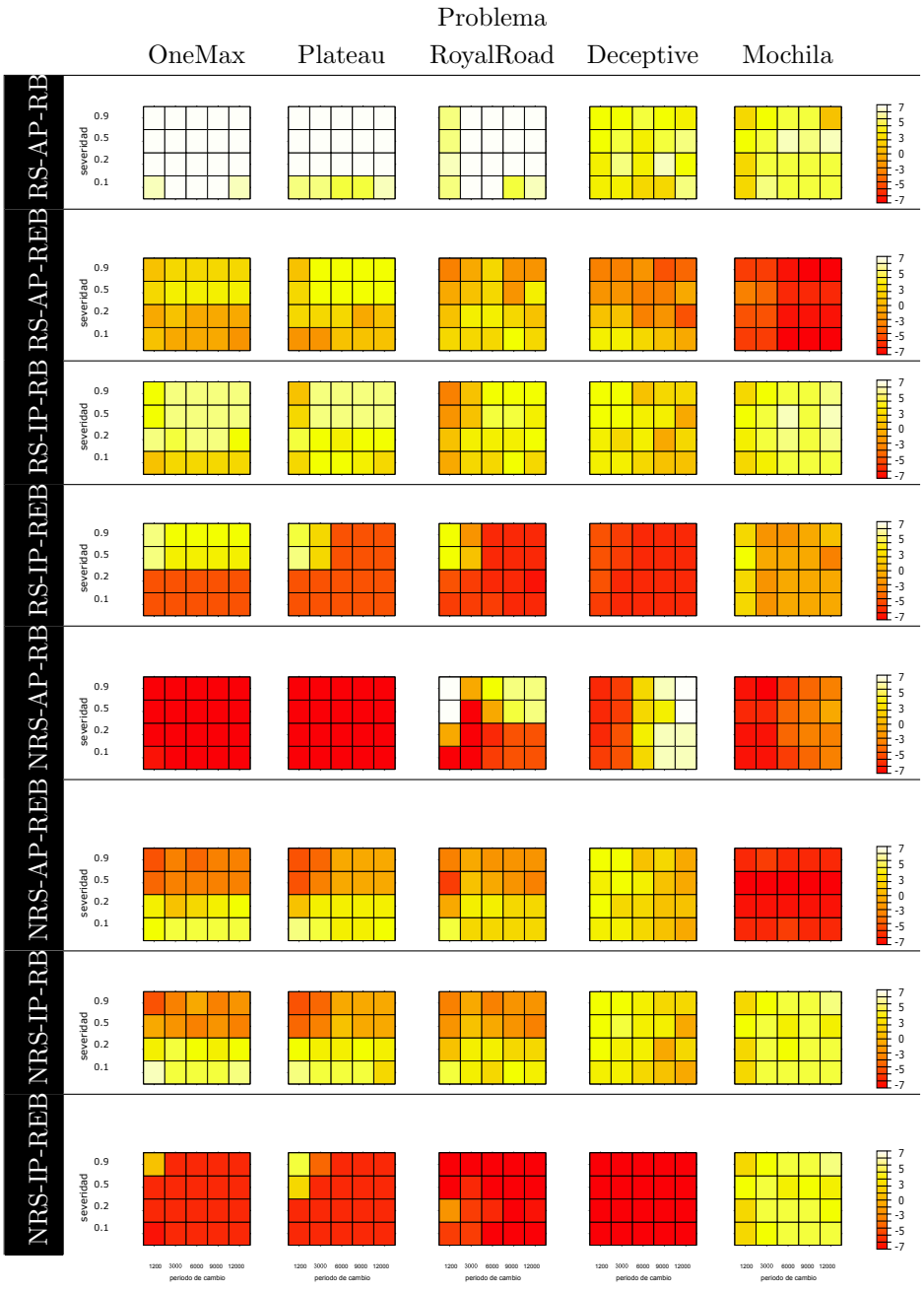

Figura 2.5: Resultados de ranking para las variantes del Portafolio de Algoritmos agrupadas por reinicio o no de los créditos. Se muestran los resultados con una matriz de dos niveles utilizando la tecnica SRCS. En el nivel superior, las columnas se corresponden con los problemas, y las filas con los algoritmos. En un nivel más inferior se encuentran las configuraciones de frecuencia de cambio y severidad de cambio respectivamente. Cada celda de estas matrices muestra el color obtenido por la técnica SRCS que corresponde al ranking de un algoritmo en una configuración del problema en específico.

2.4.1.3. Análisis de la influencia del esquema de aprendizaje en la selección de los algoritmos

Hasta el momento, nos centramos en evaluar el desempeño de los diferentes esquemas de aprendizaje. Esta sección tiene como objetivo principal estudiar como los esquemas de aprendizaje influyen en el comportamiento del portafolio de algoritmos. Específicamente se va a analizar como la selección de los algoritmos varían de un esquema de aprendizaje a otro, mediante un estudio de la evolución de las probabilidades de selección $P_{selec}^{a_i}$ de los algoritmos que componen el portafolio: Escalador de Colinas (HC), Búsqueda Aleatoria, (RndS), Recocido Simulado (SA), Búsqueda Tabú (TS), Algoritmo Genético (GA), Estrategia Evolutiva (EE) y Algoritmo de Estimación de Distribuciones (EDA).

Con fines ilustrativos seleccionamos dos esquemas de aprendizaje y problemas representativos. Concretamente, se seleccionó el mejor esquema de aprendizaje, RS-AP-RB, y su contraparte sin reinicio, NRS-AP-RB, con el objetivo de tener una visión clara de los efectos de reiniciar el crédito después de cada cambio. En cuanto a las configuraciones de problemas, hemos considerado los dos problemas más representativos desde el punto de vista práctico, RoyalRoad y Mochila, con valores intermedios para severidad y frecuencia de cambio, 0.5 y 6000, respectivamente.

La Figura 2.6 muestra la evolución de la probabilidad promedio de selección de cada algoritmo individual, medida cada 600 evaluaciones y acumulada en 30 ejecuciones, en los cinco primeros cambios de la función objetivo. Las filas y las columnas se corresponden a los esquemas de aprendizaje (RS-AP-RB y NRS-AP-RB) y configuraciones de los problemas (RoyalRoad (severidad 0.5, frecuencia de cambio 6000) y la Mochila (severidad 0.5, frecuencia de cambio 6000)), respectivamente. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en la que todos los algoritmos individuales tendrían la misma probabilidad de selección, mientras que las líneas verticales muestran cuando los cambios se producen.

En primer lugar, centramos el análisis en los escenarios del problema RoyalRoad. Para comprender mejor del análisis, es importante destacar que en este problema RS-AP-RB es significativamente mejor que NRS-AP-RB (según el test no paramétrico Mann-Whitney, con $\alpha < 0.05$); y el rendimiento de los algoritmos individuales en orden descendente ($mejor \rightarrow peor$) en función de la métrica offline performance es: HC, ES, GA, SA, RS, TS y EDA (se muestra en la Figura 2.7).

En los gráficos de la Figura 2.6 se puede observar claramente las diferencias entre reiniciar o no el crédito después de los cambios. En la variante NRS-AP-RB, las probabilidades convergen después del primer cambio a una distribución prácticamente uniforme, en el que todos los algoritmos tienen las mismas posibilidades de ser seleccionados. Sin embargo, en la variante RS-AP-RB, las probabilidades varían justo después de cada cambio y convergen

después de aproximadamente 3000 evaluaciones, es decir, en la mitad del periodo entre los cambios.

Centrándonos en los métodos individuales, en el periodo anterior al primer cambio, HC presenta la mayor probabilidad para ambos esquemas de asignación de crédito. En las etapas siguientes, observamos comportamientos muy interesantes en RS-AP-RB. La probabilidad de selección de HC llega a ser más cercana al valor de distribución uniforme, y sufren cambios bruscos: ES, SA y EDA, con altos valores en momentos similares de los periodos estacionarios de la función.

Teniendo en cuenta que SA y EDA no se encuentran entre los mejores métodos de rendimiento para escenarios y problemas cuando se ejecutan de forma individual, este comportamiento demuestra que a pesar de no tener un buen rendimiento individual pueden ser útil en algunos momentos de la búsqueda cuando se combinan con otros métodos. Esto también explica por qué "olvidar" lo aprendido después de cada cambio (reiniciar el crédito) es beneficioso en este caso, permite el aprovechamiento de esos algoritmos que sólo tienen un buen desempeño en partes específicas de la búsqueda.

La evolución de las probabilidades de selección para los escenarios de la Mochila es diferente pero mantiene algunas similitudes. En este caso, RS-AP-RB es también significativamente mejor que NRS-AP-RB (según el test no paramétrico Mann-Whitney, con $\alpha < 0.05$), y el rendimiento de los algoritmos individuales en orden descendente ($mejor \rightarrow peor$) en función de la métrica offline performance es: SA, HC, RS, TS, GA, EDA y ES.

Se debe destacar que tiene una mayor variación de las probabilidades de selección a lo largo de todo el periodo estacionario, a diferencia de RoyalRoad, donde las probabilidades tienden a la convergencia. Esto es debido a que el problema de la Mochila es más complejo, lo que relentiza la convergencia de los métodos y la difusión de las mejoras, y por lo tanto, los premios en los créditos, a lo largo de todo el proceso de búsqueda.

En el análisis de los métodos individuales, se observa que el TS y SA son los algoritmos que presentan una mayor probabilidad de selección para ambas variantes de portafolio. Es interesante ver que TS es uno de los dos algoritmos con la probabilidad de selección más alta a pesar de ser el cuarto mejor método de rendimiento cuando se ejecuta de forma individual. Esto sugiere de nuevo que los métodos individuales malos podrían ser útiles en alguna parte de la búsqueda cuando se combinan con otros algoritmos.

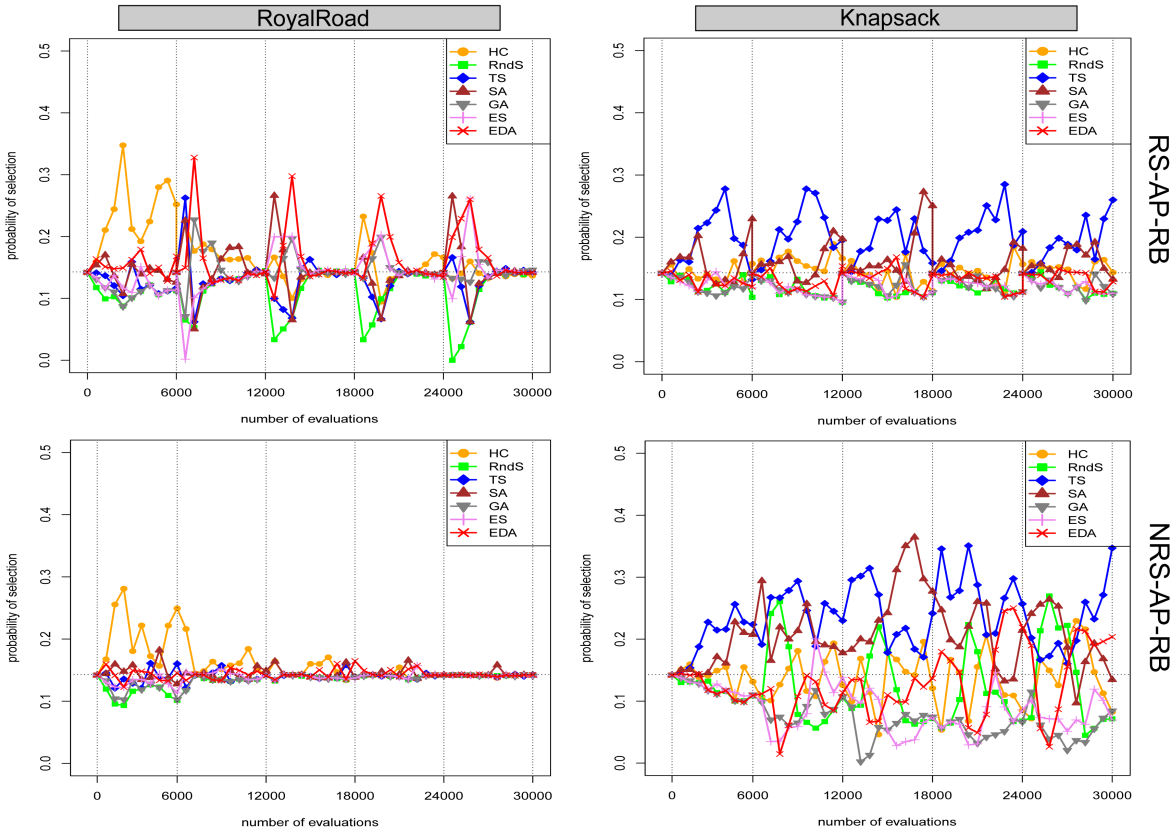

Figura 2.6: Evolución de la probabilidad promedio de selección de cada algoritmo individual, medida cada 600 evaluaciones y acumulada en 30 ejecuciones, en los cinco primeros cambios de la función objetivo. Las filas y las columnas se corresponden a los esquemas de aprendizaje (RS-AP-RB y NRS-AP-RB) y configuraciones de los problemas (RoyalRoad (severidad 0.5, frecuencia de cambio 6000) y la Mochila (severidad 0.5, frecuencia de cambio 6000)), respectivamente. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos individuales que tienen la misma probabilidad de selección, mientras que las líneas verticales muestran cuando los cambios se producen.

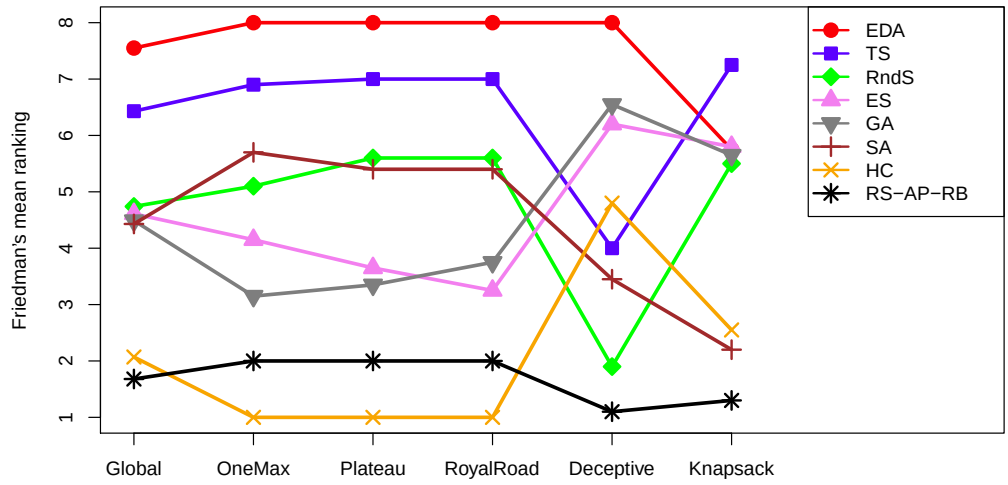

Figura 2.7: Ranking promedio proporcionado por el test de Friedman cuando se comparan todos los algoritmos. Cada serie representa el ranking promedio para cada algoritmo en específico (eje Y), cuando se consideran todos los problemas (Global) y para cada problema (OneMax, Plateau, RoyalRoad, Deceptive y Mochila).

2.4.2. Comparación con los métodos individuales

En la sección anterior se determinó el mejor esquema de aprendizaje (RS-AP-RB). Aquí comprobaremos si el portafolio es capaz de obtener mejores resultados que sus metaheurísticas individuales. Se compara la variante RS-AP-RB contra los métodos que componen al portafolio (HC, RndS, SA, TS, GA, EE y EDA). Se siguió la misma metodología empleada anteriormente en el apartado 2.4.1. En la Tabla A.2 del apéndice A se pueden encontrar los resultados obtenidos por cada uno de los algoritmos que se analizan en este apartado.

La Figura 2.7 muestra el ranking promedio que obtuvo el test no paramétrico de Friedman para todos los algoritmos individuales y el portafolio de algoritmos con la variante RS-AP-RB, a nivel global y en cada problema, respectivamente. El $p - value$ obtenido en todos los casos fue igual a 0 por lo que se puede rechazar la hipótesis nula, o sea, existen diferencias significativas entre los algoritmos.

Luego aplicamos los test post-hoc de Holm y Finner con un nivel significativo de α=0.05, con el objetivo de comprobar si la diferencia de rendimiento entre el mejor método obtenido por el ranking de Friedman es significativa o no con el resto. Estos resultados se muestran en la Tabla 2.6 (la notación utilizada es la misma que se comentó anteriormente).

De forma global sobre todos los problemas, RS-AP-RB mejora significativamente todos los métodos individuales excepto a HC (véase la Figura 2.7 y Tabla 2.6). En el análisis de los problemas por separado, se debe destacar la variabilidad en el rendimiento de las

Method	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
HC	−	∗	∗	∗	>	−
RndS	>	>	>	>	−	>
SA	>	>	>	>	>	−
TS	>	>	>	>	>	>
GA	>	>	>	>	>	>
ES	>	>	>	>	>	>
EDA	>	>	>	>	>	>
RS-AP-RB	∗	−	−	−	∗	∗

Tabla 2.6: Resultados obtenidos por los test Holm y Finner, con un nivel significativo de α=0.05, para la comparación entre el portafolio y los algoritmos individuales. El símbolo '*' indica la variante de esquema de aprendizaje considerado como método de control (método con el mejor ranking promedio de acuerdo con el test de Friedman), mientras que '>' y '−' indican la existencia o no, respectivamente, de diferencias significativas entre el método de control y el mecanismo de aprendizaje correspondiente. En caso de que ambos test no proporcionen el mismo resultados (por ejemplo, el test de Holm no rechaza la hipótesis nula y Finner si lo hace), el resultado del test de Finner se muestra entre paréntesis.

metaheurísticas, sobre todo cuando la dificultad de los problemas aumenta. Tres buenos ejemplos son HC, RndS y SA, los únicos métodos que no son significativamente peores que RS-AP-RB en todos los problemas. HC y RndS tienen un funcionamiento opuesto, mientras HC obtiene buenos resultados para OneMax, Plateau, RoyalRoad y Mochila, y peores en Deceptive, RndS se comporta muy bien en Deceptive y mal en los otros cuatro problemas.

En cuanto a SA, muestra un alto rendimiento para la Mochila y Deceptive, pero peor comportamiento en: OneMax, Plateu y RoyalRoad. Por el contrario, el portafolio presenta resultados muy buenos a lo largo de todos los problemas. En específico, para los dos más difíciles (Mochila y Deceptive) mejora todas las metaheurísticas individuales en términos del ranking promedio, aunque la hipótesis nula no puede ser rechazada con RndS en Deceptive, y para HC y SA en la Mochila.

En la Figura 2.8, se ilustran los resultados más detallados del ranking de todos los algoritmos para cada uno de los problemas en las diferentes configuraciones, como resultado de aplicar la técnica SRCS. Muestran como para los problemas que no son de tipo engañoso (Deceptive), un algoritmo mucho más sencillo, como es el Escalador de Colinas, que había sido ignorado hasta ahora en la investigación de PODs combinatorios, mejora en la mayoría de los casos a los restantes algoritmos.

Finalmente, a pesar de que HC tiene un mejor rendimiento que RS-AP-RB en OneMax, Plateau y RoyalRoad, podemos afirmar que la mejor variante del portafolio obtiene resultados similares o significativamente mejores (en al menos un problema) que las variantes aisladas de las metaheurísticas que lo integran.

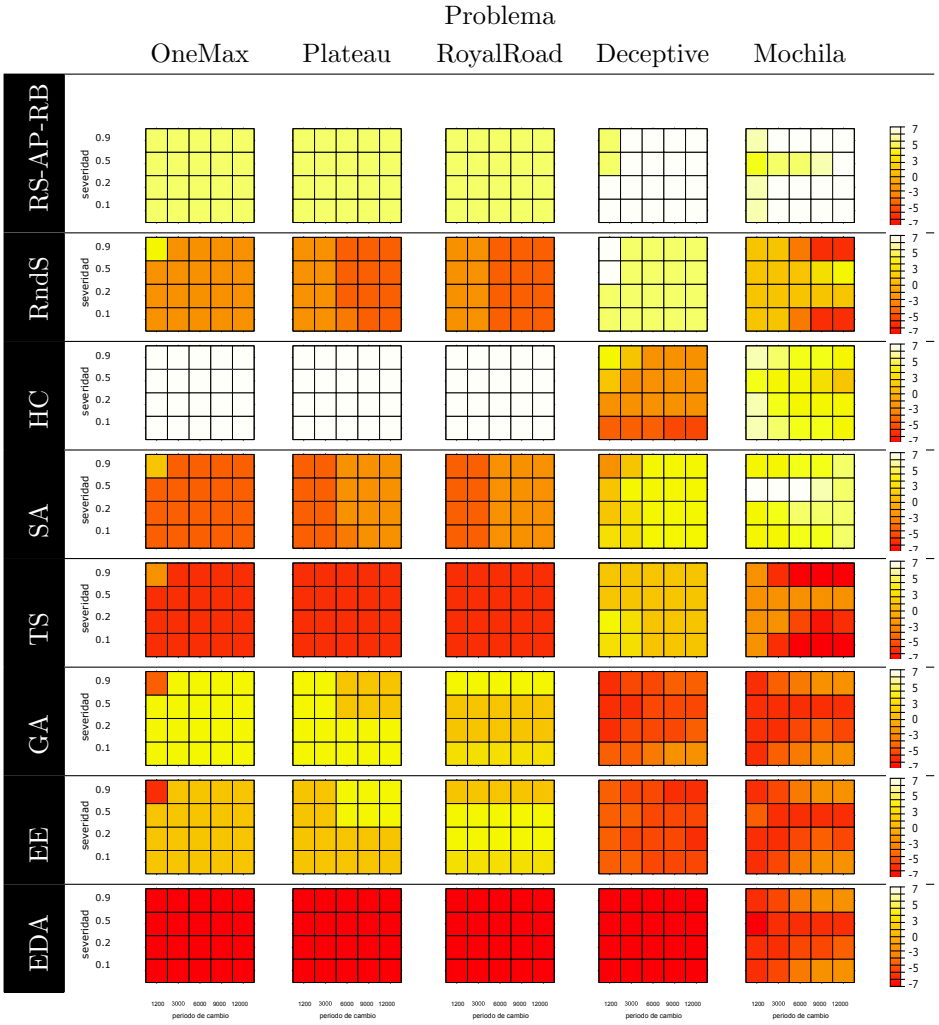

Figura 2.8: Resultados de ranking para los algoritmos individuales del portafolio. En la Sección 2.3.4 se detalla la explicación del método para obtener el ranking de los algoritmos.

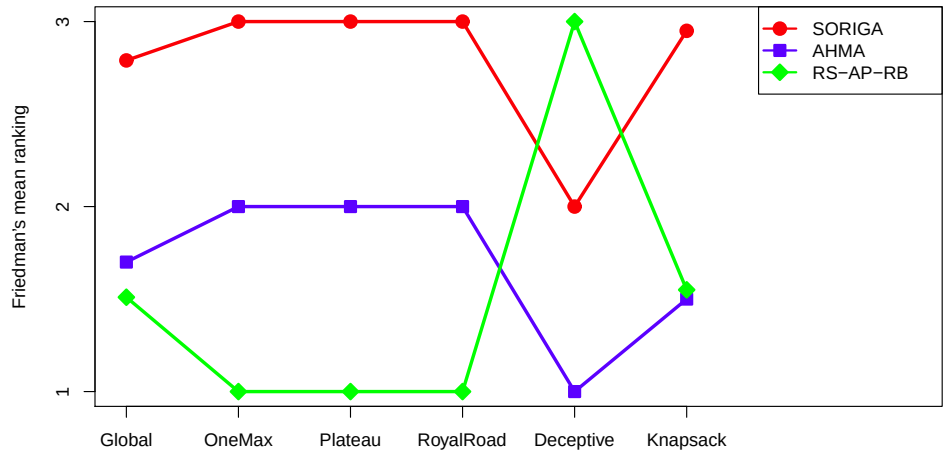

Figura 2.9: Ranking promedio proporcionado por el test de Friedman donde se compara la mejor variante del portafolio (RS-AP-RB) con los algoritmos SORIGA y AHMA. Cada serie representa el ranking promedio para cada algoritmo en específico (eje Y), cuando se consideran todos los problemas (Global) y para cada problema (OneMax, Plateau, RoyalRoad, Deceptive y Mochila).

2.4.3. Comparación con AHMA y SORIGA

En esta sección se va a comparar RS-AP-RB (variante de portafolio que mejores resultados obtuvo) contra SORIGA y AHMA (ambos descritos en la Sección 2.2), para comprobar si sus resultados son competitivos con algoritmos de alto rendimiento para estos problemas. Utilizamos el mismo estándar de comparación y el ajuste de parámetros de ambos métodos de acuerdo a los trabajos originales ([126] y [133], respectivamente). Para simplificar el análisis de los resultados, aquí nos centraremos únicamente en los resultados de los test no paramétricos. El lector interesado puede encontrar los resultados obtenidos por los algoritmos que se analizan en este apartado: RS-AP-RB, AHMA y SORIGA, en la Tabla A.3 del Apéndice A.

Se seguirá el mismo esquema de comparación utilizado en las dos secciones anteriores, es decir, vamos a utilizar el ranking promedio proporcionado por el test no paramétrico de Friedman, que rechaza la hipótesis nula en todos los casos ($p - value = 0$), y los test post-hoc de Holm y Finner con un nivel de confianza de α=0.95 con el objetivo de comprobar las diferencias significativas en el rendimiento entre los tres métodos. Estos resultados se muestran en la Figura 2.9 y la Tabla 2.7, respectivamente.

Al considerar todos los problemas y configuraciones (Global), el portafolio de algoritmos con la variante RS-AP-RB logra un rendimiento significativamente mejor que AHMA y SORIGA. Cuando desagregamos los resultados por problema, observamos que RS-AP-RB es la mejor alternativa en OneMax, Plateau y RoyalRoad, como se muestra en la Figura

Method	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
RS-AP-RB	*	*	*	*	>	–
AHMA	>	>	>	>	*	*
SORIGA	>	>	>	>	>	>

Tabla 2.7: Resultados obtenidos por los test Holm y Finner, con un nivel significativo de α=0.05, para la comparación entre el portafolio y los algoritmos AHMA y SORIGA. El símbolo '*' indica la variante de esquema de aprendizaje considerado como método de control (método con el mejor ranking promedio de acuerdo con el test de Friedman), mientras que '>' y '–' indican la existencia o no, respectivamente, de diferencias significativas entre el método de control y el mecanismo de aprendizaje correspondiente. En caso de que ambos test no proporcionen el mismo resultados (por ejemplo, el test de Holm no rechaza la hipótesis nula y Finner si lo hace), el resultado del test de Finner se muestra entre paréntesis.

2.9. Además, la hipótesis nula se rechaza en los tres casos, por lo que RS-AP-RB tiene un mejor comportamiento que AHMA y SORIGA (Tabla 2.7). En el caso del problema Deceptive, AHMA es significativamente el mejor método, mientras que para la Mochila, AHMA y RS-AP-RB lograron un rendimiento similar.

En la Figura 2.10 se muestra el ranking obtenido para cada uno de los escenarios como resultado de aplicar la técnica SRCS. Los resultados confirman el análisis anterior, e ilustran que RS-AP-RB para los problemas: OneMax, Plateau y RoyalRoad, muestra una dominancia clara, supera a los algoritmos del estado del arte en todas las configuraciones, exceptuando un empate entre RS-AP-RB y AHMA para la instancia de RoyalRoad con severidad de cambio 0.9 y frecuencia de cambio 6000, escenario en el que cambia mucho el problema. En el problema de la Mochila los resultados obtenidos ilustran que RS-AP-RB supera o iguala a los algoritmos del estado del arte en las variante con severidades de cambio: 0.5 y 0.9.

En este sentido, los resultados se pueden analizar en función de frecuencia y severidad de cambio, donde solo en el problema de la Mochila se puede observar claramente que RS-AP-RB con los valores altos de severidad tiene un mejor rendimiento, y AHMA lo contrario. Teniendo en cuenta la frecuencia de cambio, en la mayoría de los casos los algoritmos se comportaron de forma similar. Por otra parte, para el problema Deceptive, el RS-AP-RB es superado por el resto de los algoritmos, obteniendo los mejores resultados AHMA en todas las configuraciones. En general (sin tener en cuenta Deceptive, ya que tal vez es el más lejano a los problemas reales), se hace evidente que RS-AP-RB es mejor o igual que los algoritmos de referencia considerados.

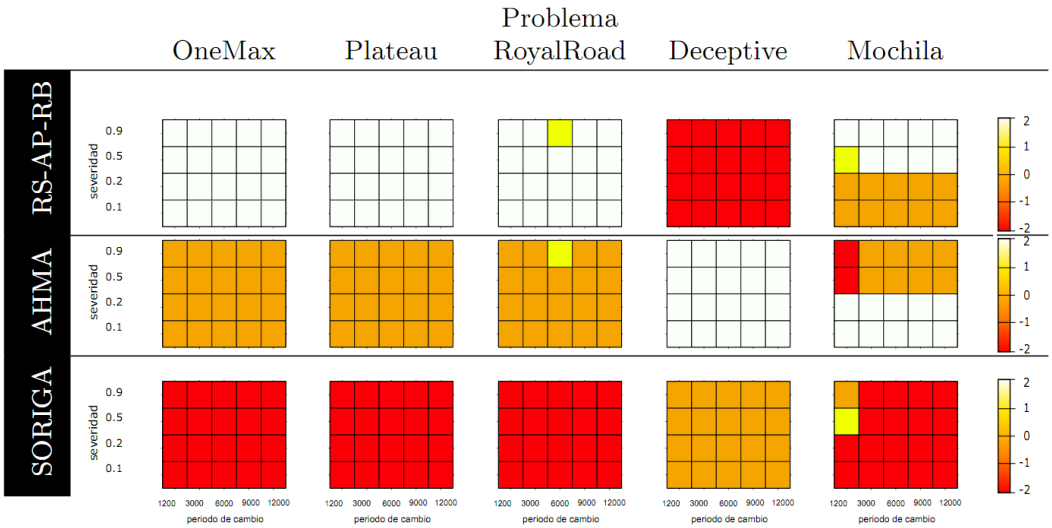

Figura 2.10: Resultados de ranking para los algoritmos: RN-P-M, AHMA, SORIGA. En la Sección 2.3.4 se detalla la explicación del método para obtener el ranking de los algoritmos.

2.5. Sumario

En este capítulo se ha presentado un portafolio de algoritmos para resolver problemas de optimización combinatoria dinámicos. Este método consiste en un conjunto de metaheurísticas que se ejecutan iterativamente. En cada etapa de la búsqueda, el portafolio selecciona qué metaheurística aplicar utilizando un enfoque basado en el crédito, que actúa como un esquema de aprendizaje.

El portafolio de algoritmos fue probado con cinco problemas de prueba (OneMax, Plateau, RoyalRoad, Deceptive y Mochila) a los que se les indujo dinamismo por medio del generador XOR-DOP. Para cada problema consideramos 4 severidades y 5 frecuencia de cambio diferentes. Para comparar los métodos, hemos empleado como medida de rendimiento el offline performance, y los test no paramétricos para comprobar las diferencias significativas entre los algoritmos.

La experimentación se orientó a comprobar: si portafolio algoritmo obtiene mejores resultados cuando se utiliza un esquema de aprendizaje que cuando no; qué esquema de aprendizaje proporciona mejores resultados; cómo en el esquema de aprendizaje influye la selección de los algoritmos; si el portafolio con el mejor esquema de aprendizaje mejoró el rendimiento de las metaheurísticas individuales que lo componen; y cómo el rendimiento del portafolio se comporta con respecto a dos métodos de referencia en la literatura, AHMA y SORIGA.

Después de analizar los resultados de la experimentación podemos extraer las siguientes conclusiones:

- El diseño adecuado del esquema de aprendizaje para el portafolio de algoritmos es una tarea fundamental, ya que sólo cuatro de los esquemas de ocho previstos en los problemas analizados resultaron mejor que la versión sin aprendizaje (AP-NoLearn) del portafolio.
- Los diferentes esquemas de aprendizaje conducen a diferentes patrones en la selección del algoritmo en cada momento de la búsqueda.
- Los algoritmos que tienen mal rendimiento cuando se ejecutan individualmente pueden ser útiles en momentos específicos de la búsqueda cuando se combinan con otros métodos.
- La variante RS-AP-RB (reinicio de los créditos cuando ocurre un cambio, penalización activa, premio en el crédito si se genera soluciones estrictamente mejores que la solución disponible) fue el único esquema de aprendizaje que tuvo mejores resultados que la variante AP-NoLearn del portafolio de algoritmo en los cinco problemas considerados.
- RS-AP-RB fue significativamente el mejor esquema de asignación de crédito a nivel global y en la mayoría de los problemas, a excepción de la Mochila. Se puede comprender que el rendimiento más bajo de RS-AP-RB en el problema de la Mochila, como un indicador de que la estructura del problema puede influir en el rendimiento del esquema de aprendizaje, porque hay algunas características más apropiadas que otras (por ejemplo, la penalización tiene un mejor rendimiento para OneMax, Plateau, RoyalRoad y Deceptive, y peor para el problema de la Mochila).
- Al considerar los resultados sobre todos los problemas, RS-AP-RB mejoró a todas las metaheurísticas individuales que lo integran, con diferencias significativas, excepto al Escalador de Colinas fue el de mejor rendimiento para los problemas: OneMax, Plateau y RoyalRoad. Como consecuencia, RS-AP-RB selecciona este método con una frecuencia más alta, por lo que al final alcanza un comportamiento como el Escalador de Colinas. La situación cambia en los problemas Deceptive y Mochila, donde las diferencias en el funcionamiento son menores entre los métodos individuales.
- RS-AP-RB obtuvo resultados muy similares en términos de posición en el ranking en todos los problemas, en contraste con la variabilidad en el rendimiento de las metaheurísticas individuales.
- Sobre todos los problemas, RS-AP-RB obtuvo resultados significativamente mejores que AHMA y SORIGA.

En términos generales, los resultados mostraron que la idea del portafolio de algoritmos también puede proporcionar buenos resultados en PODs. Un aspecto importante

a destacar es la extrema simplicidad del esquema de aprendizaje del portafolio y las metaheurísticas que lo integran. Se pudo ver como versiones muy generales y básicas de metaheurísticas comunes que trabajan juntas bajo un esquema sencillo de aprendizaje pueden proporcionar resultados competitivos con respecto a los métodos de alto rendimiento para resolver PODs. Estos resultados muestran que los portafolios de algoritmos son un buen paradigma para resolver no sólo los problemas estáticos, sino también para los dinámicos y por lo tanto, que se merecen una mayor atención en este campo.

Capítulo 3

Portafolio para el problema de máxima cobertura dinámico

En la actualidad la ubicación de instalaciones para satisfacer la demanda de servicios gana cada vez más interés. Durante las últimas décadas, una gran cantidad de técnicas de investigación de operaciones, herramientas y algoritmos se han aplicado a una amplia gama de problemas de localización. Una introducción detallada de los modelos de localización se puede encontrar en la referencia de ReVelle y Eiselt [101].

Entre los problemas de localización se encuentra el problema de máxima cobertura (MCLP), conocido como uno de los modelos clásicos de la literatura [18]. La mayoría de las publicaciones sobre el problema de localización MCLP abordan el caso en que el objetivo solo tiene en cuenta un periodo de tiempo. Sin embargo la variante de dinamismo o multiperiodo (DMCLP) es un modelo más real del problema, ya que considerando varios periodos de tiempo ayuda a los decisores a hacer planes en un rango de tiempo prolongado teniendo en cuenta fluctuaciones de la demanda.

Hasta donde conocemos, el DMCLP ha sido poco tratado y su última extensión ha sido propuesta por Zarandi y colaboradores en el 2013 [144], donde el problema es solucionado a gran escala (varios periodos de tiempo) con el algoritmo Recocido Simulado, siendo el más reciente que hemos encontrado que ha utilizado metaheurísticas para abordar este problema. Por ello, este capítulo tiene como objetivo resolver el DMCLP propuesto por Gunawardane en 1982 [58], con el portafolio de algoritmos diseñado para resolver problemas de optimización dinámicos presentado en el Capítulo 2. Pretendemos evaluar el comportamiento del portafolio de algoritmo sobre el DMCLP y comparar los resultados con los algoritmos que tienen reportados buenos resultados en la literatura.

El capítulo se organiza en los siguientes apartados. En la Sección 3.1 se realiza un revisión de la literatura sobre el problema DMCLP, como se ha abordado y los principales

métodos que se han utilizado para resolverlo. En la Sección 3.2 se describe la definición del problema. En las Secciones 3.3 y 3.4 se proponen dos variantes del problema DMCLP, incorporándole costos por apertura/cierre de instalaciones e instalaciones fijas a lo largo del tiempo, respectivamente. Luego, en la Sección 3.5 se comentan los enfoques utilizados para la creación de instancias del problema DMCLP de cara a la realización de experimentos. En la Sección 3.6, se describen los detalles de la experimentación, los algoritmos utilizados en las comparaciones y los test estadísticos no paramétricos empleados para la evaluación de los resultados. En la Sección 3.7, se realiza un análisis de los resultados obtenidos para cada variante del problema en los diferentes escenarios de experimentación. Por último, en la Sección 3.8 se presenta un breve resumen del capítulo.

3.1. Revisión de la literatura sobre el DMCLP

En este apartado se comentan los principales trabajos propuestos en la literatura que tratan el MCLP con algún tipo de dinamismo. Se realiza una descripción de las propuestas en orden cronológico, y los métodos de solución utilizados en los diferentes trabajos. El DMCLP tiene diferentes aplicaciones, entre ellas se conocen: localización de estaciones de patrullas de policía [17], localización y redistribución de ambulancias o unidades de emergencia [73], localización de instalaciones públicas [108], cámaras de seguridad de intersección en una red de tráfico urbano [122], instalaciones de cadenas de suministros [130].

Entre las primeras variantes de dinamismo se encuentra la de Schilling en el año 1980 [108], que propone un modelo multiobjetivo de localización dinámico (MODL). Se basa en MCLP y la diferencia principal entre ellos es que MODL tiene en cuenta un número de instalaciones a ubicar en cada periodo. En MODL se plantea que se pudiera analizar cada periodo por independiente y se persigue en cada uno un objetivo diferente. Trabajan con una única función objetivo que resulta de la suma de los objetivos de cada periodo. Realizan experimentos con 2 periodos de tiempo y utilizan como método de solución la programación entera.

Repede y Bernardo en el año 1994 [100] formularon un problema de localización multiperiodo maximizando la cobertura prevista en diferentes tiempos de viaje y tamaños de flota que varían en el tiempo (TIMEXCLP). En la propuesta de TIMEXCLP el método de solución que utilizaron fue la programación lineal, e integran el modelo y la solución en un sistema de apoyo a las decisiones. Luego se propuso un modelo para la reubicación dinámica (DDSM^t), desarrollado por Gendreau y colaboradores en el año 2001 [49], donde el objetivo consistía en maximizar la demanda a cubrir según el tiempo o la distancia, y minimizar los costos de reubicación. En esta propuesta utilizaron en la solución el algoritmo Búsqueda Tabú sobre la computación paralela.

Schmid y Doerner en el año 2010 [110], desarrollaron un modelo multiperiodo extensión de DSM (maximiza la demanda cubierta por al menos dos vehículos), donde tienen en cuenta que las áreas de cobertura varían con el tiempo y permiten que los vehículos vuelvan a colocarse para mantener un nivel de cobertura, a lo largo de la planificación. En la propuesta utilizan como método de solución el algoritmo de Búsqueda con Vecindad Variable. En ese mismo año Başar, Çatay y Ünlüyurt [5], modelaron el problema de múltiples periodos de localización de unidades de emergencia con doble cobertura para desarrollar un plan estratégico de varios periodos de tiempo. En el contexto de posicionamiento dinámico tuvieron en cuenta la reubicación/redistribución de los vehículos previendo regiones sin cubrir, y como método de solución utilizaron el algoritmo Búsqueda Tabú.

Tanaka en el año 2011 [122] propone un modelo de localización de instalaciones dinámico: Maximum Flow-Covering Location and Service Start Time Problem (MFCLSTP), se considera una extensión del modelo original de MCLP. Se asume que proporcionan instalaciones en horas fijas de servicio, y se considera que tanto las localizaciones como los servicios se pueden iniciar de manera tal que se maximice el flujo de cubrimiento. Se presentan dos variantes del problema: MFCLSTP1 (el tiempo de inicio del servicio de cada instalación es independiente) y MFCLSTP2 (todas las instalaciones comienzan en el mismo tiempo de servicio). En la solución del problema utilizaron la programación entera y un algoritmo de búsqueda local, diseñado específicamente para esta propuesta. El modelo fue aplicado y adaptado a la red del metro de ferrocarriles de Tokio, donde cada periodo de tiempo se trata de forma independiente.

Luego Schmid en el año 2012 [109] formuló un modelo dinámico estocástico para la planificación y envío de ambulancias, y el problema de la reubicación, aplicando como método de solución la programación dinámica aproximada. Con el fin de conseguir un modelo más cercano a la realidad, tuvo en cuenta explícitamente la información y variaciones dependientes del tiempo con respecto a los cambios en los volúmenes de solicitud y los tiempos de viaje, que varían a lo largo de un día.

Por último, recientemente Zarandi y colaboradores [144], a partir del modelo planteado por Church y ReVelle [18], redefinen la variante dinámica o multiperiodo. Trabajan sobre tres instancias del problema que se obtuvieron siguiendo el modelo propuesto en [102] donde se definen diferentes distribuciones para los datos (ubicación y demanda de los nodos), teniendo en cuenta cinco periodos de tiempo. Realizan experimentos con el algoritmo Recocido Simulado y la herramienta comercial CPLEX 12.1. Los resultados de ambas soluciones se comparan en función de la métrica tiempo y calidad de la mejor solución encontrada, obteniéndose que en las instancias de mayor tamaño el Recocido Simulado en menor tiempo alcanzó buenos resultados, ya que a medida que crece el problema, CPLEX necesita “mucho” tiempo para llegar a soluciones óptimas.

3.2. Definición del problema DMCLP

Entre los problemas de localización que han sido estudiados y modelados en la literatura se encuentran los problemas de cobertura, donde el principal interés se enfoca en cubrir un conjunto de demandas teniendo en cuenta diferentes objetivos a optimizar. Los dos tipos de problemas de cobertura, son el problema de localización de máxima cobertura (MCLP) [18] y el problema de localización de cobertura de conjuntos (SCLP) [58]. Independientemente del tipo de problema, una demanda está cubierta si se puede cubrir en un tiempo/distancia predefinido por una o más instalaciones. SCLP exige cubrir todos los nodos de demanda con el menor número posible de instalaciones, y MCLP se asocia a cubrir la máxima demanda posible con un número conocido de instalaciones.

MCLP fue introducido por primera vez por Church y ReVelle en el año 1974 [18]. El objetivo de su modelo era maximizar la cobertura de la población con recursos limitados. A continuación se detallan los parámetros y variables que definen el problema MCLP:

- Parámetros:
  - i, I: índice y conjunto de demanda de los nodos.
  - j, J: índice y conjunto de facilidades o instalaciones.
  - a_i: población o demanda de un nodo i.
  - d_{ij}: menor distancia (o tiempo) desde el nodo de demanda i hasta la facilidad o instalación j.
  - S: distancia (o tiempo) dentro de la cual un nodo se considera cubierto.
  - Ni: $\{j|d_{ij} \leq S\}$ es el conjunto de facilidades cuya distancia al nodo i es menor que S.
  - p: número de instalaciones a localizar.
- Variables:
  - X_j: $\{0, 1\}$ igual a 1 si la instalación esta situada en el nodo j, 0 caso contrario.
  - Y_i: $\{0, 1\}$ igual a 1 si el nodo i está cubierto por una o más instalaciones colocadas a una distancia S, 0 caso contrario.

Donde la función objetivo sería:

$$\text{maximizar } f(x) = \sum_{i \in I} a_i Y_i \tag{3.1}$$

Sujeta a la siguientes restricciones:

$$Y_i \leq \sum_{j \in N_i} X_j, i \in I \tag{3.2}$$

$$\sum_{j \in J} X_j = p \tag{3.3}$$

Entre las extensiones teóricas del modelo clásico de MCLP se encuentran las propuestas por Gunawardane en 1982 [58]. Entre las variantes presentadas se encuentra el modelo para maximizar la cobertura en multiples periodos y se conoce como dinámico (DMCLP), donde los decisores tienen como objetivo encontrar la manera óptima de ubicar instalaciones p en t periodos. En DMCLP, se toman algunas consideraciones, tales como: un nodo de demanda puede albergar una instalación, y cada nodo tiene una demanda, peso o utilidad. Además, aunque el número de instalaciones que se ubicarán se conoce a priori, no hay ninguna preferencia sobre el número de instalaciones que se ubicarán en cada periodo de tiempo.

Las variables y parámetros del modelo del problema DMCLP son extensión del modelo MCLP descrito anteriormente. A continuación se detallan los parámetros y variables que cambiaron para definir el problema DMCLP:

- Parámetros:
  - t, T: índice y conjunto de periodos de tiempo.
  - a_{it}: población o demanda de un nodo i en un periodo t.
  - p: número de instalaciones a localizar teniendo en cuenta todos los periodos de tiempo T.
- Variables:
  - X_{jt}: $\{0, 1\}$ igual a 1 si la instalación esta situada en el nodo j en el periodo t, 0 caso contrario.
  - Y_{it}: $\{0, 1\}$ igual a 1 si el nodo i en el periodo t está cubierto por una o más instalaciones colocadas a una distancia S, 0 caso contrario.

Se define como función objetivo:

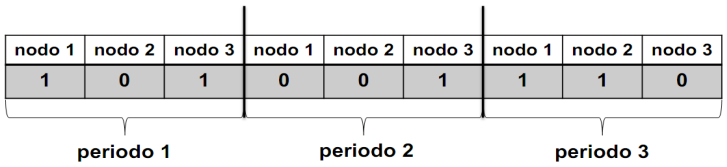

Figura 3.1: Representación de una solución para el problema DMCLP. El valor 1 significar cuales son las instalaciones disponibles en cada periodo de tiempo. En este ejemplo se tienen tres instalaciones, tres periodos de tiempo, y p es cinco.

$$\text{maximizar } f(x) = \sum_{t \in T} \sum_{i \in I} a_{it} Y_{it} \tag{3.4}$$

Sujeta a la siguientes restricciones:

$$Y_{it} \leq \sum_{j \in N_i} X_{jt}, i \in I, t \in T \tag{3.5}$$

$$\sum_{t \in T} \sum_{j \in J} X_{jt} = p \tag{3.6}$$

La restricción 3.5 prohíbe la cobertura de un nodo si no hay ninguna instalación en una distancia a menos de S a él, o sea, se cubrirá un nodo de demanda cuando al menos una instalación se encuentre a una distancia S o menor que S del punto i. La restricción 3.6 garantiza que el número de instalaciones que se establecerán en T periodos sea igual a p.

Se pueden emplear diferentes enfoques para representar una solución de DMCLP. Entre las más utilizadas se encuentra la representación de un vector binario en el que cada bit representa el estado de una instalación, donde el valor 1 significa que esa instalación se encuentra abierta o disponible y 0 en caso contrario. Esta es una representación esquemática de una solución para T periodos y se puede observar en la Figura 3.1. Está claro que debe haber exactamente p valores igual a 1 en cada cadena solución. El uso de la solución del vector binario, con los datos en una matriz de distancias o tiempos de viaje, se considera una modelación simple del problema.

3.3. DMCLP con costo por apertura y cierre de instalaciones

Sobre el problema clásico DMCLP surge el siguiente interrogante: tiene sentido que en los problemas del "mundo real" de un periodo a otro se cambien las instalaciones disponibles sin tener en cuenta un costo adicional. Teniendo en cuenta lo anterior, en este apartado proponemos una nueva variante de DMCLP conocida como: DMCLP con costo por apertura y cierre de instalaciones (DMCLP-AC). En esta variante tendríamos dos objetivos a optimizar: maximizar cobertura y minimizar costo apertura/cierre.

Se utilizó el enfoque más simple para modelar un problema multi-objetivo, conocido como factores ponderados [19], el cual consiste en multiplicar cada objetivo por un vector de peso y luego sumarlos. De esta forma, un problema multi-objetivo se transforma en un problema de un sólo objetivo. La principal ventaja de esta técnica es su simplicidad y su eficiencia. El principal inconveniente es el hecho de asignarle valores a los pesos, lo cual puede resultar difícil para el decisor.

Se define como función objetivo del problema DMCLP-AC:

$$\text{maximizar } f(x) = \alpha * f_1(x) + (1 - \alpha) * (1 - f_2(x)) \tag{3.7}$$

Los valores que se obtienen como resultado de evaluar en las funciones: $f_1(x)$ y $f_2(x)$ se encuentran en el intervalo [0,1]. La función $f_1(x)$ optimiza el objetivo definido en la ecuación 3.4, donde se maximiza la cobertura de los nodos de demanda. Para normalizar su valor al intervalo [0,1], dicho valor se divide por la demanda total del sistema. Se define como:

$$f_1(x) = \frac{\sum_{t \in T} \sum_{i \in I} a_{it} Y_{it}}{\sum_{t \in T} \sum_{i \in I} a_{it}} \tag{3.8}$$

La función $f_2(x)$ tiene como objetivo minimizar el costo de abrir y cerrar todas las instalaciones. Para normalizar el valor en [0,1], se dividen los costes de apertura y cierre por el valor del peor caso. Este caso correspondería con el escenario en el que todas las instalaciones se abren en los periodos impares y se cierran en los pares. La función $f_2(x)$ se define como:

$$f_2(x) = \frac{\sum_{j\in J}\sum_{t\in T} AP(x_{j,t}; x_{j,t+1}) * C_{AP_{j,t}} + \sum_{j\in J}\sum_{t\in T} CR(x_{j,t}; x_{j,t+1}) * C_{CR_{j,t}}}{\sum_{j\in J}\sum_{t\in T} AP_{j,t}CR_{j,t} * \lceil\frac{T}{2}\rceil} \tag{3.9}$$

donde:

- AP: función de abrir instalación j, $Ap : 0,1 \times 0,1 \rightarrow 0,1$ tal que: $AP(a,b) = 1$ si $a = 0$ y $b = 1$, y 0 en caso contrario.
- CR: función de cerrar instalación j, $C : 1,0 \times 1,0 \rightarrow 1,0$ tal que: $CR(a,b) = 1$ si $a = 0$ y $b = 1$, y 0 en caso contrario.
- C_{AP}: costo por apertura de una instalación.
- C_{CR}: costo por cierre de una instalación.

En la Figura 3.1 se muestra un ejemplo de una solución para el problema, donde se tiene: 101 001 110, para tres periodos de tiempo con cinco instalaciones a ubicar sobre todos los periodos. En este ejemplo se tendrían en cuenta las aperturas: $C_{AP_1} + C_{AP_3} + C_{AP_1} + C_{AP_2}$ y los cierres: $C_{CR_1} + C_{CR_3} + C_{CR_1} + C_{CR_2}$. En el caso de las aperturas en el primer periodo se abrieron las instalaciones 1 y 3, en el segundo periodo se mantuvo abierta la instalación 3 y se cerró la 1 (este periodo no aporta aperturas), y en tercer periodo se abrieron las instalaciones 1 y 2. En los cierres en el segundo periodo se cerró la instalación 1, y en el tercer periodo se cierra la instalación 3 que estaba abierta en el segundo periodo, y se deben cerrar las instalaciones 1 y 2 ya que es el último periodo.

3.4. DMCLP con localizaciones fijas

En esta sección se propone una nueva variante del problema de máxima cobertura dinámico o multiperiodo con instalaciones fijas a lo largo del horizonte temporal (DMCLP-LocF). La variante DMCLP-LocF consiste en que no estarían cambiando las instalaciones disponibles de un periodo a otro, seguramente se podría encontrar una distribución mejor pero no hay presupuesto para abrir y cerrar instalaciones. Esta variante tiene como principal objetivo encontrar una "solución robusta", o sea una solución que puede no ser la mejor para cada periodo pero que es lo suficientemente buena para tolerar los cambios del entorno. En la Figura 3.2 se ilustra un ejemplo de una solución que representa el problema donde no varían las instalaciones activas.

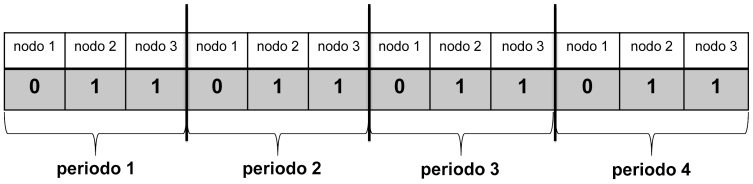

Figura 3.2: Representación de una solución para la variante DMCLP-LocF, el valor 1 significa cuales son las instalaciones disponibles en cada periodo de tiempo. En este ejemplo se tienen tres instalaciones, cuatro periodos de tiempo, y p es ocho.

La función del problema optimiza el objetivo definido en la ecuación 3.4, donde se maximiza la cobertura de los nodos de demanda. Se utilizan las mismas variables y parámetros definidos anteriormente en la Sección 3.2; y se añade una restricción para garantizar que se mantengan abiertas las mismas instalaciones en todos los periodos. Se define como:

$$X_{jt_1} = X_{jt_2} \forall t_1, t_2 \in T \tag{3.10}$$

3.5. Instancias disponibles en la literatura para la experimentación con DMCLP

En la literatura se han utilizado diferentes enfoques para la creación de instancias del problema DMCLP, entre ellas se conoce el enfoque propuesto en [6], donde se generaron las coordenadas [x, y] de cada nodo siguiendo una distribución uniforme entre 20 y 200, y para la matriz de distancia entre los nodos se utilizó la distancia Euclidiana. La demanda o el peso de cada uno de los nodos se generó siguiendo una distribución uniforme entre 10 y 100. Se crearon instancias con total de nodos N: 20, 50, 100, 200, 300 y 400; y el número de instalaciones o facilidades para cada instancia depende de N: 2 ($N = 20$), 5 ($N = 50$, 100), 10 ($50 \leq N \leq 200$), 20 ($50 \leq N \leq 300$), 30 ($N \geq 100$), 40 ($N \geq 100$), 50 ($N \geq 200$), 60 ($N \geq 300$), y 80 ($N = 400$).

Además en [64] se define otro enfoque para la creación de instancias para el problema MCLP, donde se generaron pares de datos que representan los puntos de demanda a partir de una distribución uniforme dentro de los intervalos [0, 50] y [0, 100]. Los datos para los sitios potenciales de las instalaciones se extrajeron de la misma distribución dentro de los intervalos [5, 45] y [10, 90]. Se supone que los sitios de instalaciones pueden cubrir aquellos puntos de demanda dentro de una distancia de 20 (S = 20). Los nodos de demanda van de 200 a 1000, con 2, 4 y 8 instalaciones que serán ubicadas. Los datos para los problemas con 40 sitios potenciales de instalaciones se generó con distribuciones uniformes dentro de los

rangos [0, 100], [0, 200] y [5, 95], [10, 190] para los puntos de demanda y los sitios potenciales de las instalaciones, respectivamente.

Luego en [102] se hace referencia a otro modelo, donde en primer lugar, las ubicaciones de los nodos se generaron aleatoriamente utilizando una distribución uniforme entre 0 y 30 para las dos coordenadas [x, y] y la distancia entre los nodos es la Euclidiana. Las demandas de cada uno de los nodos para cada periodo de tiempo se generaron aleatoriamente utilizando una distribución uniforme entre 0 y 100. Solo se trabaja con un periodo de tiempo, y se crean instancias hasta 900 nodos de demanda. Este último enfoque ha sido utilizado recientemente por Zarandi y colaboradores en el año 2013 [144], donde aplican el mismo procedimiento para problemas con: 1800, 2000 y 2500 nodos de demanda; con 100, 150 y 200 instalaciones; y cinco periodos de tiempo.

3.6. Marco de Experimentación

El objetivo de la experimentación realizada fue evaluar el comportamiento del portafolio de algoritmos frente al DMCLP y comparar los resultados con los algoritmos que han reportado buenos resultados en la literatura. En esta sección se describen los detalles de la experimentación, los algoritmos de referencia que utilizaron en las comparaciones y su configuración de parámetros, las instancias que se seleccionaron, y los escenarios que se definieron.

Las instancias de prueba para realizar los experimentos con el problema DMCLP se obtuvieron siguiendo el enfoque y los parámetros utilizados en [144], donde la ubicación de los nodos se generó aleatoriamente utilizando una distribución uniforme entre 0 y 30 para las coordenadas [x, y]. Las distancias entre los nodos se definió entonces como la distancia Euclidiana. Las demandas de cada uno de los nodos para cada periodo de tiempo se generaron aleatoriamente utilizando una distribución uniforme entre 0 y 100. Este procedimiento se utilizó para problemas con 1800, 2000 y 2500 nodos de demanda, con 100, 150 y 200 instalaciones, respectivamente. Consecuentemente para cada una de las instancias se generaron las demandas de los nodos para cinco periodos de tiempo. Se tuvieron en cuenta varios escenarios para la experimentación: distancia o tiempo $S = \{3, 4.5, 5\}$, y número de instalaciones a localizar $p = \{65, 70, 75, 80, 85\}$. Se realizaron 30 ejecuciones, con 10000 iteraciones por cada ejecución, que es la condición de parada de los algoritmos.

3.6.1. Consideraciones sobre el diseño de los algoritmos

En el estudio experimental se va a realizar un análisis del comportamiento del portafolio de algoritmos propuesto en el Capítulo frente a algunos de los algoritmos de referencia

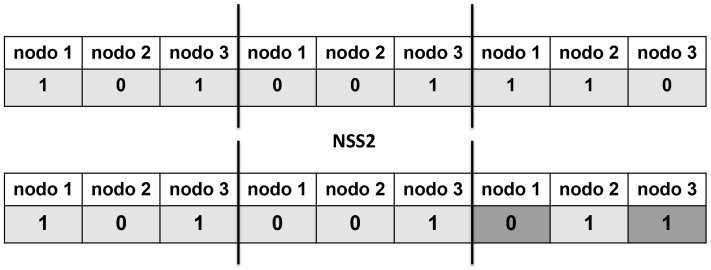

Figura 3.3: Ejemplo del funcionamiento del operador NSS2. Se basa en cerrar una instalación y abrir otra instalación aleatoriamente, las dos instalaciones que se intercambia son del mismo periodo (el cual se selecciona aleatoriamente). Se selecciona el periodo 3, se cierra la instalación del nodo 1 y se abre la instalación del nodo 3.

que se han utilizado para resolver el problema DMCLP. En la revisión de la literatura más reciente se muestra que se ha utilizado el Recocido Simulado [144]. Aunque existen otras metaheurísticas para resolver el DMCLP, las variantes de Recocido Simulado han reportado buenos resultados. Para las comparaciones de los resultados seleccionamos las variantes propuestas en [144]: Recocido Simulado Exponencial (RS-E), Recocido Simulado Hiperbólico (RS-H) y Recocido Simulado Lineal (RS-L).

Partiendo de los algoritmos de referencia en la literatura nos planteamos el siguiente interrogante: ¿Un portafolio de algoritmos compuesto por RS-E, RS-H y RS-L funcionará bien en el problema DMCLP? Teniendo en mente la pregunta anterior, seleccionamos como composición del portafolio a las tres variantes de Recocido Simulado (RS-E, RS-H y RS-L) y al algoritmo Estrategia Evolutiva (EE). Se decidió utilizar estos algoritmos con el objetivo de tener un conjunto de algoritmos con los comportamientos de búsqueda homogéneos y un componente poblacional que pudiera aumentar la diversidad en las soluciones que se generen. En la Tabla 3.1 se describe la configuración de parámetros utilizada para cada uno de los algoritmos que utilizaron.

El operador de vecindad o mutación que utilizaron los algoritmos se conoce como NSS2 en [144]. Este movimiento se basa en: se cierra una instalación y se abre otra instalación aleatoriamente, las dos instalaciones que se intercambia son del mismo periodo (el cual se selecciona aleatoriamente). El funcionamiento del operador NSS2 se muestra en la Figura 3.3, donde a partir de una solución se obtiene una solución vecina como resultado de aplicar el operador NSS2.

Para el caso de la variante del problema DMCLP-LocF, se utilizó una variante del operador NSS2. Su funcionamiento sería el siguiente: se cierra una instalación y se abre otra instalación del primer periodo, ambas seleccionadas aleatoriamente; después, se replica

Método	Parámetro	Valor del Parámetro
Recocido Simulado Exponencial	temperatura inicial t_0 [23]	$t_0 = \frac{\Omega}{ln\chi_0}$ Ω: deterioro promedio de la función objetivo χ_0: probabilidad de aceptación
	temperatura final t_f	0
	esquema de enfriamiento exponencial	$t_n = \frac{A}{n+1} + B$; $A = \frac{(t_0 - t_f)(N+1)}{N}$; $B = t_0 - A$
	operador de vecindad	NSS2[144]
Recocido Simulado Hiperbólico	temperatura inicial t_0 [23]	$t_0 = \frac{\Omega}{ln\chi_0}$ Ω: deterioro promedio de la función objetivo χ_0: probabilidad de aceptación
	temperatura final t_f	0
	esquema de enfriamiento hiperbólico	$t_n = \frac{1}{2}(t_0 - t_f)(1 - tgh(\frac{10n}{N} - 5)) + t_f$
	operador de vecindad	NSS2[144]
Recocido Simulado Lineal	temperatura inicial t_0 [23]	$t_0 = \frac{\Omega}{ln\chi_0}$ Ω: deterioro promedio de la función objetivo χ_0: probabilidad de aceptación
	temperatura final t_f	0
	esquema de enfriamiento lineal	$t_n = t_0 - n\frac{t_0 - t_f}{N}$
	operador de vecindad	NSS2 [144]
Estrategia Evolutiva	tamaño de la población	50
	probabilidad de mutación	0.9
	operador de selección	truncamiento (20)
	operador de mutación	NSS2 [144]

Tabla 3.1: Configuración de Parámetros para cada Algoritmo.

ese movimiento al resto de los periodos. De esta forma se garantiza mantener las mismas instalaciones disponibles en cada periodo de tiempo. En la Figura 3.4 se muestra un ejemplo del funcionamiento del operador NSS2 sobre DMCLP-LocF.

Además se utilizó para todos los algoritmos el procedimiento de inicialización propuesto en [144], el cual se basa en el operador de selección por ruleta, donde la asignación es proporcional a la contribución de las instalaciones al valor objetivo de la solución. Se diseñó una ruleta donde el area potencial de cada instalación está directamente relacionada con el total de demanda que puede cubrir. En otras palabras, la probabilidad de selección de cada instalación en un periodo específico de tiempo es proporcional al total de demanda cubierta por esa instalación en ese periodo en particular.

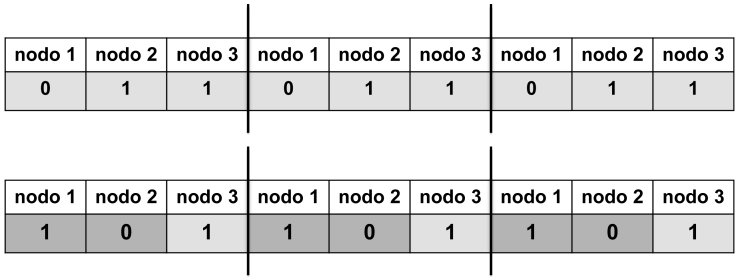

Figura 3.4: Ejemplo del funcionamiento del operador NSS2 para el problema DMCLP-LocF, donde no cambian las instalaciones disponibles de un periodo a otro. El movimiento se basa en cerrar una instalación y abrir otra instalación aleatoriamente, se abre la instalación del nodo 1 y se cierra la instalación del nodo 2, sobre en todos los periodos de tiempo.

3.7. Estudio Experimental

En el siguiente apartado se presentan los resultados obtenidos en la experimentación de las tres variantes del problema DMCLP y se realiza un análisis de las siguientes cuestiones:

- Analizar el rendimiento del portafolio de algoritmos con respecto a los resultados de referencia en la literatura.
- Analizar como influye la ejecución de cada componente aislado del portafolio, o sea comportamiento del portafolio frente a los algoritmos individuales.
- Analizar el funcionamiento interno del portafolio de algoritmos, evaluando la evolución de la probabilidad de selección de los algoritmos individuales y que componentes proveen mejoras.

3.7.1. Resultados DMCLP clásico

El principal objetivo de este apartado es evaluar el rendimiento del portafolio de algoritmos (PF) con respecto a los algoritmos individuales que lo componen, sobre la variante clásica del problema DMCLP. Inicialmente se utilizó el test de Friedman, donde se compararon los diferentes algoritmos para todos los problemas y en cada instancia por separado. En todos los casos el test devolvió un $p - valor = 0.0$, lo que indica que existen diferencias significativas entre los algoritmos. El test de Friedman también obtiene como salida un ranking promedio para cada método que se compara. Estos resultados se muestran en la Tabla 3.2 y se encuentra señalado en negritas el algoritmo con mejor ranking promedio. En todos los casos el algoritmo con mejor ranking promedio fue el portafolio de algoritmos. El lector interesado puede encontrar los resultados con los valores concretos del promedio de

Algoritmo	Global	Instancia-1800	Instancia-2000	Instancia-2500
PF	**1**	**1**	**1**	**1**
RS-E	2	2	2	2
RS-H	4	4	4	4
RS-L	5	5	5	5
EE	3	3	3	3
p-valor	0.00	0.00	0.00	0.00

Tabla 3.2: Resultado proporcionado por el test de Friedman, para cada algoritmo específico cuando se consideran todos los problemas (Global), y sobre todas las instalaciones y distancias. En todos los casos el algoritmo de mejor ranking se encuentra marcado en negrita.

la evaluación en la función objetivo de cada algoritmo en cada escenario para el problema DMCLP clásico, en el Apéndice B Sección B.1, en las Tablas B.4, B.5 y B.6.

Con el objetivo de evaluar si el mejor algoritmo en cada caso según el test de Friedman tiene un rendimiento significativamente diferente a los demás, se les aplicó los test post-hoc: Holm y Finner, con un nivel de significación de $\alpha = 0.05$. Los resultados obtenidos se muestran en la Tabla 3.3. El símbolo '*' indica el algoritmo considerado como método de control (el método con el mejor ranking promedio según el test de Friedman: Portafolio); '>' significa que el método de control mejora significativamente al método correspondiente; y '-' indica que no hay diferencias significativas entre los dos métodos. En todos los casos los test de Holm y Finner coincidieron en todos los resultados. Se puede apreciar que de forma general en el análisis global que existen diferencias significativas entre el portafolio (el método con el mejor ranking promedio según el test de Friedman) y el resto de los algoritmos. Cuando desagregamos por problemas entre el portafolio y RS-E no existen diferencias significativas. Lo que ocurre es que el portafolio es un poco mejor que los métodos individuales en cada una de las instancias, aunque no lo suficiente como para que esa diferencia sea significativa estadísticamente. Sin embargo, de forma global, cuando agrupamos todas esas mejoras, entonces las diferencias si que llegan a ser significativas.

Los resultados que evidencian los anterior se ilustran en la Figura 3.5, donde se muestran los gráficos de la evolución del porcentaje de demanda cubierta con respecto al total de demanda, para todos los escenarios: 1800 nodos de demanda y 100 nodos de instalaciones, 2000 nodos de demanda y 150 nodos de instalaciones, 2500 nodos de demanda y 200 nodos de instalaciones; y para cada algoritmo y parámetros: cobertura e instalaciones activas. Los valores concretos del porcentaje de demanda cubierta se encuentran en el Apéndice B Sección B.1, en las Tablas B.1, B.2 y B.3.

Se puede observar que en la mayoría de los casos el porcentaje de demanda tiende a aumentar en la medida en que aumenta el radio de cobertura y el total de instalaciones

Algoritmo	Global	Instancia-1800	Instancia-2000	Instancia-2500
PF	*	*	*	*
RS-E	>	–	–	–
RS-H	>	>	>	>
RS-L	>	>	>	>
EE	>	>	>	>

Tabla 3.3: Resultados obtenidos por los test post-hoc Holm y Finner, con un nivel de significación de α = 0.05 para el algoritmo mejor de forma global (en todos los problemas) y en cada problema concreto, se compara con los demás. El símbolo '*' indica el algoritmo considerado como método de control (el método con el ranking de la mejor media); '>' significa que el método de control mejora significativamente al método correspondiente; y '-' indica que no hay diferencias significativas entre los dos métodos.

activas. Además se puede apreciar claramente que en todos los casos el algoritmo con mejor rendimiento o sea el que obtiene mayor porcentaje de demanda cubierta es el portafolio, y en la mayoría de los escenarios el segundo mejor es el RS-E, lo cual corrobora el análisis realizado anteriormente.

Además se realizó un estudio de como evoluciona la probabilidad de selección $P_{selec}^{a_i}$ de los algoritmos que componen el portafolio: Recocido Simulado Exponencial (RS-E), Recocido Simulado Hiperbólico (RS-H), Recocido Simulado Lineal (RS-L) y Estrategia Evolutiva (EE). A modo de ejemplo se seleccionó una muestra de cada escenario con valores intermedios para cobertura e instalaciones a ubicar, 4.5 y 75, respectivamente. En la Figura 3.6 se muestra la evolución de la probabilidad promedio de selección de cada algoritmo individual, medida cada 500 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. Cada gráfico se corresponde con cada una de las instancias: 1800, 2000 y 2500 puntos de demanda. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos individuales, que al inicio del proceso de búsqueda tienen la misma probabilidad de selección.

Para comprender mejor el análisis, es importante destacar que en este problema RS-E es significativamente mejor que EE según los resultados anteriores, y el rendimiento de los algoritmos individuales en orden descendente ($mejor \rightarrow peor$) en función de la métrica promedio de la mejor solución es: RS-E, EE, RS-H y RS-L (como se muestra en la Tabla 3.2). En cuanto a los gráficos se puede observar que las probabilidades de selección no convergen en ningún caso, se van alternando indistintamente sobre todos los componentes. Teniendo en cuenta que RS-H y RS-L no se encuentran entre los mejores métodos de rendimiento para escenarios y problemas cuando se ejecutan de forma individual, este comportamiento demuestra que a pesar de no tener un buen rendimiento individual pueden ser útil en algunos momentos de la búsqueda cuando se combinan con otros métodos.

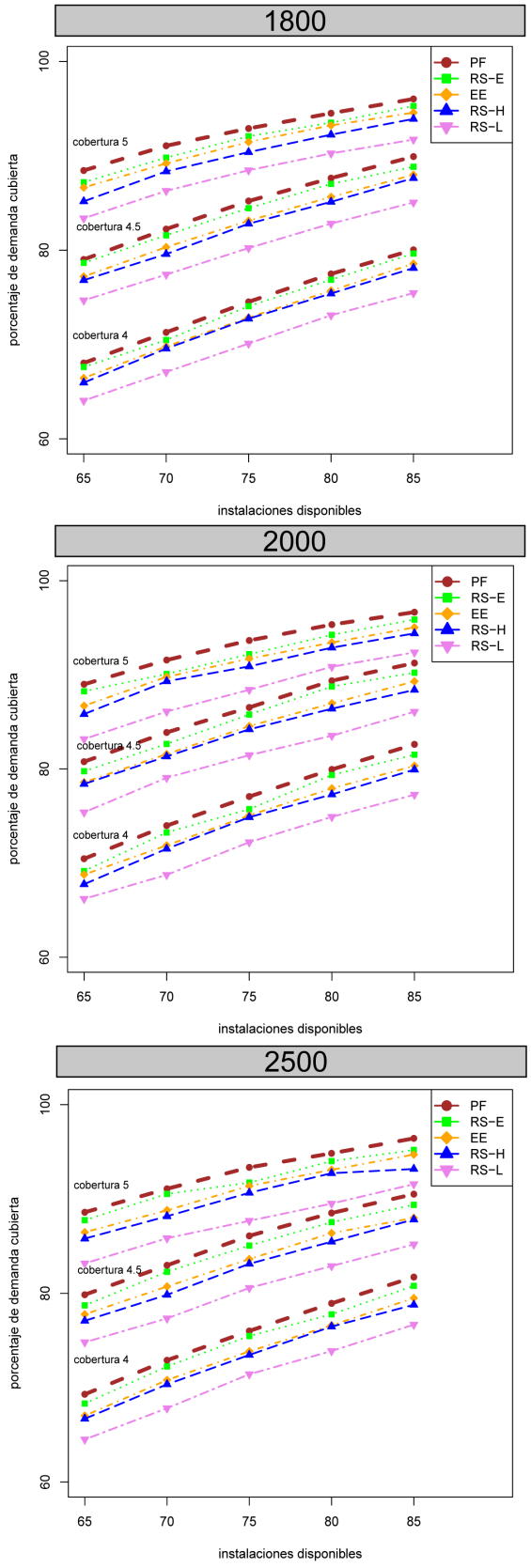

Figura 3.5: Evolución del porcentaje de demanda cubierta con respecto al total de demanda para todas las instancias, para el problema DMCLP clásico. Se representa cada algoritmo y escenario: cobertura e instalaciones activas.

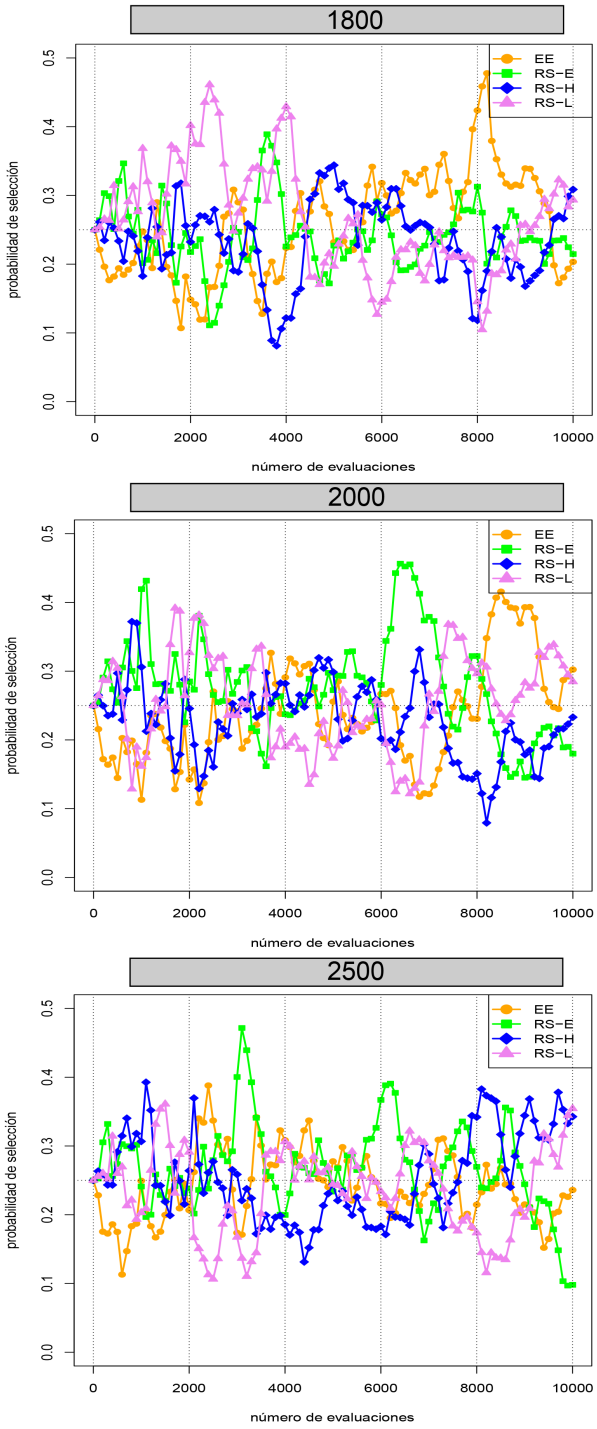

Figura 3.6: Evolución de la probabilidad promedio de selección de cada algoritmo individual para el problema DMCLP, medida cada 500 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos individuales que tienen la misma probabilidad de selección.

Algoritmo	Global	Instancia-1800	Instancia-2000	Instancia-2500
PF	**1**	**1**	**1**	**1**
RS-E	3	3	3	3
RS-H	4.5556	4.6667	4.4667	4.5333
RS-L	4.4444	4.3333	4.5333	4.4667
EE	2	2	2	2
p-valor	0.00	0.00	0.00	0.00

Tabla 3.4: Resultado proporcionado por el test de Friedman, para cada algoritmo específico cuando se consideran todos los problemas (Global), y sobre todas las instalaciones y distancias. En todos los casos el algoritmo de mejor ranking se encuentra marcado en negrita.

3.7.2. Resultados DMCLP con costo de apertura y cierre

En esta sección se evalúa el rendimiento del portafolio de algoritmos (PF) con respecto a los algoritmos individuales que lo componen sobre la variante del problema DMCLP que incluye costo por apertura o cierre de instalaciones. Inicialmente se utilizó el test de Friedman, donde se compararon los diferentes algoritmos para todas las instancias, y en cada instancia por separado. En todos los casos el test devolvió un $p - valor = 0.0$, lo que indica que existen diferencias significativas entre los algoritmos. El test de Friedman también obtiene como salida un ranking promedio para cada método que se compara. Estos resultados se muestran en la Tabla 3.4 y se encuentra señalado en negritas el algoritmo con mejor ranking promedio. En todos los casos el algoritmo con mejor ranking promedio fue el portafolio de algoritmos.

El lector interesado puede encontrar los resultados con los valores concretos del promedio de la evaluación en la función objetivo de cada algoritmo en cada escenario para el problema DMCLP con costo por apertura o cierre de instalaciones, en el Apéndice B Sección B.2, en las Tablas B.10, B.11 y B.12.

Con el objetivo de evaluar si el mejor algoritmo en cada caso según el test de Friedman tiene un rendimiento significativamente diferente a los demás, se les aplicaron los test post-hoc: Holm y Finner, con un nivel de significación de $\alpha = 0.05$. Los resultados obtenidos se muestran en la Tabla 3.5. El símbolo '*' indica el algoritmo considerado como método de control (el método con el mejor ranking promedio según el test de Friedman: Portafolio); '>' significa que el método de control mejora significativamente al método correspondiente; y '-' indica que no hay diferencias significativas entre los dos métodos. En todos los casos los test de Holm y Finner coincidieron en los resultados.

En los resultados obtenidos, el portafolio (método con el mejor ranking promedio según el test de Friedman) es un poco mejor que los métodos individuales, por lo tanto de

Algoritmo	Global	Instancia-1800	Instancia-2000	Instancia-2500
PF	*	*	*	*
RS-E	>	>	>	>
RS-H	>	>	>	>
RS-L	>	>	>	>
EE	>	–	–	–

Tabla 3.5: Resultados obtenidos por los test post-hoc Holm y Finner, con un nivel de significación de $\alpha = 0.05$ para el algoritmo mejor de forma global (en todos los problemas) y en cada problema concreto, se compara con los demás. El símbolo '*' indica el algoritmo considerado como método de control (el método con el ranking de la mejor media); '>' significa que el método de control mejora significativamente al método correspondiente; y '-' indica que no hay diferencias significativas entre los dos métodos.

forma global el portafolio es estadísticamente mejor que el resto de los algoritmos. Cuando descomponemos el análisis por problemas, entre el portafolio y la Estrategia Evolutiva no existen diferencias significativas estadísticamente, o sea las mejoras no son suficientes como para producir diferencias. El segundo mejor algoritmo es la Estrategia Evolutiva, sin embargo en el caso del problema DMCLP clásico la Estrategia Evolutiva siempre estuvo en tercer lugar del ranking.

Los resultados que evidencian lo anterior se ilustran en las Figuras 3.7, 3.8 y 3.9, donde se muestran los gráficos de la evolución del porcentaje de demanda cubierta con respecto al total de demanda, para todos los escenarios: 1800 nodos de demanda y 100 nodos de instalaciones, 2000 nodos de demanda y 150 nodos de instalaciones, 2500 nodos de demanda y 200 nodos de instalaciones, desagregando en cada instancia por el umbral de cobertura S (4, 4.5, 5). Los valores concretos del porcentaje de demanda cubierta se encuentran en el Apéndice B Sección B.2, en las Tablas B.7, B.8 y B.9.

En todos los casos el portafolio obtuvo mejores resultados que el resto de los algoritmos. Además, a medida en que aumenta el radio de cobertura y el número de instalaciones a localizar, aumenta el porcentaje de demanda cubierto por cada algoritmo. En este sentido, se puede apreciar claramente que los algoritmos con mejores rendimientos son el portafolio de algoritmos y la Estrategia Evolutiva, lo cual corrobora el análisis realizado anteriormente.

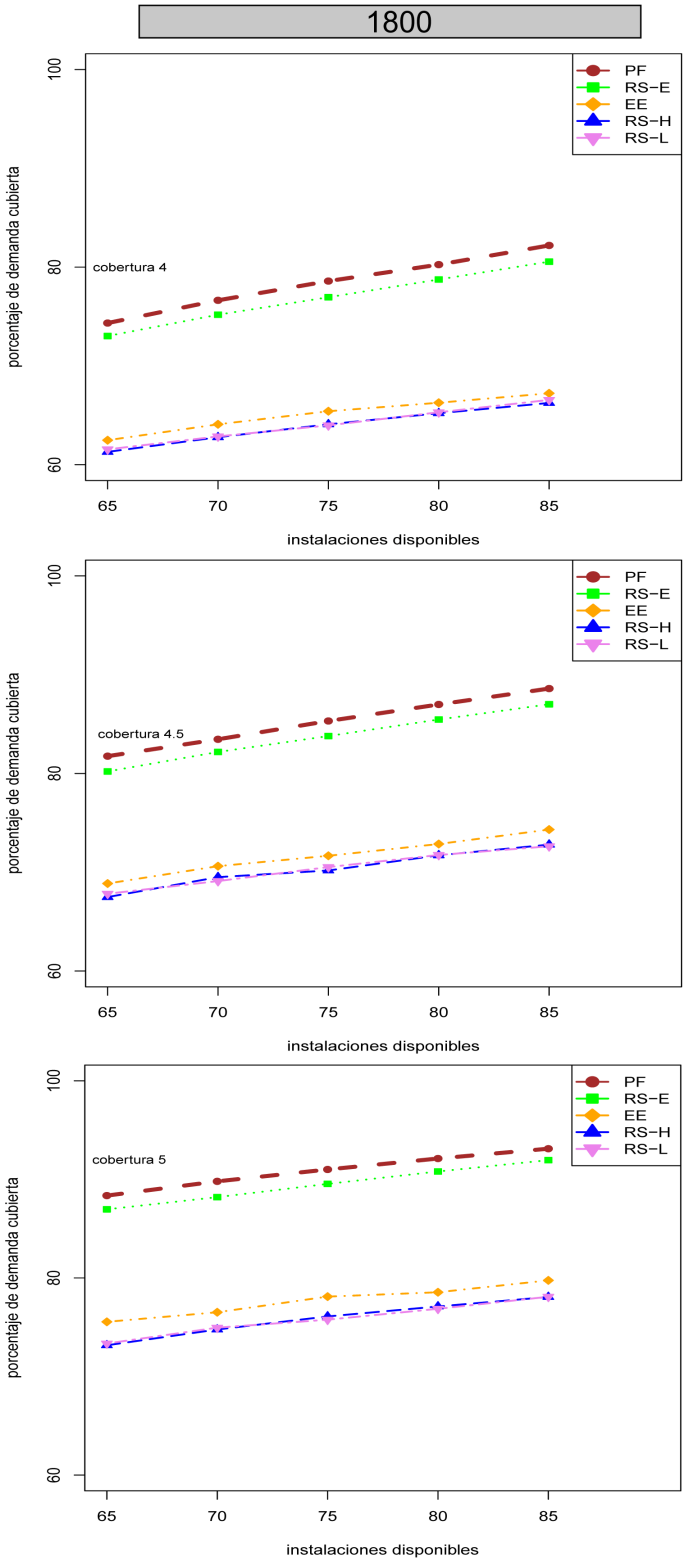

Figura 3.7: Evolución del porcentaje de demanda cubierta para la instancia de 1800 nodos de demanda, con respecto al total de demanda para todas las instancias, para el problema DMCLP-AC. Se representa cada algoritmo y escenario: cobertura e instalaciones activas.

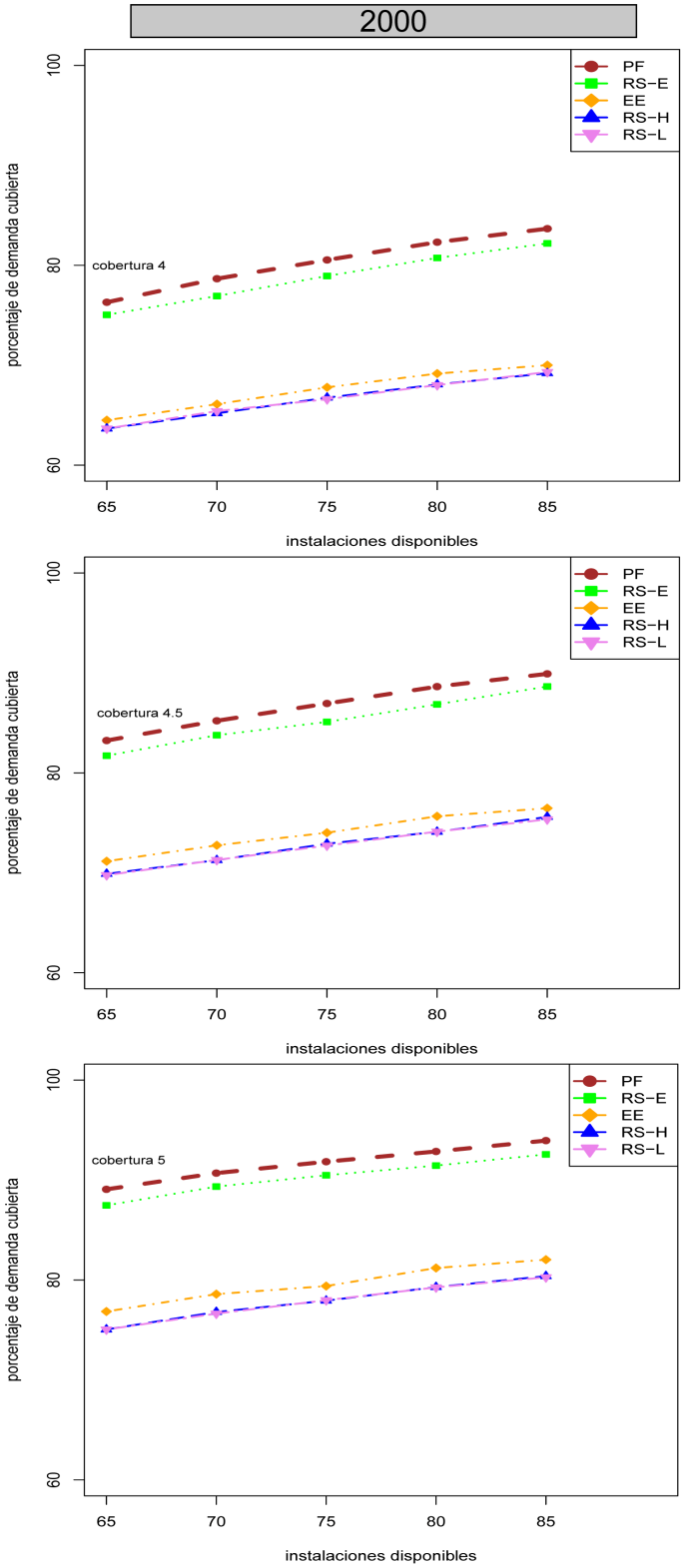

Figura 3.8: Evolución del porcentaje de demanda cubierta para la instancia de 2000 nodos de demanda, con respecto al total de demanda para todas las instancias, para el problema DMCLP-AC. Se representa cada algoritmo y escenario: cobertura e instalaciones activas.

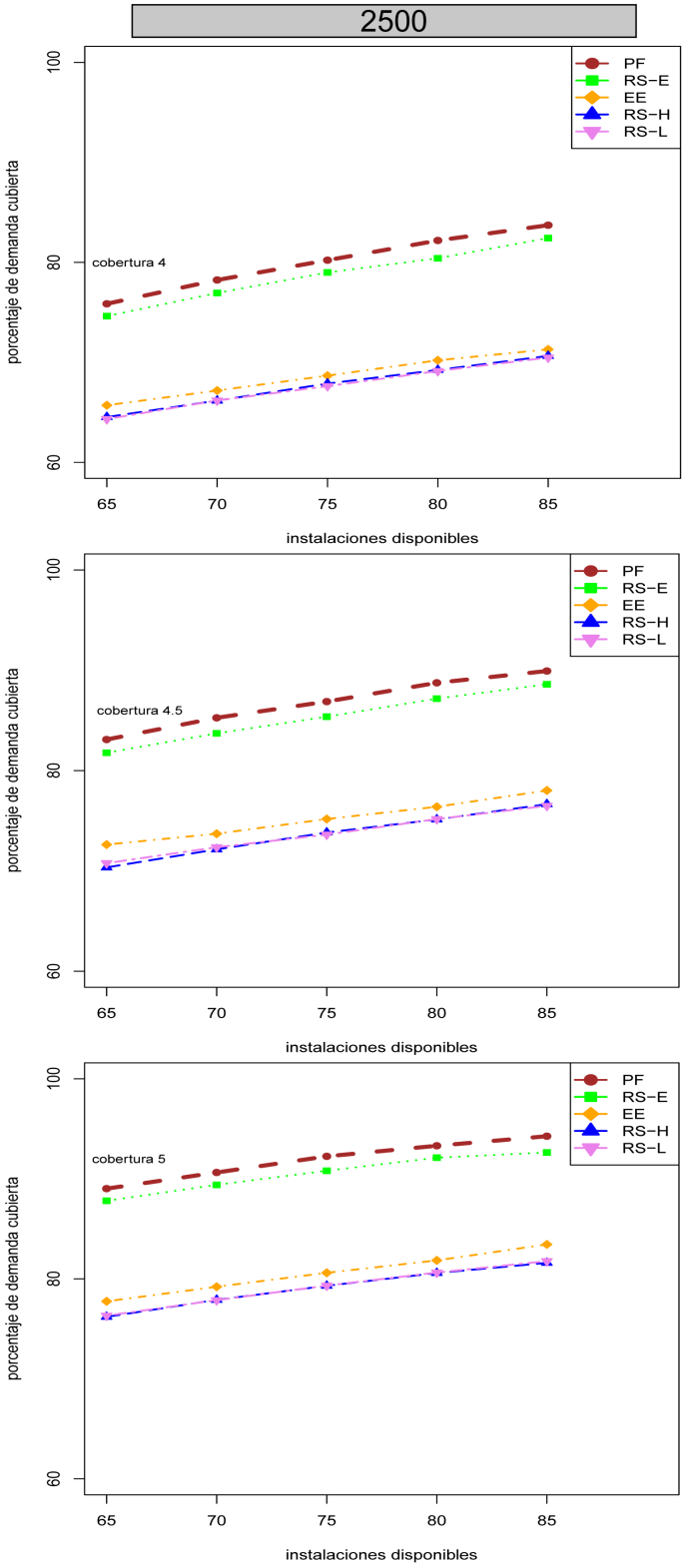

Figura 3.9: Evolución del porcentaje de demanda cubierta para la instancia de 2500 nodos de demanda, con respecto al total de demanda para todas las instancias, para el problema DMCLP-AC. Se representa cada algoritmo y escenario: cobertura e instalaciones activas.

Además se realizó un estudio de como evoluciona la probabilidad de selección $P_{selec}^{a_i}$ de los algoritmos que componen el portafolio: Recocido Simulado Exponencial (RS-E), Recocido Simulado Hiperbólico (RS-H), Recocido Simulado Lineal (RS-L) y Estrategia Evolutiva (EE). A modo de ejemplo se seleccionó se seleccionó una muestra de cada escenario con valores intermedios para cobertura e instalaciones a ubicar, 4.5 y 75, respectivamente.

La Figura 3.10 muestra la evolución de la probabilidad promedio de selección de cada algoritmo individual, medida cada 500 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. Cada gráfico se corresponde con cada una de las instancias: 1800, 2000 y 2500 puntos de demanda. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos individuales, que al inicio del proceso de búsqueda tienen la misma probabilidad de selección.

Para comprender mejor el análisis, es importante destacar que en este problema la EE es significativamente mejor que RS-E según los resultados anteriores, y el rendimiento de los algoritmos individuales en orden descendente ($mejor \rightarrow peor$) en función de la métrica promedio de la mejor solución es: EE, RS-E, RS-L y RS-H (como se muestra en la Tabla 3.4).

En cuanto a los gráficos se puede observar claramente las diferencias en las probabilidades, donde los algoritmos EE y RS-E presentan las mayores probabilidades de selección para las tres instancias del problema. En el caso de los algoritmos RS-H y RS-L convergen a bajos valores de probabilidad de selección, con una tendencia muy similar. Este comportamiento demuestra que el portafolio en este problema está combinando los algoritmos de buen rendimiento, teniendo en cuenta que EE y RS-E son los mejores métodos de rendimiento cuando se ejecutan de forma individual.

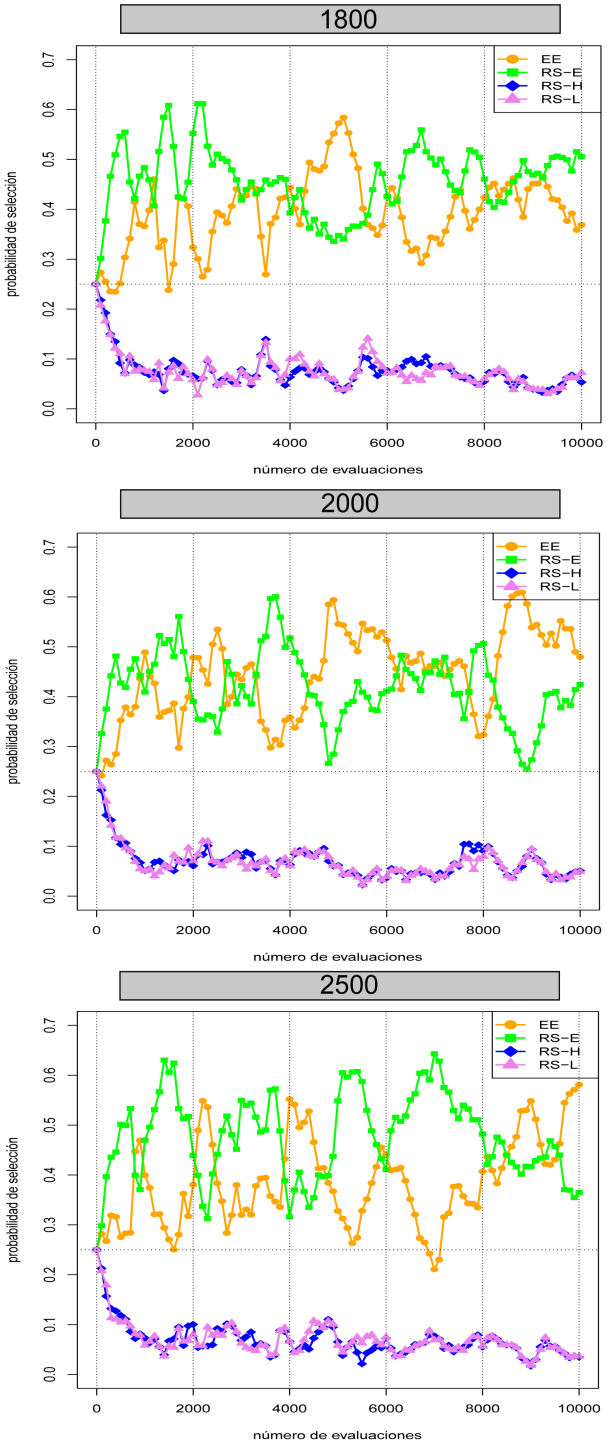

Figura 3.10: Evolución de la probabilidad promedio de selección de cada algoritmo individual para el problema DMCLP-AC, medida cada 500 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos individuales que tienen la misma probabilidad de selección.

Algoritmo	Global	Instancia-1800	Instancia-2000	Instancia-2500
PF	**1**	**1**	**1**	**1**
RS-E	3.9778	4.4667	3.9333	3.5333
RS-H	4.3333	4.2	4.2667	4.5333
RS-L	3.6889	3.3333	3.8	3.9333
EE	2	2	2	2
p-valor	0.00	0.00	0.00	0.00

Tabla 3.6: Resultado proporcionado por el test de Friedman, para cada algoritmo específico cuando se consideran todos los problemas (Global), y sobre todas las instalaciones y distancias. En todos los casos el algoritmo de mejor ranking se encuentra marcado en negrita.

3.7.3. Resultados DMCLP localizaciones fijas

En este apartado se realiza el análisis del rendimiento del portafolio de algoritmos (PF) con respecto a los algoritmos individuales que lo componen sobre la variante del problema DMCLP donde no cambian las instalaciones disponibles de un periodo a otro, o sea las localizaciones se mantienen fijas. Primeramente se utilizó el test de Friedman, donde se compararon los diferentes algoritmos para todas las instancias, y en cada instancia por separado. En todos los casos, el test devolvió un $p - valor = 0.0$, lo que indica que existen diferencias significativas entre los algoritmos. Los resultados del test de Friedman se muestran en la Tabla 3.6 y se encuentra señalado en negritas el algoritmo con mejor ranking promedio. En todos los casos el algoritmo con mejor ranking promedio fue el portafolio de algoritmos.

Luego, con el objetivo de evaluar si el mejor algoritmo en cada caso según el test de Friedman tiene un rendimiento significativamente diferente a los demás, se les aplicaron los test post-hoc: Holm y Finner, con un nivel de significación de $\alpha = 0.05$. Los resultados obtenidos se muestran en la Tabla 3.7. El símbolo '*' indica el algoritmo considerado como método de control (el método con el mejor ranking promedio según el test de Friedman: Portafolio); '>' significa que el método de control mejora significativamente al método correspondiente; y '-' indica que no hay diferencias significativas entre los dos métodos. En todos los casos los test de Holm y Finner coincidieron en los resultados.

Se puede apreciar que en este análisis ocurre algo similar a las dos variantes anteriores del problema DMCLP, donde de forma global existen diferencias significativas entre el portafolio (el método con el mejor ranking promedio según el test de Friedman) y el resto de los algoritmos, pero cuando descomponemos el análisis por problemas en este caso no hay diferencias entre el portafolio y la Estrategia Evolutiva. Esto se debe a las mismas razones que se expusieron en la Sección 3.7.1.

En la Figura 3.11 se muestran los gráficos de la evolución del porcentaje de demanda

Algoritmo	Global	Instancia-1800	Instancia-2000	Instancia-2500
PF	*	*	*	*
RS-E	>	>	>	>
RS-H	>	>	>	>
RS-L	>	>	>	>
EE	>	–	–	–

Tabla 3.7: Resultados obtenidos por los test post-hoc Holm y Finner, con un nivel de significación de $\alpha = 0.05$ para el algoritmo mejor de forma global (en todos los problemas) y en cada problema concreto, se compara con los demás. El símbolo '*' indica el algoritmo considerado como método de control (el método con el ranking de la mejor media); '>' significa que el método de control mejora significativamente al método correspondiente; y '-' indica que no hay diferencias significativas entre los dos métodos.

cubierta con respecto al total de demanda, para todos los escenarios: 1800 nodos de demanda y 100 nodos de instalaciones, 2000 nodos de demanda y 150 nodos de instalaciones, 2500 nodos de demanda y 200 nodos de instalaciones, desagregando en cada instancia por el umbral de cobertura S (4, 4.5, 5). Los valores concretos de porcentaje de demanda cubierta se encuentran en el Apéndice B Sección B.3, en las Tablas B.13, B.14 y B.15. En la mayoría de los casos el porcentaje de demanda tiende a aumentar en la medida en que aumenta el radio de cobertura. Además se puede apreciar claramente que el algoritmo con mejor rendimiento es el portafolio, lo cual corrobora el análisis realizado anteriormente.

Por último, se realizó un estudio de como evoluciona la probabilidad de selección $P_{selec}^{a_i}$ de los algoritmos que componen el portafolio(RS-E, RS-H, RS-L y EE), y se seleccionó una muestra de cada escenario con valores intermedios para cobertura e instalaciones a ubicar, 4.5 y 75, respectivamente. La Figura 3.12 ilustra la evolución de la probabilidad promedio de selección para cada algoritmo individual, medida cada 500 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. Cada gráfico se corresponde con cada una de las instancias, y la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos, que al inicio tienen la misma probabilidad de selección.

En cuanto a los gráficos, se puede apreciar que no se muestra claramente cual es el componente que más utiliza el portafolio. Estos resultados obtenidos son muy similares a los que se expusieron en la Sección 3.7.1. En este sentido, algoritmos como: RS-E y RS-H, que no se encuentran entre los de mejor rendimiento cuando se ejecutan de forma individual, pudieran ser útil en algunos momentos del proceso de búsqueda cuando se combinan con otros métodos.

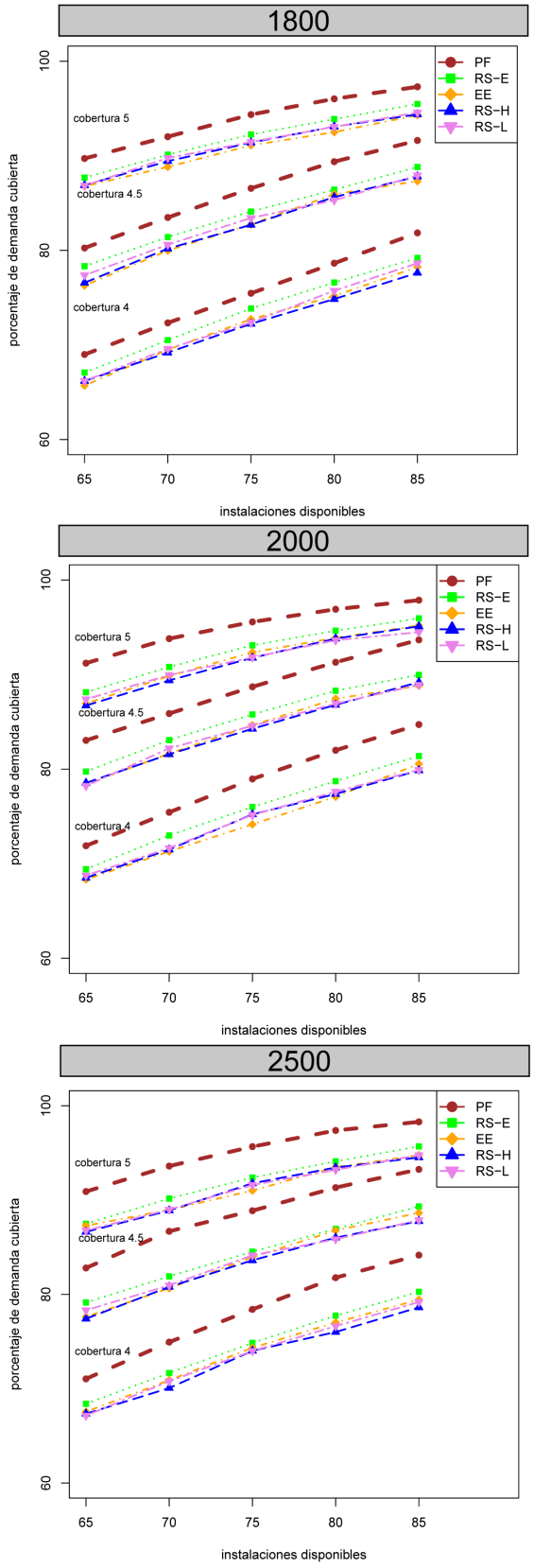

Figura 3.11: Evolución del porcentaje de demanda cubierta con respecto al total de demanda para todas las instancias, para el problema DMCLP-LocF. Se representa cada algoritmo y escenario: cobertura e instalaciones activas.

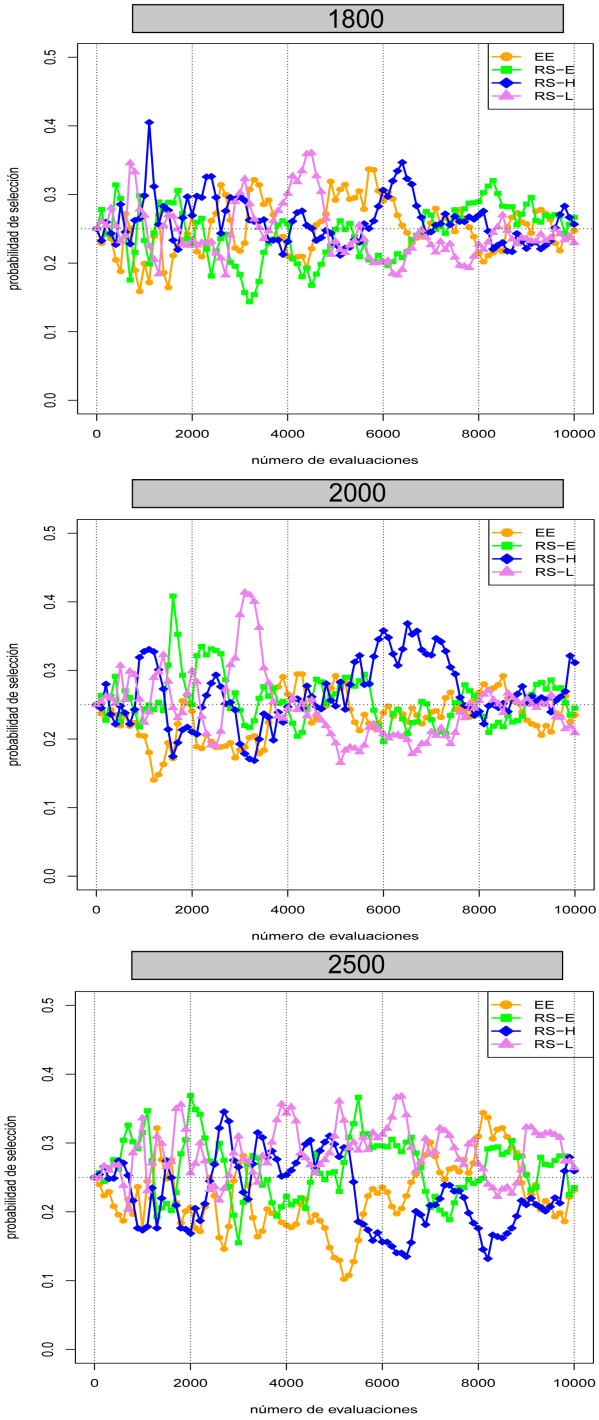

Figura 3.12: Evolución de la probabilidad promedio de selección de cada algoritmo individual para el problema DMCLP-LocF, medida cada 500 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. En cada gráfico, la línea horizontal marca el valor de probabilidad para una distribución uniforme en todos los algoritmos individuales que tienen la misma probabilidad de selección.

3.8. Sumario

Este capítulo estuvo dedicado al problema de máxima cobertura dinámico o multi-periodo (DMCLP), el cual tiene como objetivo maximizar la cobertura de un conjunto de puntos de demanda dentro de una distancia de servicio o tiempo, con la limitación de un número máximo de instalaciones para todos los periodos de tiempo que se analicen. Se propusieron dos variantes nuevas del problema: DMCLP-AC, que incorpora costes de apertura y cierre de las instalaciones o facilidades, y DMCLP-LocF, en el que las instalaciones abiertas permanecen fijas a lo largo de todos los periodos de tiempo.

Se realizaron experimentos con el portafolio de algoritmos diseñado para resolver problemas de optimización dinámicos propuesto en el Capítulo 2 y los algoritmos individuales: Recocido Simulado Exponencial, Recocido Simulado Hiperbólico y Recocido Simulado Lineal. Para tres instancias de DMCLP de tamaños diferentes (1800, 2000 y 2500 nodos de demanda, y 100, 150 y 200 instalaciones) y combinaciones de cobertura (4, 4.5 y 5) e instalaciones a localizar (65, 70, 75, 80 y 85), para cinco periodos de tiempo. Para comparar los métodos, utilizamos como medida de rendimiento el promedio de la evaluación en la Función Objetivo de la mejor solución encontrada, y test estadísticos no paramétricos para comprobar las diferencias significativas entre los algoritmos.

Después de analizar los resultados de la experimentación, podemos extraer las siguientes conclusiones:

- Al considerar los resultados sobre las tres variantes del problema DMCLP (clásico, costo por apertura o cierre y localizaciones fijas) el portafolio fue mejor que los métodos individuales en cada una de las instancias, pero el mejor método individual no fue siempre el mismo.
- Los resultados del portafolio de algoritmos para las tres variantes del DMCLP muestra que se crea una sinergia entre los algoritmos que lo componen, ya que de forma global mejora significativamente a todos los métodos individuales. De esta manera, con la combinación de diferentes metaheurísticas comunes bajo un esquema de aprendizaje simple se pueden obtener resultados muy competitivos con respecto a los algoritmos individuales que se encuentran reportados con buenos resultados en la literatura, tanto sobre la variante clásica del problema como con las dos variantes propuestas en este capítulo.
- En el análisis de la evolución de las probabilidades de selección se pudo observar que los algoritmos que tienen mal rendimiento cuando se ejecutan individualmente, pueden ser útiles en momentos específicos de la búsqueda cuando se combinan con otros métodos en el portafolio de algoritmos.

- Para las variantes: DMCLP y DMCLP-LocF, el análisis del comportamiento de las probabilidades de selección arrojó que no hay un método que se utilice claramente más que el resto.
- En la variante del problema que incluye costo por apertura o cierre (DMCLP-AC) el análisis de la evolución de las probabilidades de selección se pudo apreciar claramente que el portafolio tiende a utilizar la Estrategia Evolutiva y el Recocido Simulado Exponencial, teniendo mayores valores de probabilidades en las tres instancias del problema. Las probabilidades de los algoritmos Recocido Simulado Hiperbólico y Recocido Simulado Lineal convergen a valores bajos.

En términos generales, los resultados mostraron que el portafolio de algoritmos proporciona buen comportamiento frente al DMCLP.

Capítulo 4

BiCIAM: una librería software para resolver problemas de optimización

En la práctica existe una gran diversidad de problemas de optimización y una constante evolución en los modelos asociados a los problemas, ya que un problema puede cambiar o necesitar adaptaciones. Algunos de los objetivos y/o restricciones pueden añadirse, eliminarse o modificarse. En general, para resolver un problema es necesario hacer experimentos con varios algoritmos, ajustar los parámetros de cada uno, y evaluar el rendimiento. Las metaheurísticas han demostrado su utilidad en una amplia variedad de problemas reales, ya que ofrecen un equilibrio razonable entre la calidad de la solución y el tiempo de cálculo. Además, en el dominio de las metaheurísticas se está evolucionando a nuevos algoritmos, y se desarrollan metaheurísticas más complejas (por ejemplo: estrategias cooperativas [56, 55], hiperheurísticas [15], portafolios de algoritmos [62], modelos paralelos [53], entre otros).

Es importante para los usuarios que tienen pocos conocimientos de programación y sin previa experiencia en la optimización, tener una implementación disponible de algoritmos metaheurísticos para seleccionar y aplicar los métodos del estado de arte en la solución de su problema. En el caso de los expertos en optimización y desarrolladores, es útil para evaluar y comparar diferentes algoritmos frente a un problema, transformarlos, diseñar nuevos métodos, combinar y paralelizar algoritmos.

En este capítulo se presenta la herramienta software más importantes que hemos realizado a lo largo de toda la tesis. Concretamente, es una librería de software (framework) que permite la reutilización y aplicación de las metaheurísticas para resolver problemas de optimización, y que se ha denominado BiCIAM. Se basa en la separación conceptual de la parte invariante o genérica: metaheurísticas, y su parte específica: problema. Este capítulo tiene como objetivo constituir una referencia para comprender la estructura y comporta-

miento del framework propuesto. Para la modelación del framework y la descripción de sus componentes se utilizó el lenguaje de modelado UML [105], y algunos artefactos, tales como: vista de arquitectura, diagramas de clases, diagrama de actividad.

El capítulo se organiza en las siguientes secciones. En la Sección 4.1 se comentan las principales características de BiCIAM y los algoritmos que tienen disponibles. En la Sección 4.2 se propone el diseño a alto nivel o diseño arquitectónico de BiCIAM, el cual tiene como objetivo identificar y describir la estructura del framework a alto nivel de abstracción, piezas principales que lo conformarán, sus relaciones y formas de interacción, con especial énfasis en sus subcomponentes y las decisiones de diseño tomadas para cada uno de ellos. En la Sección 4.3 se muestra un diseño detallado con los elementos significativos identificados, se describe cada elemento interno con un nivel de detalles suficiente para comprender la implementación utilizando diagramas de clases de los paquetes principales. En la Sección 4.4 se describen los patrones y mecanismo de diseño que se tuvieron en cuenta en la implementación del framework, los cuales garantizan su flexibilidad y robustez. Luego en la Sección 4.5 se presenta un caso de estudio que ilustra el empleo del framework. Finalmente, en la Sección 4.6 se presenta un breve resumen del capítulo.

4.1. Principales características de BiCIAM

En la actualidad se ha propuesto numerosos software que implementan algoritmos metaheurísticos, cuyos códigos se encuentran disponibles en la Web y pueden ser reutilizados y adaptados a un problema. Sin embargo, en muchos de los casos el usuario tiene que examinar a fondo el código y volver a escribir sus secciones específicas para el problema. Esta tarea es a menudo tediosa, propensa a errores, lleva mucho tiempo, y hace más difícil el mantenimiento del código producido. Por ello la tendencia es a no reutilizar ningún código.

Para desarrollar un framework de metaheurísticas se utilizan tres criterios principales:

Desde cero: en la actualidad este método se utiliza bastante. Los programadores desarrollan sus propios códigos de los algoritmos. Por lo tanto, se enfrentan a varios problemas: el desarrollo requiere tiempo y energía, es propenso a errores, difícil de mantener y evolucionar.

Solo reutilizar código: consiste en el uso de código de terceros que se encuentren disponibles, ya sean libres o como bibliotecas en la Web. Un código de terceros tiene secciones generalmente dependientes de la aplicación de donde son extraídos, el cambio de estas secciones a menudo es lento y propenso a errores. La mejor forma de reutilizar el código de metaheurísticas es a través de las bibliotecas, porque suelen estar probadas y documentadas, por lo tanto son más confiables. Sin embargo, muchas bibliotecas permiten la reutilización de código, pero no permiten la reutilización de algunas partes invariantes de los algoritmos para modificar su funcionamiento. Algunas bibliotecas no permiten la reutilización del di-

seño. Por lo tanto, el esfuerzo de codificación cuando se utilizan bibliotecas continúa siendo importante.

Diseño y reutilización de código: el objetivo de combinar diseño con reutilización es superar los problemas que se presentan en los enfoques anteriores, y reescribir la menor cantidad de código posible cuando nos enfrentamos a nuevo problema de optimización. La idea básica es capturar en componentes especiales la parte recurrente (o invariante) de los métodos de solución a los problemas que pertenecen a un dominio específico. Estos componentes especiales se denominan patrones de diseño. La mayoría de los patrones de diseño en relación al dominio de las metaheurísticas, son implementados como frameworks. Un framework puede ser orientado a objetos y se define como un conjunto de clases que incorporan un diseño abstracto de soluciones con diferentes metaheurísticas relacionadas.

Teniendo en cuenta los aspectos anteriores se desarrolló BiCIAM, que es un framework no comercial escrito en Java, destinado a ser explotado por la comunidad científica. La modelación y construcción del framework, es también una representación formal, donde se intentan evitar los detalles relacionados con la tecnología, como el lenguaje de programación o el protocolo de comunicación entre los componentes. En el diseño de BiCIAM se tuvo en cuenta un conjunto de requisitos que permiten satisfacer a los usuarios e influir en el éxito de su explotación, que a continuación se describen:

Máxima reutilización de diseño y código: el diseño de la arquitectura que se utilizó fue basado en el enfoque de reutilización, lográndose una separación entre las metaheurísticas y el problema a resolver, simplificando el desarrollo y reduciendo el tiempo para modelar un nuevo problema e utilizar los algoritmos disponibles.

Flexibilidad y adaptabilidad: es posible añadir fácilmente nuevas características a las metaheurísticas o modificar las existentes sin implicaciones en la creación de nuevos componentes del framework. En la práctica los problemas evolucionan o surgen otros nuevos, y estos pueden a ser abordados mediante la especialización o adaptación de los componentes del framework.

Utilidad: permite al usuario cubrir una amplia gama de metaheurísticas, problemas, mecanismos de hibridación, diferentes técnicas multi-objetivo. Además se pueden modelar problemas de optimización dinámicos y la estrategia de búsqueda se adapta a este tipo de problemas y los cambios que pueden ocurrir en el tiempo.

Portabilidad: satisface un gran número de usuarios ya que se desarrolló en el lenguaje Java que es multiplataforma, por lo tanto es compatible con diferentes sistemas operativos (Windows, Unix, MacOs). Además tiene facilidad de integración con otras herramientas, ya que tiene un bajo grado de acoplamiento entre los componentes, así como el uso de formatos de intercambio de datos es estándar.

Algoritmos	Referencia	Mono-Objetivo	Multi-Objetivo
Búsqueda Aleatoria	[121]	✓	
Escalador de Colinas	[1, 104]	✓	✓
(mejor y primer ascenso, reinicio)			
Recocido Simulado	[16, 69]	✓	✓
(clásico, exponencial, lineal, hiperbólico)			
Búsqueda Tabú	[50, 51]	✓	✓
Limitado por Umbral	[1]	✓	
Estrategia Evolutiva	[96]	✓	
Algoritmo Genético	[60, 52, 43]	✓	✓
EDA-UMDA	[107]	✓	
NSGAII	[28]		✓
Enjambre de Partículas	[67]	✓	
Portafolio de Algoritmos	[37]	✓	✓

Tabla 4.1: Algoritmos metaheurísticos disponibles en BiCIAM.

Fácil de usar: es fácil de utilizar y no incorpora un costo adicional en términos de tiempo o complejidad, ya que su diseño está basado en mecanismos y patrones de diseño, los cuales están documentados. Además la configuración de los algoritmos y definición del problema es simple e intuitiva.

En BiCIAM se incluyen una gran recopilación de algoritmos metaheurísticos, la Tabla 4.1 resume la lista de estos algoritmos, identificando los algoritmos basados en trayectoria y los algoritmos basados en poblaciones, para resolver problemas mono-objetivos y multi-objetivos.

4.2. Descripción de la Arquitectura

La identificación de las mejores decisiones de diseño para cada componente del framework es una tarea difícil debido a los numerosos grados de libertad disponibles. Por otra parte, lo que es "mejor" para un problema, podría ser peor en uno diferente. Por ello, la arquitectura del framework se definió a partir de las piezas o componentes que lo conforman, conjuntamente con las restricciones para las interacciones entre estos. Además de las propias piezas o componentes que quedan identificadas en la arquitectura, se pueden visualizar en el código resultante (paquetes, clases, interfaces), que son las bases sobre las cuales se construye el sistema.

En la Figura 4.1 se muestra el diagrama de estructuración en capas que es una artefacto que contiene paquetes, clases e interfaces que componen la arquitectura estructurada en capas. El enfoque utilizado para la estructuración en capas está basado en reutilización

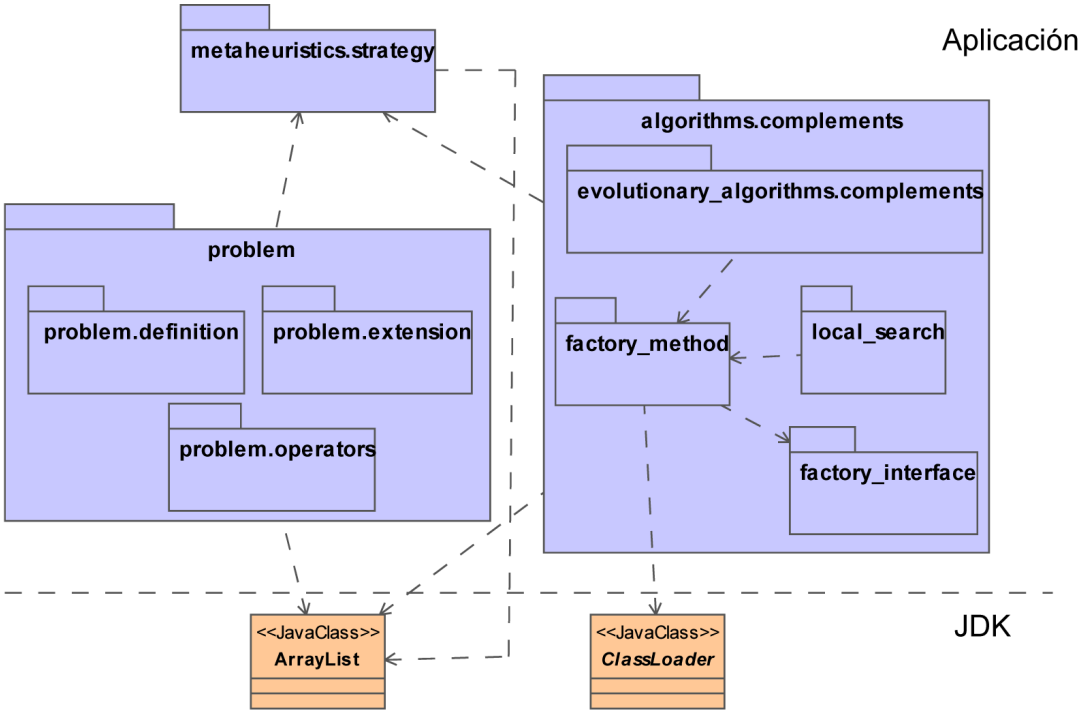

Figura 4.1: Vista de la arquitectura de BiCIAM, basada en el enfoque de reutilización. La capa general se corresponde con: Aplicación, y la capa intermedia se corresponde con: JDK.

propuesto por Peter Eeles en el año 2001 [35], el cual permite ver la aplicación desplegada desde sus componentes más específicos hasta los más reutilizables [111].

La vista de la arquitectura se encuentra proyectada en dos capas: general e intermedia. La capa general a la cual se le denominó Aplicación contiene todos los paquetes referentes al framework, proporcionando una estructura de control para los algoritmos y el problema, basada en una clara separación conceptual entre los métodos de solución y los problemas que abordan. La capa intermedia a la cual se le denominó JDK, la cual refleja las clases e interfaces que brinda la plataforma de Java. Es empleada para mostrar los elementos o paquetes que no son propios del sistema que se está desarrollando, sino de terceros, es brindado por el entorno de desarrollo (Java).

Los paquetes contenidos en la capa Aplicación se describen a continuación:

- paquete *metaheuristics.strategy*: contiene la clase controladora de BiCIAM: Strategy, encargada de implementar la estrategia para llevar a cabo el proceso de búsqueda, y abstraer el problema de los algoritmos. Además contiene las clases con los diferentes algoritmos que se encuentran implementados en BiCIAM.
- paquete *algorithms.complements*: contiene los paquetes *evolutionary.algorithms.complements*, *factory.method*, *factory.interface* y *local.search*, los

cuales agrupan las clases que son reutilizadas por varias clases del paquete *metaheuricticsstrategy* y no son propiamente parte de la lógica del modelo, sino que garantizan la implementación de los algoritmos.

- paquete *problem*: se considera como la parte variable del framework dependiendo de cada problema en específico, es fijo en el framework, pero implementado por el usuario. Contiene en un conjunto de huecos o puntos calientes que sirven para llenar los esqueletos que proporciona el framework al crear problemas específicos. En los paquetes: *problem.definition*, *problem.extension* y *problem.operators* se encuentran las clases que abstraen el problema del algoritmo, permitiendo que estas sean empleadas para definir cualquier problema.

4.3. Elementos significativos del diseño

Entre los elementos significativos del diseño se encuentran los paquetes y los diagramas de clases asociados a los paquetes. En el presente apartado se tiene como objetivo representar los diagramas de clases de cada paquete definido en la descripción de la arquitectura propuesta. En los diagramas de clases de diseño se van a describir gráficamente las especificaciones de las clases y relaciones más significativas del framework.

En la Figura 4.2 se ilustra el diagrama de clases del paquete *problem*, el cual tiene como principal objetivo contener las clases que necesita redefinir el usuario para modelar su problema. En este diagrama se han excluido algunos métodos y atributos de las clases para una mejor comprensión. Se puede apreciar que a la clase *Problem* se le asigna: función objetivo (*ObjectiveFunction*), operador o modificador (*Operator*), codificación o representación (*Codification*), estado (*State*). Además se define el método de solución (*SolutionMethod*) para el caso de los problemas multi-objetivos donde se debe tener en cuenta si el tratamiento es por: factores ponderados, ordenamiento lexicográfico o multi-objetivo puro.

La arquitectura de BiCIAM también permite incorporar las restricciones a la evaluación de una solución y penalizar la evaluación de acuerdo con el grado violación de dichas restricciones. Note en este sentido que las clases: *ObjectiveFunction*, *Operator* y *Codification*, son abstractas y para implementar un nuevo problema el usuario debe redefinir los métodos contenidos en las mismas, los cuales tienen como objetivo:

ObjectiveFunction

- *evaluation*: método que devuelve la/las evaluación/nes de la/las función/funciones objetivo/s de una solución del problema.

Operator

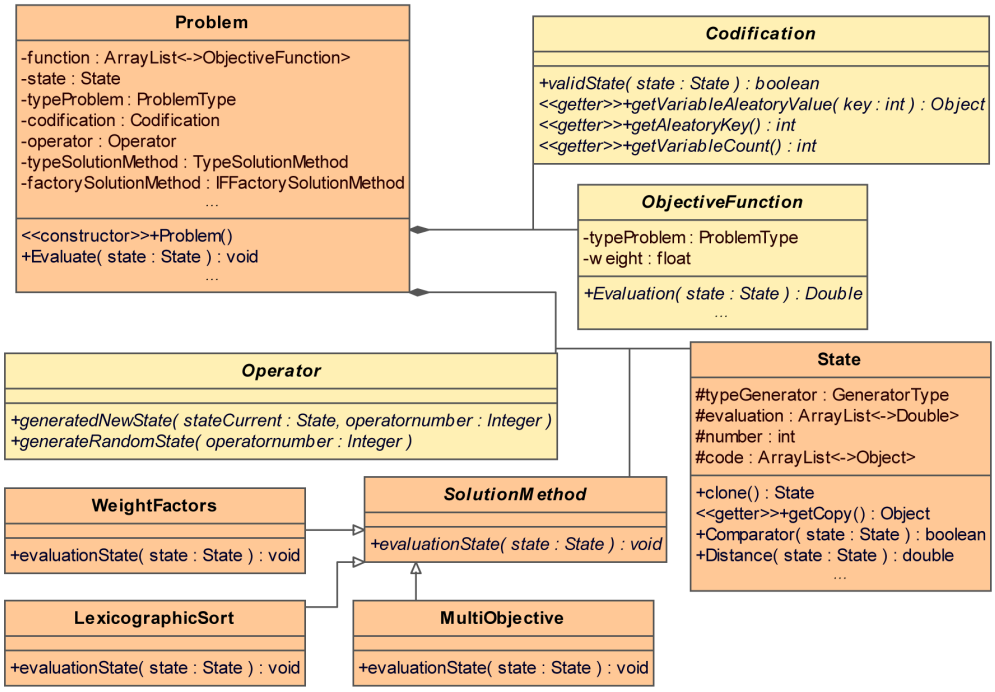

Figura 4.2: Diagrama de clases del paquete: *problem*

- *generatedNewState*: método que devuelve la vecindad de una solución. Se le pasa como parámetro la cantidad de soluciones de la vecindad que se desea obtener.
- *generateRandomState*: devuelve soluciones del problema generadas aleatoriamente. Se le pasa como parámetro el número de soluciones aleatorias que se desean generar.

Codification

- *getAleatoryKey*: devuelve una variable aleatoria de la codificación del problema.
- *getVariableAleatoryValue*: dada una variable del problema devuelve un valor aleatorio.
- *validState*: devuelve verdadero si una solución (parámetro de entrada) es válida, o sea cumple con las restricciones de las variables del problema, y falso en caso contrario.

En las Figura 4.3 y 4.4 se muestran los diagramas de clases del paquete *metaheuristics.strategy*, tiene como objetivo contener las metaheurísticas que se encuentran implementadas en BiCIAM. Como estos diagramas tiene mucho contenido se han excluido algunos métodos y atributos de las clases para una mejor comprensión. La clase *Strategy* es la encargada de llevar a cabo el proceso de búsqueda, teniendo la responsabilidad de manipular los datos del problema (*Problem*) y los parámetros de entrada. Las clases con tonalidad amarillo claro representan los algoritmos basados en trayectoria para problemas mono-objetivos

y multi-objetivos, y las clases con tonalidad amarillo más oscuro representan los algoritmos basados en poblaciones.

La clase *Metaheuristics* es abstracta y delega sus responsabilidades en las clases concretas que contienen las metaheurísticas que heredan de ella, teniendo que reimplementar los métodos:

- *generate*: en los algoritmos basados en trayectoria tiene como objetivo obtener una solución con la aplicación del operador de vecindad a la solución actual del algoritmo. Para el caso de los algoritmos basado en poblaciones obtener un nuevo individuo resultado de la aplicación secuencial de sus operadores (por ejemplo: selección → cruzamiento → mutación, en Algoritmos Genéticos)

- *updateReference*: tiene como objetivo la evaluación del criterio de aceptación del movimiento (por ejemplo, tabú y criterios de aspiración en la Búsqueda Tabú, probabilidad de aceptación del Recocido Simulado, reemplazo generacional de los algoritmos basados en poblaciones, etc).

En el diagrama de clases de la Figura 4.3 se encuentra la clase *Portfolio*, la cual tiene como objetivo implementar la lógica del portafolio de algoritmos propuesto en el Capítulo 2. Contiene un conjunto de métodos que garantizan la implementación de los diferentes esquemas de aprendizaje, inicialización de los algoritmos del portafolio, selección de un algoritmo aplicando el método de la ruleta, entre otros. De manera que, la clase *Strategy* es la encargada de ejecutar el proceso de búsqueda del portafolio de algoritmos, logrando integrar los componentes principales.

En la Figura 4.5 se muestra el diagrama de clases del paquete *algorithms.complements*, este módulo agrupa las clases que son reutilizadas por el paquete *metaheurictics.strategy* para garantizar la implementación de los algoritmos. En este diagrama se han excluido algunos métodos y atributos de las clases para mejorar la comprensión porque tienen mucho contenido. Se puede apreciar que las clases: *ParentSelection*, *Crossover*, *Mutation*, *Replace* y *Distribution*, son las clases que garantizan la implementación de los operadores que utilizan los algoritmos evolutivos. Además se encuentran las clases: *UpdateParameter* y *StopExecute*, las cuales tienen la responsabilidad de garantizar actualizar los parámetros de los algoritmos y detener el proceso de búsqueda, respectivamente.

Por otra parte, la estrategia de búsqueda de BiCIAM es independientemente del algoritmo que se desee ejecutar, estático o dinámico. Para el caso de los problemas dinámicos en la implementación del problema el usuario debe definir un subcomponente de dinamismo que requiera el problema en cuestión. El subcomponente de dinamismo depende de alguna condición que debe cumplirse para que un cambio tenga lugar. Por ejemplo, cuando la

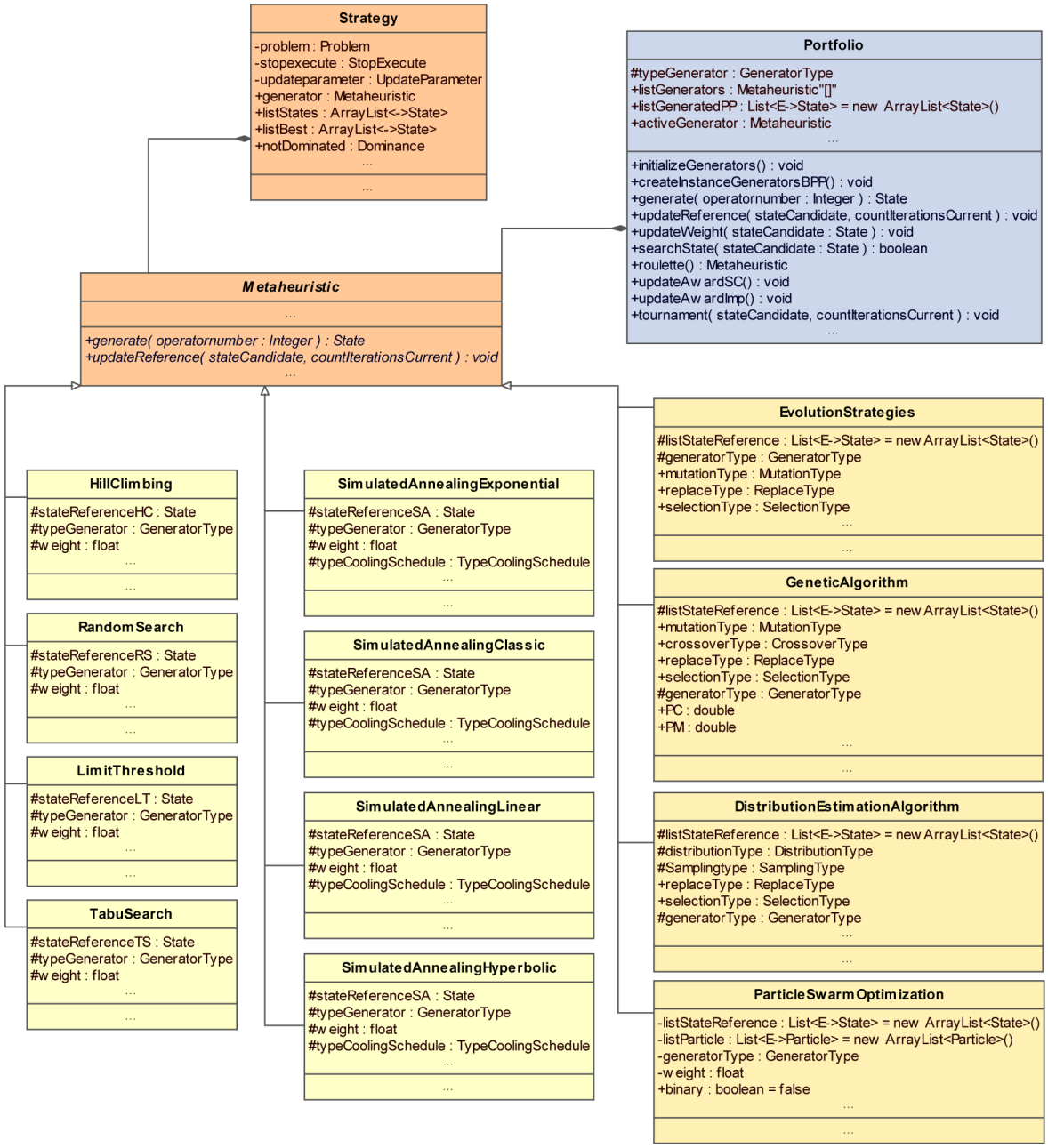

Figura 4.3: Diagrama de clases del paquete: *metaheuristics.strategy*, donde se representan los algoritmos para resolver problemas mono-objetivos.

condición para que un cambio ocurra es un cierto número de evaluaciones de la función objetivo, es obvio que esta condición depende directamente del proceso de búsqueda que se esté ejecutando. Por ello, el framework provee en cualquier momento de la búsqueda el número de evaluaciones actuales. Se sugiere a los usuarios que antes de evaluar una solución se compruebe si se da la condición para que ocurra el cambio, y luego continuar con el proceso de evaluación. En este sentido, se debe implementar en método *evaluation* de la

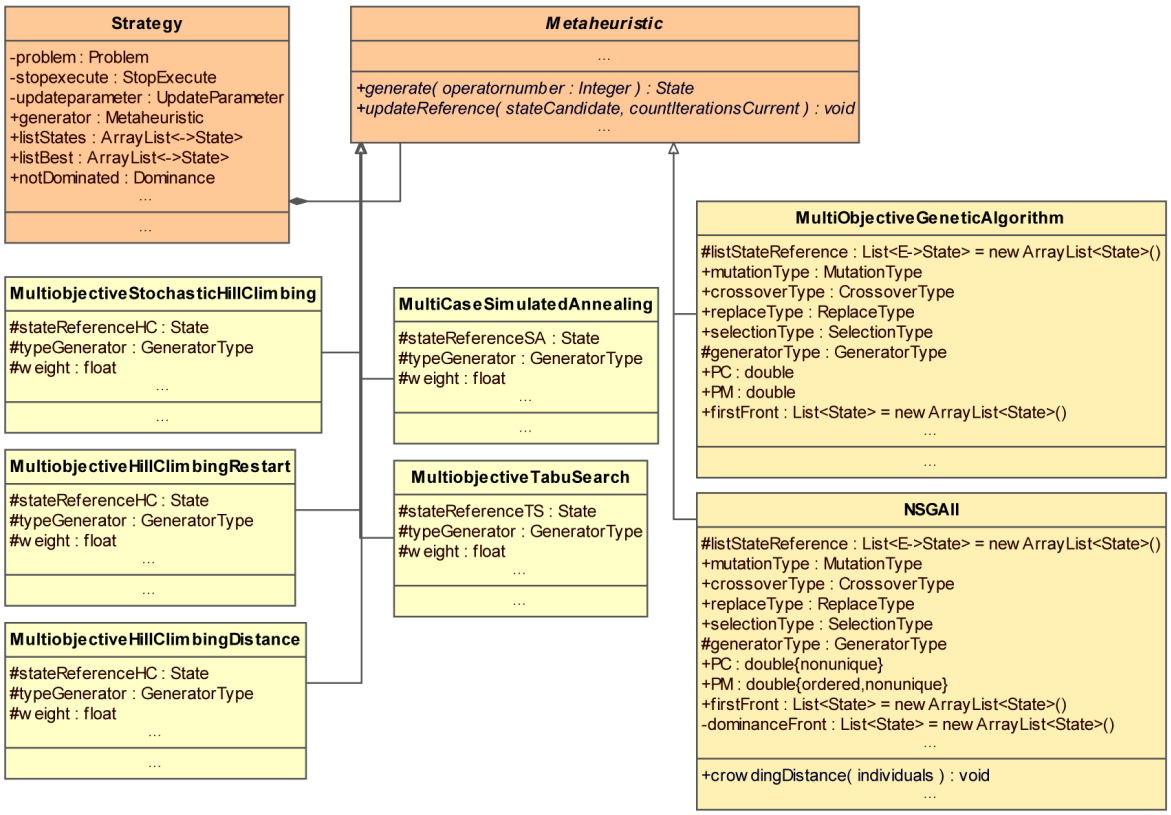

Figura 4.4: Diagrama de clases del paquete: *metaheuristics.strategy*, donde se representan los algoritmos para resolver problemas multi-objetivos.

clase *ObjectiveFunction* que se haya creado.

La Figura 4.6 se ilustra el flujo de ejecución de un experimento, donde la estrategia realiza el proceso de búsqueda de forma continua dentro de un ciclo hasta que se cumpla la condición de parada. En cada iteración se evalúa en la función objetivo una solución. Dentro de esta llamada el problema verifica si se cumplen condiciones para algún cambio. En caso positivo se realizan los cambios correspondientes y se evalúa, en caso negativo solo se evalúa la solución.

Además de los requisitos anteriores, BiCIAM tiene la capacidad para trabajar con problemas continuos y discretos. El framework ha sido diseñado desde el principio para manejar tipos de datos abstractos, tales como: *Operator*, *ObjetiveFunction*, *Codification*. Al encapsular los tipos de datos principales del framework, independientemente de las implementaciones particulares de dichos datos, se pueden gestionar problemas continuos con los componentes de las soluciones de tipos: *doble*, *float*; problemas discretos donde los componentes de las soluciones son de tipo: *integer*, *long*; problemas mixtos donde las soluciones tienen ambos componentes continuos y discretos. Los experimentos de esta tesis se han realizado sobre problemas discretos.

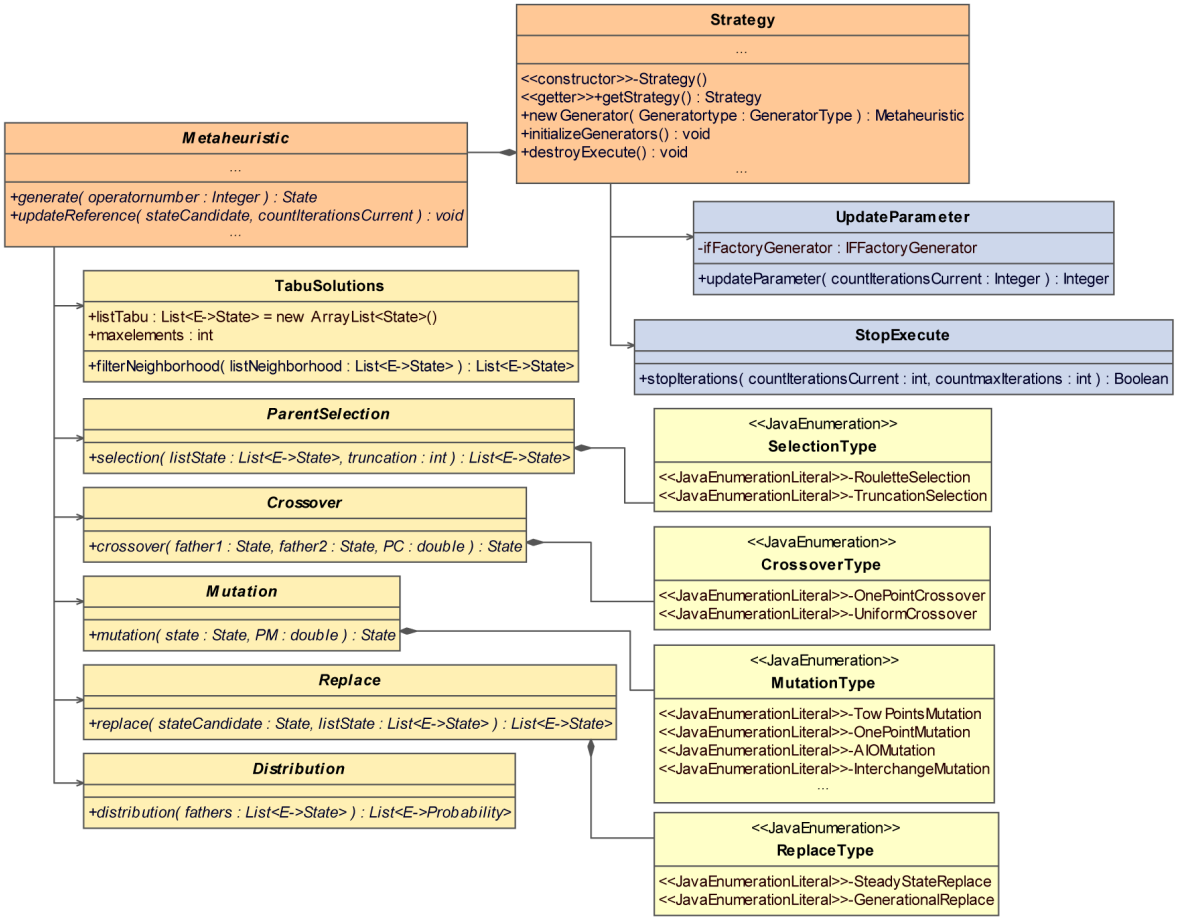

Figura 4.5: Diagrama de clases del paquete: *algorithms.complements*

Por otra parte, la solución inicial del proceso de búsqueda puede ser generada aleatoriamente o algunos problemas parten de construir una buena solución inicial, como resultado de aplicar alguna heurística de construcción. La implementación de las heurísticas de construcción son específicas en cada problema, por ello se decidió delegar esta responsabilidad a la clase *Operator*, independientemente de si el problema es estático o dinámico, mono-objetivo o multi-objetivo. En la Figura 4.7 se ilustra el flujo de ejecución de un experimento, donde la solución inicial se puede generar aleatoriamente o ser construida por alguna heurística en específico del problema en cuestión, y el proceso de búsqueda se realiza de forma continua dentro de un ciclo hasta que no se cumpla la condición de parada.

BiCIAM se ha utilizado para resolver un gran número de problemas de referencia de la literatura. Contiene un repositorio de problemas combinatorios estáticos clásicos, tales como: trazado de grafos, asignación de recursos humanos a equipos de proyectos de software, viajante de comercio. Además incluye por defecto un conjunto de problemas ampliamente utilizados en la optimización dinámica. Entre ellos podemos destacar, problemas de optimización discretos combinatorios: OneMax, Plateau, RoyalRoad, Deceptive, Mochila y viajante

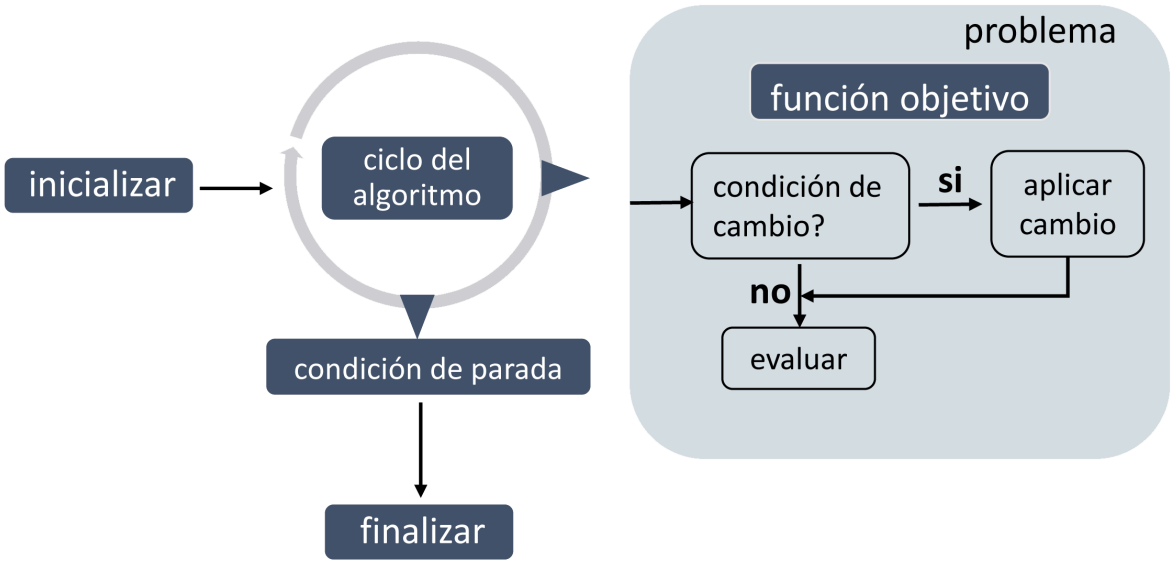

Figura 4.6: Flujo de ejecución de un experimento, donde la estrategia realiza el proceso de búsqueda de forma continua dentro de un ciclo hasta que no se cumpla la condición de parada, la solución que se genera en cada iteración se debe evaluar en la función objetivo, se verifica si se cumplen condiciones para algún cambio, y en tal caso se realizan los cambios correspondientes, antes de realizar la evaluación en el problema.

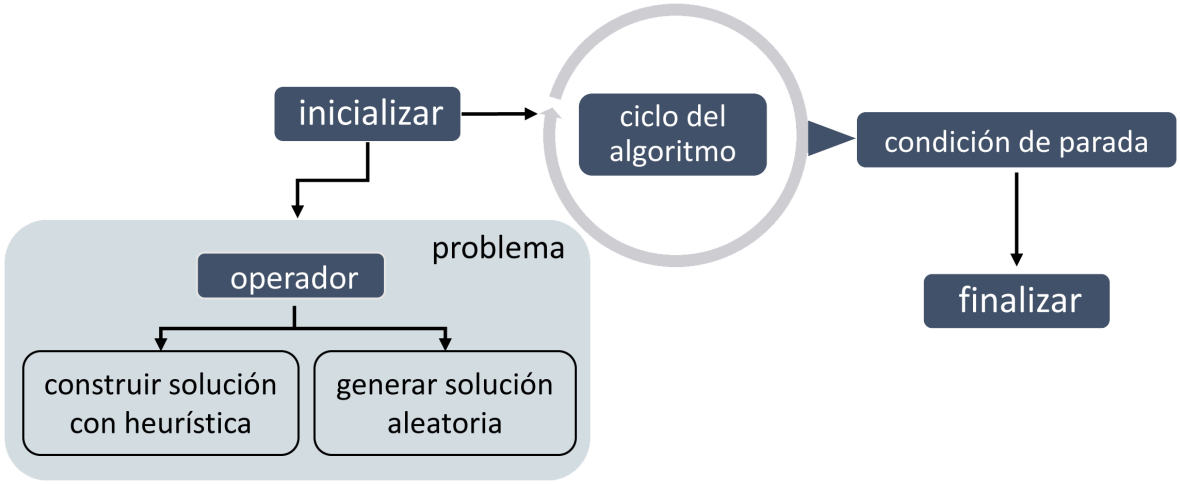

Figura 4.7: Flujo de ejecución de un experimento, donde la estrategia realiza el proceso de búsqueda de forma continua dentro de un ciclo hasta que no se cumpla la condición de parada. La solución inicial se puede generar aleatoriamente o ser construida por alguna heurística en específico del problema en cuestión.

de comercio dinámico. De las funciones continuas, podemos mencionar: Ackley, Griewank, Rastrigin, Sphere. En este sentido, se pueden mencionar resultados de investigación en los

que se ha utilizado:

- T. Ceruto Cordovés, Método para el descubrimiento de predicados difusos en forma normal en bases de datos utilizando metaheurísticas. Tesis de Doctorado, Instituto Superior Politécnico José Antonio Echeverría, CUJAE: La Habana, 2014.
- H. Díaz Pando, Modelo y estrategias de partición de componentes Hardware/Software en el co-diseño de sistemas embebidos. Tesis de Doctorado, Departamento de Tecnología Informática y Computación (DTIC), Escuela Politécnica Superior, Universidad de Alicante, 2014.
- I. Wilford Rivera, Modelo de integración de conocimiento huérfano descubierto mediante minería de datos. Tesis de Doctorado, Departamento de Tecnología Informática y Computación, Universidad de Alicante, 2010.
- A. Infante Abreu, Algoritmos de trayectoria multi-objetivo aplicados al problema de asignación de recursos humanos a equipos de proyecto de software. Tesis de Maestría, Instituto Superior Politécnico José Antonio Echeverría, CUJAE: La Habana, 2012.

4.4. Patrones y mecanismo de diseño en BiCIAM

En el diseño e implementación del framework BiCIAM se tuvieron en cuenta algunos patrones de diseño, que contribuyen a garantizar su robustez y flexibilidad. Los patrones de diseño se conocen como modelos formales que son aplicables a diferentes dominios; es decir, ayudan a seguir pautas comunes en la solución de problemas diferentes, pero semejantes en su estructura. Garantizan la flexibilidad y reusabilidad de los diseños, evitan la reiteración en la búsqueda de soluciones a problemas ya conocidos y proporcionan una estandarización del modo en que se realiza el diseño [114, 46].

Entre los patrones utilizados se encuentran los de creación, los cuales ayudan a que un sistema sea independiente de cómo se crean, se componen y representan sus objetos. Un patrón de creación de clases usa la herencia para cambiar la clase de instancia a crear, mientras que un patrón de creación de objetos delega la creación de la instancia en otro objeto. Se utilizan principalmente cuando la creación requiere toma de decisiones, y la toma de decisiones pudiese cambiar en el tiempo; ayudan a estructurar y encapsular estas decisiones. Los patrones de creación utilizados fueron: *Singleton* y *Factory Method* [46].

El patrón *Singleton* garantiza que solamente se cree una única instancia de una clase, proporcionando un punto de acceso global a ella y todos los objetos que utilizan una instancia de esa clase usan la misma instancia, o sea, controla el acceso a esta instancia. Provee un mecanismo para limitar el número de instancias de una clase, por lo que el mismo objeto

es siempre compartido por distintas partes del código. *Strategy* es la clase controladora del framework, por lo que se decidió crear un único punto de acceso global hacia ella, implementando el patrón *Singleton*. Para ello se puso su constructor con visibilidad privada, y para ser instanciada se debe invocar a su método: *getStrategy*, el cual es estático y retorna un atributo estático y privado que es del mismo tipo que la clase.

El patrón *Factory Method* es empleado para construir instancias a objetos individuales con un propósito específico, sin que la petición de construcción conozca las clases específicas que serán instanciadas en tiempo de ejecución. Este patrón permite a las superclases abstractas delegar responsabilidades de instanciación a las subclases, así como crear objetos dentro de una clase con un método de fabricación, que es más flexible que hacerlo directamente. En BiCIAM se implementa el *Factory Method* puesto que existen clases que no pueden anticipar que clases de objetos se deben crear. Por ejemplo, la clase *Metaheuristics* es abstracta y delega sus responsabilidades en las clases concretas que contienen los algoritmos. Dependiendo del algoritmo que desee el usuario ejecutar es la instancia de la clase concreta que deberá crearse en tiempo de ejecución. Se realiza indirectamente a través de una instancia de las clases *Factory* que implementan las interfaces *IFFactory*, la cual tiene declarado un método *create* que tiene como argumento necesario un *TypeMetaheuristics*, que es un enumerado, para determinar que clase concreta instanciar.

Para garantizar la carga dinámica de clases que utiliza el patrón *Factory Method* fue necesario implementar una clase, que pudiera ser un mecanismo de diseño, ya que se puede utilizar con el mismo objetivo en cualquier otra aplicación. Se creó la clase *FactoryLoader*, la cual contiene el método estático *getInstance*, al que se le pasa por parámetro el nombre de la clase que se desea instanciar. El comportamiento de este método se basa en verificar que exista una clase con el mismo nombre que el parámetro que se le pasa y devolver una instancia de la misma. Esta forma dinámica de cargar clases en tiempo de ejecución se realiza utilizando el API (o *Application Program Interface* por su nombre en inglés) *Java Reflection*, que permite a los programas examinar y hacer cambios a su estructura y comportamiento en tiempo de ejecución. *Java Reflection* posibilita obtener un objeto de una clase, examinar los métodos de una clase, invocar métodos descubiertos en tiempo de ejecución y explorar la jerarquía de la herencia.

4.5. Caso de estudio del uso de BiCIAM

En esta sección se presenta un caso de estudio que ilustra el funcionamiento de BiCIAM para resolver cualquier problema de optimización. El objetivo de este apartado es mostrar una forma simple de implementar un problema utilizando BiCIAM y la flexibilidad en la mayor parte del diseño para la modificación y ejecución de cualquiera de sus algoritmos.

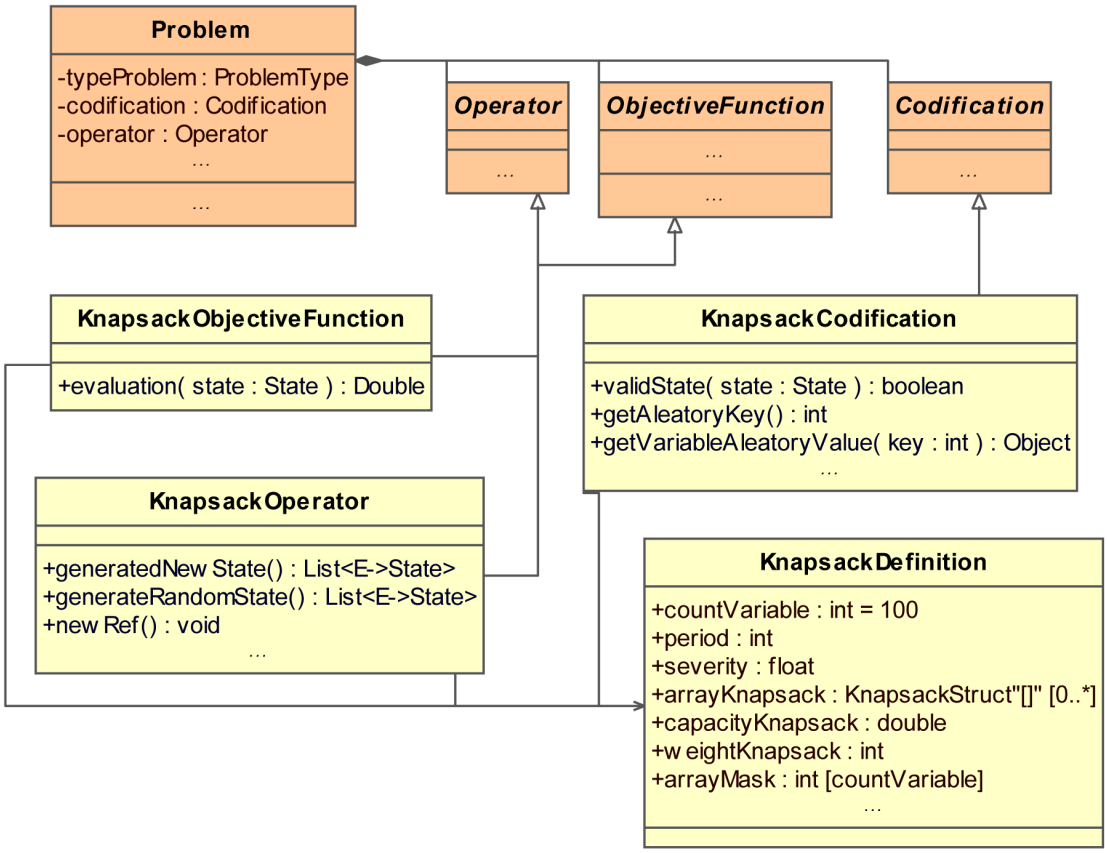

Figura 4.8: Diagrama de clases que modela para el caso de estudio sobre el problema de la Mochila

En el caso de estudio se utilizó el problema de Mochila en la variante dinámica comentada en la Sección 2.3.1 del Capítulo 2, y se definieron escenarios de experimentación para ilustrar el correcto funcionamiento del framework.

En la Figura 4.8 se muestra el diagrama de clases referente al diseño e implementación del problema de la Mochila, teniendo en cuenta la arquitectura que propone el framework. Las clases con tonalidad amarillo claro son las definidas por el usuario, que determinan los componentes que proporcionan las funcionalidades específicas del problema: función objetivo, representación o codificación, operador para generar la vecindad y definición de los datos específicos. Estos componentes son comunes en los problemas de optimización, por lo cual se deben redefinir para implementar un nuevo problema.

Después de definir las clases que necesita el problema, se debe crear el escenario de experimentación. En la Figura 4.9 se ilustra un diagrama de actividad con la secuencia de eventos que deben ocurrir para definir y ejecutar un experimento utilizando BiCIAM. Inicialmente se deben asignar los parámetros de los algoritmos, crear el problema con los componentes definidos anteriormente, cargar los datos del problema (costo y beneficio de

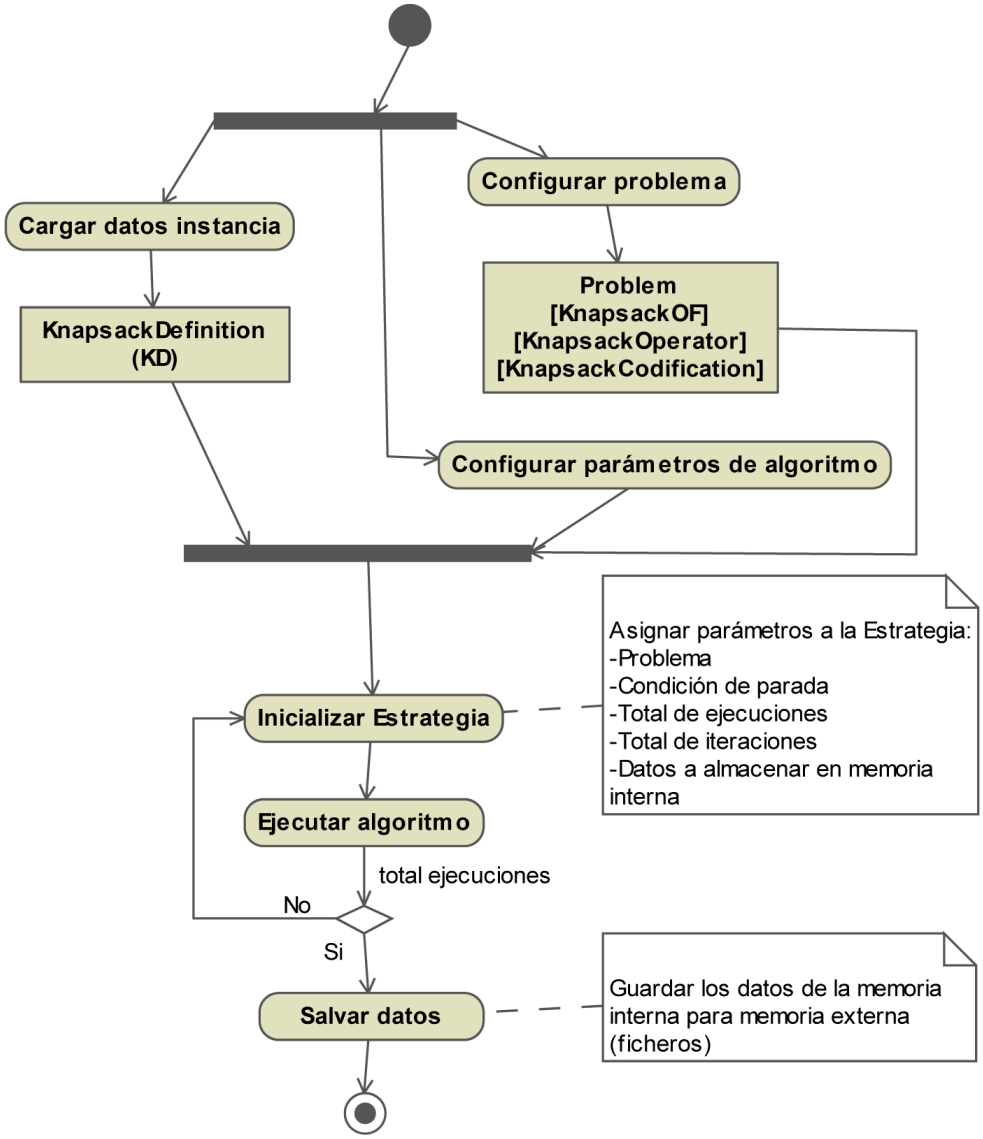

Figura 4.9: Diagrama de actividad con los eventos que deben ocurrir para definir y ejecutar un experimento utilizando BiCIAM

cada elemento que se desea analizar en la Mochila). Luego se crea la estrategia del framework, inicializando y asignando valor a: condición de parada del algoritmo, instancia del problema, datos que el usuario quisiera almacenar en memoria interna (mejor solución encontrada y su evaluación en la función objetivo, lista de las soluciones vecinas generadas en cada momento de la búsqueda, tiempo de ejecución del algoritmo), total de iteraciones y total de ejecuciones. Se ejecutará el algoritmo la cantidad de veces que se defina en el total de ejecuciones, un número máximo de iteraciones. Posteriormente se almacenará en memoria externa los datos que el usuario necesite para las comparaciones de los resultados.

Uno de los aspectos más importantes es la implementación de los principales com-

ponentes del problema: operadores para obtener la vecindad de una solución, representación/codificación y función objetivo. El tipo de operadores para obtener la vecindad de una solución tiene un impacto importante en el desempeño de las metaheurísticas. Además la interdependencia entre la codificación y los operadores que se utilicen debe estar clara, ya que dependiendo del tipo de datos y las variables del problema en cuestión, deberán definirse los operadores a aplicar. También no se debe descuidar la implementación de la función objetivo, porque dependiendo de este componente puede aumentar o no el costo computacional de la ejecución de un algoritmo.

Por el contexto de esta memoria, que contempla los ambientes dinámicos, se decidió utilizar como caso de estudio una variante dinámica del problema de la Mochila. En la Figura 4.8 se puede observar que la clase *KnapsackDefinition* contiene varios atributos que permiten modelar el problema como dinámico: severidad (cuanto cambia el problema) y frecuencia de cambio (cada cuanto tiempo cambia el problema). Los cambios se producen cada cierto número de evaluaciones en la función objetivo. En la evaluación de la función objetivo del problema (método *evaluation* de la clase *KnapsackObjectiveFunction*) se verifica si se cumplen condiciones para algún cambio. En caso positivo se realizan los cambios correspondientes y se evalúa la solución, mientras que en caso negativo solo se evalúa la solución. Cuando se produce un cambio se genera una nueva máscara (*arrayMask*) que se utiliza en cada evaluación de la función objetivo en el operador XOR.

Por otra parte, según los intereses del usuario debe proveer una configuración para guardar los datos contenidos en memoria interna el framework como resultado de la ejecución de los algoritmos, y luego llevarlos a memoria externa o ficheros. Los datos que se pueden guardar son: mejor solución encontrada y su valor objetivo, lista de las mejores soluciones encontrados en cada iteración, lista de las soluciones vecinas generadas en cada momento de la búsqueda y sus respectivos valores objetivo, tiempo de ejecución del algoritmo, y frente de Pareto o soluciones no dominadas. Además se pueden almacenar en memoria algunas métricas clásicas que permiten evaluar el comportamiento de los algoritmos, entre ellas: promedio, mediana, máximo, mínimo y desviación estándar. Para el caso de los problemas de optimización dinámicos se puede salvar el valor de métricas que están definidas dentro de BiCIAM: offline error y offline performance [12, 14], comentadas anteriormente en el Capítulo 1 Sección 1.3.3.

La estructura de los ficheros de los resultados se puede definir en dos formatos diferentes, dependiendo del tipo de análisis que se desee realizar. El primer formato consiste en almacenar por cada ejecución el valor promedio de las métricas definidas, o sea habrán tantas filas como ejecuciones. En la Tabla 4.2 se muestra un ejemplo del formato de los resultados obtenidos sobre la métrica offline performance, para un escenario del problema de la Mochila con severidad 0.5 y frecuencia de cambio 6000, para la primera ejecución de los algoritmos con 100 cambios de la función objetivo. Analizando los resultados obtenidos

Ejecución	*Algoritmo*	*Promedio*	*Mediana*	*Máximo*	*Mínimo*	*DesvEstándar*
1	Escalador Colinas	1668.63	1667.34	1692.97	1640.75	10.43
1	Búsqueda Aleatoria	1667.66	1667.50	1680.11	1660.22	4.21
1	Recocido Simulado	**1676.53**	**1677.15**	**1698.66**	1653.65	7.55
1	Búsqueda Tabú	1661.84	1662.14	1681.9	1643.54	7.03
1	Estrategia Evolutiva	1652.15	1652.53	1682.92	1600.68	14.91
1	Algoritmo Genético	1651.52	1653.28	1678.77	1615.96	13.38
1	EDA (UMDA)	1649.24	1651.54	1682.51	**1590.95**	**16.66**

Tabla 4.2: Resultados obtenidos por algoritmos de BiCIAM, utilizando el primer formato donde para la primera ejecución del problema de la Mochila con el escenario: severidad 0.5 y frecuencia de cambio 6000, se muestran los valores de las métricas definidas por el usuario. En todos los casos el mejor valor se encuentra marcado en negrita.

se puede observar que el algoritmo Recocido Simulado obtiene los mejores resultados en la mayoría de los casos.

Para el caso del segundo formato se salvará el valor de las métricas definidas en cada iteración por cada ejecución. En este último los ficheros pueden crecer en capacidad de almacenamiento dependiendo del total de iteraciones por ejecuciones, por lo que se debe tener mucho cuidado con algún error de memoria en el momento de crear el fichero. En caso de que la muestra de la experimentación sea grande se recomienda crear un fichero con este formato por cada ejecución.

En BiCIAM también se dispone del portafolio de algoritmos propuesto en el Capítulo 2, el cual utiliza los mismos formatos para almacenar los resultados que el resto de los algoritmos. Además se puede realizar un estudio del funcionamiento interno del portafolio, o sea, analizar como evolucionan los créditos o probabilidades de selección de los algoritmos a lo largo de una ejecución. El portafolio dispone de una estructura de datos interna que se puede almacenar en un fichero con un formato diseñado específicamente para este tipo de análisis. Se debe definir cada cuanto tiempo o evaluaciones en la función objetivo se desea medir los valores de los créditos. Las columnas y las filas del fichero se corresponden con los diferentes algoritmos que integran el portafolio y los valores de las probabilidades de selección, respectivamente.

En la Tabla 4.3 se muestra un ejemplo del formato de los resultados obtenidos para la evolución de la probabilidad de selección de los algoritmos en el portafolio. Sobre una muestra de experimentos con el problema de la Mochila y el escenario: severidad 0.5 y frecuencia de cambio 6000, para 100 cambios de la función objetivo. La variante del portafolio que se seleccionó fue RS-AP-RB (reinicio de los créditos cuando ocurre un cambio, penalización activa, premio en el crédito si se genera soluciones estrictamente mejores que la solución disponible) comentada anteriormente en el Capítulo 2 Sección 2.1.2. RS-AP-RB dispone de los

Medida	*EC*	*BA*	*BT*	*RS*	*AG*	*EE*	*EDA*
0	0.142	0.142	0.142	0.142	0.142	0.142	0.142
1	0.155	0.128	0.137	0.160	0.136	0.135	0.145
2	0.133	0.138	0.163	0.168	0.136	0.128	0.131
3	0.149	0.132	0.160	0.167	0.128	0.122	0.139
4	0.133	0.112	0.214	0.202	0.112	0.113	0.112
5	0.140	0.114	0.223	0.138	0.110	0.131	0.142
6	0.132	0.124	0.243	0.124	0.106	0.144	0.123
7	0.124	0.111	0.277	0.131	0.110	0.121	0.122
8	0.161	0.126	0.197	0.126	0.120	0.132	0.134
9	0.128	0.139	0.187	0.173	0.117	0.127	0.126

Tabla 4.3: Resultados de la evolución de la probabilidad promedio de selección de cada algoritmo individual, medida cada 600 evaluaciones y acumulada en 30 ejecuciones, para el problema de la Mochila con el escenario: severidad 0.5 y frecuencia de cambio 6000.

algoritmos: Escalador de Colinas (EC), Búsqueda Aleatoria (BA), Recocido Simulado (RS), Búsqueda Tabú (BT), Estrategia Evolutiva (EE), Algoritmo de Estimación de Distribuciones (EDA).

La evolución de la probabilidad promedio de selección de cada algoritmo individual fue medida cada 600 evaluaciones de la función objetivo y acumulada en 30 ejecuciones. Se ilustran las 10 primeras muestras de los resultados. Analizando los datos obtenidos se puede observar como al inicio todos los algoritmos tienen la misma probabilidad de selección, y luego se van actualizando en la medida en que producen mejoras o empeoramientos. Se puede observar que para esta muestra de valores, los algoritmos que producen mejoras son el Recocido Simulado y la Búsqueda Tabú, mientras que los algoritmos que producen empeoramientos deben ser: Algoritmo Genético, Estrategia Evolutiva y Búsqueda Aleatoria. Este tipo de análisis puede ser muy interesante porque puede resultar que existan algoritmos que no se encuentren entre los mejores métodos de rendimiento cuando se ejecutan de forma individual y sin embargo pueden ser útiles en algunos momentos de la búsqueda cuando se combinan con otros métodos.

4.6. Sumario

En este capítulo se ha propuesto un framework para dar solución a problemas de optimización, estáticos y dinámicos. Para la modelación del diseño del framework se utilizó el lenguaje de modelado UML, y algunos artefactos como: vista de arquitectura, diagramas de clases, diagramas de actividad. El uso de este framework ofrece varias ventajas, de lo que podemos concluir:

- BiCIAM incluye diferentes algoritmos para resolver problemas de optimización tanto mono-objetivos como multi-objetivos, estáticos y dinámicos. Además, le reduce a los investigadores del esfuerzo de programación, permitiéndoles centrarse en el diseño de problema, el análisis de los resultados obtenidos y en la comparación entre algoritmos.
- Gracias al empleo del paradigma de orientación a objetos en BiCIAM, puede ser utilizado con cualquier máquina virtual de Java, con independencia del sistema operativo existente. Esto simplifica considerablemente el empleo del framework por parte de cualquier investigador.
- Amplía el rango de posibles usuarios de estos algoritmos. Es un framework fácil de usar, con una gran cantidad de propuestas de algoritmos ya implementados, por lo que reduce considerablemente el nivel de conocimientos y experiencia, requeridos a la hora de realizar un estudio en esta temática, siendo especialmente útil para investigadores con menor experiencia.
- En la actualidad se carece de frameworks que permitan modelar un portafolio de algoritmos adaptable a problemas estáticos y dinámicos. En este sentido la propuesta del framework constituye un paso de avance.
- El enfoque que se definió para la modelación de los problemas de optimización en el contexto de optimización estacionaria puede ser reutilizada en ambientes dinámicos, sin mayores cambios, solo depende del problema en cuestión y el tipo de dinamismo.
- BiCIAM provee un diseño simple para la inclusión de un nuevo algoritmo y modelación de cualquier tipo de problema. Además en la implementación se siguieron patrones de software bien conocidos, garantizando la robustez y flexibilidad del framework.
- BiCIAM gestiona eficientemente los datos que se pueden obtener como resultados de los experimentos, logrando con diferentes formatos de ficheros almacenar información de interés para los usuarios.

Este framework se ha utilizado para el desarrollo de todos los experimentos realizados en esta tesis, y ha evolucionado con ellos. Se encuentra disponible en el siguiente enlace:

`http://modo.ugr.es/algorithmportfolio/index.html`

Conclusiones y trabajos futuros

Dedicaremos esta sección a la presentación de un resumen de los resultados obtenidos y de las conclusiones que se pueden extraer de esta tesis. Presentaremos las publicaciones asociadas a la tesis y comentaremos algunos aspectos sobre trabajos futuros que siguen la línea aquí expuesta, y sobre otras líneas de investigación que se pueden derivar.

A. Conclusiones

La investigación realizada en esta tesis se ha centrado en estudiar, diseñar y evaluar esquemas de portafolios basados en metaheurísticas para abordar problemas de optimización dinámicos de tipo combinatorio; teniendo en mente las ideas de adaptación, cooperación y aprendizaje que consideramos indispensables en este contexto. Los objetivos propuestos para esta tesis fueron los siguientes:

- Realizar un estudio en profundidad en el ámbito de la Soft Computing, de cara a identificar las técnicas que se utilizan para resolver problemas de optimización dinámicos, incluyendo las diferentes posibilidades de hibridación de técnicas.
- Diseñar un portafolio de algoritmos que integre técnicas de adaptación, cooperación y aprendizaje, para resolver problemas de optimización dinámicos.
- Validar el funcionamiento del portafolio de algoritmos con respecto a algoritmos del estado del arte sobre problemas de optimización dinámicos combinatorios de prueba y reales.
- Proponer una librería de software para facilitar la reutilización y aplicación de metaheurísticas en la resolución de problemas de optimización estacionarios y dinámicos.

En primer lugar, con respecto al objetivo 1, del estudio del estado del arte realizado se puede concluir que los problemas de optimización combinatorios dinámicos pueden presentar diferentes tipos de cambios, lo cual influye considerablemente en la caracterización del

problema. En este sentido, existe diversidad en la aplicación de metaheurísticas para resolver los PODs, y se han desarrollado técnicas que mejoran el funcionamiento de los algoritmos de búsqueda e integran componentes de aprendizaje que guían de forma adaptativa el proceso de búsqueda.

Entre las técnicas utilizadas para resolver PODs destacan los portafolios de algoritmos que obtienen buenos resultados en estos problemas. Los portafolios propuestos en la literatura contemplan enfoques diferentes y combinan diferentes técnicas, en cuanto a: selección de los algoritmos, tipo de ejecución (secuencial o paralela) e intercambio de información durante el proceso de búsqueda. Además, los portafolios de algoritmos pueden estar formados por metaheurísticas poblacionales o de trayectoria, o por una combinación de ellas, pudiéndose ejecutar secuencialmente o en paralelo.

En relación al objetivo 2, se ha presentado un portafolio de algoritmos que incluye un esquema de aprendizaje. El portafolio contiene un conjunto de metaheurísticas que se ejecutan iterativamente. En cada iteración selecciona qué metaheurística aplicar utilizando un enfoque basado en el crédito que actúa como un esquema de aprendizaje. Se realizaron experimentos sobre cinco problemas combinatorios binarios de prueba a los que se les indujo dinamismo con el generador XOR, tales como: OneMax, Plateau, RoyalRoad, Deceptive y Mochila, en diferentes escenarios de severidad y frecuencia de cambio. Del estudio experimental realizado podemos concluir que solo cuatro de los ocho esquemas de aprendizaje previstos en los problemas analizados resultaron mejor que la variante del portafolio sin aprendizaje.

Se realizó un análisis de la evolución de la probabilidad de selección de los algoritmos en el portafolio. En este análisis se observó que los algoritmos que tienen un mal funcionamiento cuando se ejecutan individualmente pueden ser útiles cuando se combinan con otros métodos en diferentes momentos de la búsqueda. Además, se comparó el rendimiento de los algoritmos individuales que componen el portafolio, frente al propio portafolio. Se pudo afirmar que la mejor variante del portafolio obtiene resultados similares o significativamente mejores que las variantes individuales de las metaheurísticas que lo integran. También se pudo comprobar que el portafolio obtuvo resultados muy robustos en todos los problemas, en contraste con la variabilidad en el rendimiento de las metaheurísticas individuales.

Se comparó la mejor variante de aprendizaje del portafolio con dos algoritmos de la literatura que han reportado muy buenos resultados en estos problemas (objetivo 3). Los resultados del portafolio con respecto a AHMA y SORIGA fueron significativamente mejores en la mayoría de los problemas. De forma general, los resultados mostraron que el enfoque del portafolio de algoritmos con un esquema de aprendizaje simple, proporciona buenos resultados para resolver PODs. En esencia se pudo observar como variantes muy generales y básicas de metaheurísticas comunes, al trabajar juntas bajo un esquema sencillo de aprendizaje pueden proporcionar resultados competitivos con respecto a los métodos de

alto rendimiento para resolver PODs.

Otro de los aspectos asociado al objetivo 3, fue aplicar el portafolio a un problema dinámico real. Se realizó un estudio del problema de máxima cobertura dinámico (DMCLP). En este problema se plantea como objetivo encontrar la ubicación de las instalaciones que maximiza la demanda cubierta, es decir, la demanda que se encuentra dentro de un determinado tiempo o distancia de cobertura. Además se impone la restricción de que el número de instalaciones total (sobre todos los periodos de tiempo) sea igual a un cierto valor prefijado. Sobre la variante clásica de DMCLP se definieron dos variantes nuevas: DMCLP-AC y DMCLP-LocF, las cuales consisten en: incorporar costos por apertura/cierre de instalaciones de un periodo a otro y definir las instalaciones fijas a los largo del tiempo, respectivamente. Se realizaron experimentos con un portafolio compuesto por una recopilación de los algoritmos que se encuentran en la literatura con muy buenos resultados, y se comparó el portafolio con los algoritmos individuales que lo componen.

De forma general, cuando se consideraron los resultados del portafolio frente a los algoritmos individuales sobre las tres variantes del problema (DMCLP, DMCLP-AC y DMCLP-LocF), el portafolio fue mejor que los métodos individuales en cada una de las instancias, aunque estas diferencias no eran significativas. Sin embargo, de forma global, cuando agrupamos todas esas mejoras, entonces las diferencias si que llegan a ser significativas. En esencia, los resultados del portafolio de algoritmos para las tres variantes del DMCLP muestran que se crea una sinergia entre los algoritmos que lo componen, que de forma global es significativa con respecto a los algoritmos individuales.

Es importante destacar que los métodos individuales habían reportado muy buenos resultados para este problema, y son considerados como métodos de referencia en la literatura. También se realizó un análisis del comportamiento de las probabilidades de selección de los algoritmos en el portafolio. En el caso de las variantes DMCLP y DMCLP-LocF, el análisis mostró que ningún algoritmo destacó sobre los otros, ya que las probabilidades de selección no convergen. En el caso de la variante DMCLP-AC se pudo apreciar claramente que el portafolio tiende a utilizar la Estrategia Evolutiva y el Recocido Simulado Exponencial, siendo estos algoritmos los de mejor comportamiento cuando se ejecutaron de forma individual.

En términos generales, los resultados mostraron que el portafolio de algoritmos proporciona un buen comportamiento en DMCLP. Es importante tener en cuenta que este problema tiene muchas aplicaciones reales en la producción y los servicios, tales como: localización de estaciones de patrullas de policía, localización y redistribución de ambulancias o unidades de emergencia, localización de instalaciones públicas, cámaras de seguridad de intersección en una red de tráfico urbano, instalaciones de cadenas de suministros, entre otras.

Por último, para cumplir con el objetivo 4, se desarrolló una librería de software (framework) llamada BiCIAM que incluye diferentes algoritmos para resolver problemas de optimización tanto mono-objetivos como multi-objetivos, estáticos y dinámicos. BiCIAM le reduce a los investigadores el esfuerzo de programación, permitiéndoles centrarse en el diseño del problema, el análisis de los resultados obtenidos y en la comparación entre algoritmos. Se considera un framework fácil de usar, con una gran cantidad de algoritmos ya implementados, por lo que reduce el nivel de conocimiento requerido a la hora de realizar un estudio en esta temática.

B. Publicaciones Asociadas a la Tesis

A continuación se presenta un listado de las publicaciones asociadas a la tesis.

- Publicaciones en revistas internacionales:

  1. J. Fajardo, A. D. Masegosa, D. Pelta, Algorithm portfolio based scheme for dynamic optimization problems, International Journal of Computational Intelligence Systems, Vol. 8, No. 4, 667-689, 2015.
  2. J. Fajardo and A. Rosete. Algoritmo multigenerador de soluciones, para la competencia y colaboración de generadores metaheurísticos. Revista Internacional de Investigación de Operaciones, No. 1, 57-63, 2011.

- Publicaciones en congresos:

  1. J. Fajardo, A. D. Masegosa, A. Rosete D. Pelta, Adaptation Schemes and Dynamic Optimization Problems: A Basic Study on the Adaptive Hill Climbing Memetic Algorithm, In Nature Inspired Cooperative Strategies for Optimization (NICSO 2013), Studies in Computational Intelligence, Vol. 512, 85-97, 2013.
  2. J. Fajardo, A. D. Masegosa, A. Rosete D. Pelta, Aprendizaje en problemas de optimización dinámicos: un análisis sobre el algoritmo AHMA, IX Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, ISBN: 978-84-695-8348-7, 673-682, España 17-19 de septiembre 2013.
  3. J. Fajardo, J. R. González, D. Pelta, A. Rosete, Comparación de Escalador de Colinas y estrategias cooperativas en cuatro problemas de optimización dinámicos, VIII Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, ISBN: 978-84-615-6931-1, Albacete, España, 8 al 10 de febrero de 2012.
  4. J. Fajardo, A. D. Masegosa, D. Pelta, A. Rosete, Nuevos Esquemas Basados en SoftComputing para problemas Combinatorios Dinámicos, Foro de Innovaciones

Tecnológicas para el Transporte (TRANSNOVA 2012), Islas Canarias, España, 28 al 30 de junio 2012.

C. Trabajos Futuros

Esta última sección está dedicada a ofrecer las nuevas ideas, posibilidades de mejora y líneas de investigación que han surgido a partir del trabajo realizado en esta tesis. Se abren tres líneas principales de investigación:

- Aplicar el portafolio de algoritmos en nuevos problemas de optimización:
  - Analizar el funcionamiento del portafolio de algoritmos en los problemas de optimización combinatorios clásicos de la literatura, y problemas que son de interés para la CUJAE (Instituto Superior Politécnico Jose Antonio Echeverría), tales como: trazado de grafos, viajante de comercio, ruteo de vehículos, asignación de recursos humanos a equipos de proyectos de software y planificación de tareas.
  - Analizar el rendimiento del portafolio de algoritmos en PODs continuos de la literatura.
  - Estudiar el funcionamiento del portafolio de algoritmos frente a problemas reales de optimización dinámicos con multiples-objetivos.
- Estudiar posibles mejoras del portafolio de algoritmos:
  - Diseñar otros esquemas de asignación de créditos y enfoques de adaptación para el funcionamiento del portafolio, y comparar los resultados con los presentados en esta tesis.
  - Incorporar nuevas estrategias de selección de algoritmos en el portafolio y estudiar su funcionamiento.
  - Estudiar con más detalle el funcionamiento interno del portafolio en PODs, analizando específicamente los siguientes aspectos: redundancia en las soluciones que generan los algoritmos, efectividad de los algoritmos, evolución de los créditos y auto-adaptación de los algoritmos cuando ocurre un cambio.
- Ampliar y mejorar el framework BiCIAM:
  - Incorporar en BiCIAM algoritmos del estado del arte para resolver problemas de optimización, tales como: Colonia de Hormigas, Algoritmos Meméticos, o Programación Genética, entre otros.

- Incorporar un módulo en BiCIAM que contenga las medidas de rendimiento propuestas en la literatura para evaluar el rendimiento de los algoritmos frente a los problemas de optimización dinámicos.
- Diseñar e implementar una interfaz gráfica para el diseño de experimentos y visualización de los resultados, así como para realizar comparaciones y pruebas estadísticas no paramétricas.
- Desarrollar un módulo para heurísticas de construcción de soluciones, en problemas específicos.

Apéndice A

Resultados del estudio experimental del portafolio de algoritmos

Este apéndice contiene los resultados obtenidos en los estudios experimentales desarrollados para evaluar la efectividad de los métodos presentados sobre cada uno de los problemas: OneMax, Plateau, RoyalRoad, Deceptive y Mochila.

		OneMax				Plateau				RoyalRoad				Deceptive				Knapsack			
		severity				severity				severity				severity				severity			
	change	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
RS-IP-REB	1200	96.697	93.627	88.183	88.009	92.951	85.863	73.776	73.967	75.169	59.801	44.084	40.929	18.201	17.282	16.710	16.893	1674.750	1668.979	1665.183	1674.889
	3000	98.650	97.330	94.934	94.932	96.861	93.046	86.141	86.149	83.193	71.834	56.033	51.294	20.123	19.249	18.599	18.599	1679.836	1674.244	1670.331	1679.611
	6000	99.326	98.674	97.472	97.461	98.274	95.740	91.089	91.197	87.647	77.650	63.426	58.637	21.726	20.413	19.931	19.707	1684.083	1677.994	1674.314	1683.923
	9000	99.540	99.117	98.311	98.300	98.789	96.696	92.894	92.975	89.458	80.805	67.657	63.931	22.679	21.388	20.829	20.702	1687.852	1680.461	1676.542	1687.469
	12000	99.662	99.339	98.731	98.724	99.023	97.227	93.892	94.069	90.199	81.972	70.500	67.099	23.729	22.160	21.690	21.353	1689.696	1682.967	1678.778	1689.932
RS-IP-RB	1200	97.363	94.564	87.864	87.637	94.402	87.784	71.548	71.467	77.185	60.925	43.184	40.244	19.714	18.985	18.339	18.346	1675.406	1669.828	1664.828	1675.449
	3000	98.942	97.811	95.086	95.024	97.736	94.774	86.920	86.983	87.960	75.308	56.465	52.667	22.491	21.438	20.995	20.737	1681.025	1675.253	1671.222	1681.000
	6000	99.473	98.907	97.564	97.510	98.841	97.207	92.955	93.053	92.340	82.820	66.147	63.261	24.206	23.468	22.815	22.562	1686.174	1680.274	1676.124	1687.073
	9000	99.649	99.271	98.370	98.328	99.203	98.074	95.131	95.151	94.125	86.425	71.761	70.069	25.510	24.476	23.928	23.675	1690.165	1682.660	1678.850	1689.953
	12000	99.737	99.455	98.778	98.752	99.395	98.541	96.299	96.277	94.902	88.690	75.223	74.054	26.851	25.399	24.765	24.664	1692.488	1685.421	1681.317	1692.558
RS-AP-REB	1200	97.321	94.353	87.360	87.149	94.250	87.546	71.518	71.416	77.925	61.276	43.467	40.495	19.762	18.644	17.992	17.596	1672.132	1667.715	1663.481	1672.086
	3000	98.924	97.731	94.904	94.810	97.624	94.602	86.541	86.573	87.756	75.211	56.438	51.641	22.002	21.181	20.400	20.214	1674.764	1670.086	1666.705	1675.079
	6000	99.462	98.865	97.441	97.404	98.792	97.109	92.653	92.698	92.218	82.934	66.019	62.806	23.953	22.899	22.351	22.221	1675.907	1671.159	1668.374	1675.161
	9000	99.640	99.238	98.296	98.273	99.177	97.968	94.724	94.865	94.030	86.067	70.856	68.555	25.454	24.058	23.503	23.210	1676.288	1671.747	1668.235	1676.354
	12000	99.728	99.429	98.721	98.706	99.353	98.417	95.928	95.922	94.946	88.001	74.934	72.949	26.417	24.717	24.434	24.067	1676.373	1671.604	1668.943	1676.417
RS-AP-RB	1200	99.532	99.266	99.617	99.544	98.739	95.148	86.702	85.823	92.048	92.452	84.869	82.448	20.195	18.998	18.431	18.367	1675.046	1669.866	1665.136	1675.534
	3000	99.007	98.096	96.215	98.167	98.839	95.425	89.887	89.925	89.058	77.949	61.739	79.613	22.407	21.855	21.060	20.933	1682.253	1675.598	1671.495	1681.525
	6000	99.507	99.047	98.105	98.091	98.878	97.591	94.667	94.770	92.914	85.540	73.682	73.129	24.589	23.488	23.006	22.973	1686.111	1680.050	1676.114	1685.967
	9000	99.670	99.363	98.740	98.725	99.228	98.314	96.345	96.384	94.455	88.680	79.680	79.797	25.917	25.024	24.265	24.001	1689.472	1682.749	1678.951	1689.532
	12000	99.750	99.523	99.056	99.042	99.429	98.720	97.231	97.194	95.329	90.588	93.004	93.013	27.272	25.686	25.304	24.927	1691.908	1684.759	1681.273	1691.020
NRS-IP-REB	1200	95.195	90.859	86.096	87.285	89.885	82.284	71.610	73.837	74.705	60.427	40.266	37.505	15.349	14.593	13.986	14.462	1675.527	1669.410	1664.945	1675.560
	3000	98.042	95.894	92.528	93.357	95.506	90.181	83.431	84.105	82.838	71.478	52.451	43.020	17.284	16.034	15.224	15.753	1681.742	1675.308	1671.148	1681.524
	6000	99.008	97.885	95.590	95.696	97.134	93.402	87.985	90.382	86.181	77.351	60.528	47.917	19.355	17.585	16.564	17.183	1686.731	1679.761	1675.553	1686.680
	9000	99.337	98.576	96.656	97.160	98.128	94.737	89.811	91.811	87.860	80.065	65.131	49.795	21.020	18.957	17.636	18.274	1689.568	1682.132	1678.363	1689.815
	12000	99.511	98.932	97.172	97.301	98.518	95.368	90.601	92.256	88.372	81.515	68.120	51.566	22.070	19.566	17.852	18.747	1692.428	1684.379	1680.528	1692.290
NRS-IP-RB	1200	97.496	94.488	86.434	86.146	94.653	87.671	69.833	69.350	78.607	60.814	43.116	40.024	20.012	19.111	18.463	18.476	1675.520	1669.685	1664.952	1675.860
	3000	98.972	97.791	94.466	94.332	97.785	94.686	85.306	85.377	88.181	75.075	55.933	51.808	22.428	21.499	21.073	20.852	1681.993	1675.298	1671.091	1681.263
	6000	99.486	98.891	97.210	97.182	98.860	97.179	92.118	92.074	92.402	83.072	65.364	61.482	24.632	23.363	22.801	22.625	1686.723	1679.341	1675.554	1686.996
	9000	99.659	99.259	98.147	98.120	99.228	98.062	94.443	94.403	94.110	86.124	70.421	68.465	25.682	24.400	24.133	23.792	1689.640	1682.575	1678.596	1689.953
	12000	99.745	99.449	98.607	98.590	99.379	98.481	95.600	95.649	94.944	88.181	74.229	72.275	26.574	25.345	24.723	24.686	1692.360	1684.905	1680.654	1692.137
NRS-AP-REB	1200	97.441	94.457	86.258	86.063	94.616	87.441	69.535	69.256	78.241	60.636	42.639	39.959	19.879	19.164	18.190	18.326	1659.158	1647.679	1630.299	1660.947
	3000	98.973	97.743	94.377	94.268	97.784	94.672	85.390	85.327	88.111	74.814	55.847	51.629	22.233	21.337	20.791	20.626	1671.098	1654.507	1641.605	1673.837
	6000	99.486	98.879	97.211	97.146	98.833	97.122	92.024	91.999	92.415	82.778	65.419	61.875	24.344	23.263	22.818	22.360	1676.858	1660.464	1648.235	1671.442
	9000	99.657	99.257	98.133	98.103	99.191	98.000	94.389	94.394	93.932	86.067	70.303	68.329	25.527	24.556	23.758	23.814	1678.867	1670.334	1657.083	1678.964
	12000	99.743	99.446	98.603	98.584	99.408	98.479	95.608	95.591	94.793	88.177	74.102	72.589	26.556	25.123	24.606	24.433	1678.505	1669.627	1637.374	1678.866
NRS-AP-RB	1200	89.827	86.918	73.941	76.055	81.079	73.119	51.658	53.481	69.108	60.407	54.910	51.738	17.710	16.308	16.101	15.808	1662.259	1656.879	1650.774	1661.837
	3000	85.416	81.313	81.423	79.265	75.669	64.629	62.475	61.077	76.467	64.109	50.036	50.735	21.173	19.783	18.875	19.068	1670.644	1666.472	1661.524	1671.387
	6000	97.360	89.784	89.206	90.113	94.805	87.656	80.108	79.627	88.007	78.186	64.692	63.529	24.386	23.318	22.556	22.506	1677.665	1674.624	1670.708	1679.441
	9000	98.261	96.288	93.037	93.369	96.596	92.771	86.053	86.608	91.951	83.735	71.930	72.178	26.382	25.237	24.296	24.495	1683.665	1678.657	1674.753	1684.315
	12000	98.700	97.442	95.130	95.303	97.428	94.571	88.966	90.208	93.423	87.068	77.265	77.509	27.652	26.108	25.800	25.670	1686.898	1682.265	1679.556	1686.938

Tabla A.1: Offline performance obtenido en 30 ejecuciones por cada esquemas de aprendizaje considerados en el portafolio de algoritmos en la experimentación para cada problema, severidad y frecuencia de cambio.

		OneMax				Plateau				RoyalRoad				Deceptive				Knapsack			
		severity				severity				severity				severity				severity			
	change	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
HC	1200	99.197	98.384	95.920	92.639	98.258	96.182	89.746	83.748	89.268	80.940	66.838	60.687	11.972	11.915	11.980	13.516	1676.305	1671.407	1665.839	1675.655
	3000	99.682	99.348	98.369	97.043	99.298	98.459	95.909	93.519	95.719	92.163	85.759	82.872	11.988	11.972	16.766	17.594	1677.620	1672.574	1668.479	1677.862
	6000	99.840	99.673	99.183	98.523	99.656	99.234	97.953	96.769	97.838	96.089	92.901	91.411	11.996	11.990	18.077	18.922	1678.150	1673.188	1668.759	1679.739
	9000	99.893	99.783	99.453	99.015	99.768	99.491	98.636	97.844	98.566	97.411	95.307	94.281	11.997	11.992	18.665	19.271	1678.829	1673.836	1669.775	1679.033
	12000	99.920	99.837	99.592	99.259	99.829	99.619	98.981	98.372	98.926	98.041	96.461	95.708	11.999	11.995	19.137	19.456	1679.537	1673.282	1669.751	1679.821
RS	1200	65.550	64.894	64.653	64.644	39.346	38.548	38.245	38.138	24.437	24.000	23.727	23.767	18.818	18.652	18.622	18.610	1657.082	1656.955	1656.826	1656.963
	3000	66.721	66.219	65.969	65.915	41.268	40.469	40.118	40.183	26.914	26.475	26.415	26.413	19.598	19.559	19.519	19.497	1663.341	1663.146	1663.080	1663.413
	6000	67.521	67.052	66.919	66.874	42.285	41.784	41.551	41.618	27.889	27.630	27.553	27.496	19.976	19.933	19.917	19.910	1667.171	1666.993	1667.124	1667.158
	9000	67.998	67.614	67.467	67.482	43.038	42.582	42.492	42.430	28.397	28.066	27.939	27.978	21.872	21.490	21.574	21.573	1669.207	1669.229	1669.017	1669.082
	12000	68.340	67.948	67.844	67.790	43.645	43.104	43.062	43.055	29.526	29.162	29.075	29.063	23.233	23.210	23.229	23.021	1670.454	1670.362	1670.198	1670.487
TS	1200	60.265	60.183	60.115	60.193	32.217	32.151	32.168	32.149	18.503	18.537	18.528	18.537	13.490	13.578	13.217	13.469	1636.800	1636.055	1636.392	1636.426
	3000	62.245	62.365	62.213	62.273	35.168	35.219	35.153	35.154	21.287	21.459	21.283	21.305	17.035	16.904	16.990	16.921	1652.397	1652.123	1652.610	1652.826
	6000	63.770	63.690	63.727	63.708	37.205	37.125	37.131	37.252	23.180	23.288	23.174	23.209	18.344	18.347	18.371	18.348	1660.527	1660.413	1660.311	1660.419
	9000	64.481	64.473	64.564	64.546	38.274	38.352	38.126	38.254	24.129	24.000	24.275	24.110	18.830	18.852	18.868	18.871	1663.741	1664.157	1663.850	1663.749
	12000	65.027	65.013	65.054	65.026	39.143	39.101	39.105	39.043	25.207	25.186	25.370	25.161	19.257	19.194	19.223	19.232	1666.039	1666.021	1666.035	1666.191
SA	1200	62.523	62.610	62.322	62.088	36.643	36.493	36.154	35.933	22.105	21.875	21.890	21.650	13.051	12.865	12.857	12.782	1669.539	1668.425	1666.993	1669.549
	3000	64.930	64.937	64.810	64.634	40.244	40.158	40.034	39.757	25.947	25.917	25.698	25.610	17.069	17.074	17.122	16.818	1674.405	1673.263	1673.662	1674.654
	6000	66.615	66.581	66.627	66.554	42.911	42.938	42.775	42.768	28.652	28.683	28.343	28.352	18.615	18.584	18.684	18.539	1677.910	1676.437	1677.128	1678.012
	9000	67.532	67.480	67.480	67.423	44.438	44.394	44.227	44.257	30.340	30.245	30.233	30.081	19.247	19.240	19.273	19.219	1680.058	1678.683	1678.785	1679.600
	12000	68.189	68.087	68.016	67.969	45.469	45.418	45.232	45.269	31.251	31.259	31.109	31.153	19.641	19.587	19.591	19.617	1681.037	1679.657	1679.558	1680.794
GA	1200	96.216	91.330	74.958	58.341	91.788	80.084	50.348	44.777	64.647	43.782	24.990	33.893	11.803	11.000	10.701	11.000	1662.625	1649.844	1614.986	1662.139
	3000	98.518	96.503	88.401	74.220	96.815	90.894	67.155	49.101	75.416	54.964	31.542	38.823	11.998	11.760	11.091	11.801	1671.867	1661.739	1642.618	1672.269
	6000	99.262	98.264	94.202	86.971	98.416	94.917	75.969	51.863	79.464	60.332	35.031	41.308	12.000	11.936	11.664	11.946	1676.324	1666.720	1652.077	1676.947
	9000	99.511	98.844	96.142	91.297	98.960	96.482	79.285	53.469	82.451	62.888	37.404	42.410	12.000	11.947	11.790	11.973	1677.934	1669.617	1656.146	1677.686
	12000	99.631	99.132	97.098	93.479	99.216	97.615	81.024	54.213	82.912	64.791	38.573	42.848	12.000	11.970	11.851	11.982	1678.553	1669.102	1657.887	1678.778
EE	1200	95.793	88.685	70.493	56.554	91.008	75.659	46.331	42.389	66.570	45.620	26.410	33.078	11.887	11.274	11.000	11.649	1662.833	1649.219	1630.740	1662.719
	3000	98.399	95.677	84.508	67.933	96.647	89.959	65.792	48.535	76.919	57.889	34.027	38.150	11.976	11.859	11.566	11.860	1671.422	1660.911	1642.799	1672.319
	6000	99.201	97.833	92.211	82.942	98.342	94.761	78.203	54.035	81.406	64.034	38.173	41.213	11.992	11.943	11.774	11.958	1676.044	1666.298	1652.384	1675.258
	9000	99.468	98.549	94.830	88.621	98.891	96.589	83.001	58.020	83.211	66.209	40.996	42.147	11.990	11.959	11.878	11.966	1678.461	1668.898	1656.286	1677.548
	12000	99.603	98.917	96.116	91.477	99.166	97.449	85.251	61.961	83.910	69.133	42.627	42.432	12.000	11.972	11.912	11.984	1678.946	1669.402	1658.208	1678.212
EDA	1200	51.879	50.645	50.247	50.874	20.987	18.256	18.011	18.038	15.654	18.995	16.654	17.002	9.556	9.111	8.776	9.554	1662.933	1650.063	1381.000	1661.928
	3000	51.654	50.213	50.749	55.845	18.587	18.558	18.841	18.446	15.655	15.699	15.325	15.967	8.997	9.223	9.568	9.668	1672.846	1661.352	1641.698	1672.438
	6000	51.652	50.878	50.625	50.451	20.211	18.659	19.654	18.965	11.558	15.694	16.875	15.874	9.351	9.662	8.231	9.884	1675.443	1666.444	1652.521	1676.018
	9000	51.635	51.524	50.894	52.451	20.111	18.654	18.963	18.549	16.854	13.325	16.895	16.552	9.622	9.304	9.002	9.684	1677.082	1668.926	1656.387	1677.307
	12000	51.623	51.841	50.254	50.647	20.249	20.654	18.122	18.998	17.654	16.215	16.888	17.995	9.335	9.664	8.399	9.645	1678.552	1669.649	1658.087	1678.341

Tabla A.2: Offline performance obtenido en 30 ejecuciones por las variantes independientes de los algoritmos que componen el portafolio para cada problema, severidad y frecuencia de cambio.

		OneMax				Plateau				RoyalRoad				Deceptive				Knapsack			
		severity				severity				severity				severity				severity			
	change	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
AHMA	1200	97.648	95.355	90.662	90.607	95.073	89.423	76.900	77.154	79.723	62.762	45.140	42.663	65.123	57.088	53.930	65.736	1679.172	1671.414	1663.916	1669.617
	3000	99.060	98.126	96.236	96.179	98.037	95.763	90.229	90.253	89.845	78.694	62.560	61.201	73.112	64.655	56.092	75.094	1693.010	1681.298	1667.938	1677.169
	6000	99.531	99.069	98.119	98.097	99.023	97.897	95.118	95.180	93.771	86.590	76.134	75.379	78.593	70.534	62.542	80.269	1701.750	1689.347	1671.447	1682.675
	9000	97.688	99.379	98.750	98.727	99.350	98.595	96.739	96.780	95.019	89.807	82.400	82.514	81.574	74.068	65.763	82.302	1705.639	1694.148	1674.089	1688.286
	12000	99.763	99.532	99.060	99.049	99.512	98.943	97.580	97.581	95.731	91.983	85.730	85.788	83.433	76.954	67.740	83.265	1707.703	1697.228	1676.013	1692.966
SORIGA	1200	90.016	81.473	67.956	78.296	81.042	65.262	42.366	59.845	61.767	42.152	26.158	38.683	37.055	35.562	34.111	36.985	1664.800	1664.800	1664.800	1664.600
	3000	94.362	88.155	73.357	88.395	89.072	76.935	51.593	79.263	77.991	58.863	34.353	58.015	37.493	35.244	34.454	35.989	665.920	665.920	665.760	665.920
	6000	96.259	91.274	78.305	90.896	92.677	82.238	57.556	82.715	85.276	69.339	40.614	73.028	39.665	35.925	34.322	36.554	332.940	332.980	332.920	332.960
	9000	97.283	93.250	82.096	92.090	94.657	86.391	61.528	83.634	88.848	74.916	44.886	75.672	40.364	36.345	34.672	36.000	221.960	221.973	221.973	221.947
	12000	97.914	94.420	84.204	93.044	95.945	88.850	64.558	84.777	90.973	77.586	47.633	75.622	41.358	37.051	34.959	36.957	166.450	166.490	166.460	166.480

Tabla A.3: Offline performance obtenido por AHMA y SORIGA para cada problema, severidad y frecuencia de cambio

Apéndice B

Resultados del estudio experimental con el problema de máxima cobertura dinámico

El siguiente apéndice contiene los resultados obtenidos en los estudios experimentales desarrollados con el problema de máxima cobertura dinámico. Este apéndice se divide en tres secciones para mostrar los resultados de las variantes del problema: clásica (DMCLP), con costo por apertura o cierre de instalaciones (DMCLP-AC) y localizaciones fijas (DMCLP-LocF), respectivamente.

B.1. Resultados DMCLP

		1800				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**68.04**	67.62	65.96	64.05	66.44
	70	**77.48**	76.87	75.40	73.10	75.71
	75	**82.24**	81.58	79.59	77.41	80.32
	80	**89.92**	88.84	87.64	85.06	88.00
	85	**92.90**	92.08	90.39	88.47	91.47
4.5	65	**71.29**	70.48	69.57	67.08	69.79
	70	**80.04**	79.64	78.10	75.45	78.57
	75	**85.21**	84.45	82.80	80.23	83.12
	80	**88.45**	87.19	85.18	83.38	86.64
	85	**94.52**	93.51	92.24	90.26	93.23
5	65	**74.52**	74.06	72.74	70.10	72.86
	70	**79.02**	78.66	76.82	74.68	77.21
	75	**87.64**	87.03	85.12	82.82	85.64
	80	**91.07**	89.81	88.36	86.29	89.20
	85	**96.03**	95.29	93.91	91.72	94.58

Tabla B.1: Resultados del problema DMCLP para la instancia de 1800 nodos de demanda y 100 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		2000				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**70.45**	69.15	67.75	66.20	68.78
	70	**73.97**	73.25	71.52	68.74	71.87
	75	**77.07**	75.72	74.87	72.21	75.07
	80	**79.94**	79.35	77.29	74.91	77.93
	85	**82.61**	81.52	79.93	77.26	80.33
4.5	65	**80.77**	79.76	78.40	75.39	78.58
	70	**83.88**	82.65	81.32	79.06	81.57
	75	**86.53**	85.78	84.20	81.46	84.55
	80	**89.37**	88.76	86.40	83.53	86.96
	85	**91.25**	90.22	88.40	86.08	89.31
5	65	**88.99**	88.25	85.82	83.17	86.71
	70	**91.57**	90.09	89.31	86.10	89.78
	75	**93.65**	92.18	90.89	88.41	91.71
	80	**95.33**	94.24	92.88	90.85	93.43
	85	**96.66**	95.87	94.40	92.38	95.03

Tabla B.2: Resultados del problema DMCLP para la instancia de 2000 nodos de demanda y 150 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		2500				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**69.29**	68.31	66.70	64.51	67.03
	70	**72.91**	72.25	70.35	67.82	70.81
	75	**76.01**	75.47	73.47	71.42	73.85
	80	**78.94**	77.78	76.47	73.88	76.61
	85	**81.73**	80.79	78.79	76.68	79.49
4.5	65	**79.86**	78.73	77.09	74.81	77.78
	70	**82.97**	82.29	79.84	77.34	80.72
	75	**86.10**	85.06	83.13	80.57	83.63
	80	**88.51**	87.54	85.47	82.89	86.39
	85	**90.52**	89.38	87.83	85.19	87.99
5	65	**88.60**	87.76	85.80	83.17	86.48
	70	**91.08**	90.54	88.17	85.84	88.81
	75	**93.35**	91.73	90.66	87.69	91.39
	80	**94.84**	94.02	92.73	89.51	93.10
	85	**96.43**	95.20	93.18	91.57	94.71

Tabla B.3: Resultados del problema DMCLP para la instancia de 2500 nodos de demanda y 200 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		1800				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**307828.50**	305932.00	298419.50	289794.50	300608.50
	70	**322536.00**	318861.50	314736.00	303506.00	315734.50
	75	**337171.50**	335090.50	329098.50	317160.50	329623.50
	80	**350563.50**	347769.50	341150.50	330727.00	342533.50
	85	**362137.50**	360334.00	353365.50	341372.50	355459.00
4.5	65	**357508.00**	355867.00	347552.00	337891.50	349309.00
	70	**372081.50**	369081.00	360072.50	350249.00	363395.50
	75	**385507.50**	382101.50	374629.50	362975.00	376078.00
	80	**396498.00**	393747.00	385133.00	374698.50	387484.00
	85	**406813.50**	401951.00	396495.50	384838.50	398129.50
5	65	**400181.0**0	394499.50	385361.50	377244.00	392008.50
	70	**412030.00**	406333.50	399750.50	390408.00	403560.50
	75	**420331.00**	416593.00	408965.50	400272.00	413843.00
	80	**427639.00**	423078.50	417343.00	408352.00	421799.00
	85	**434464.50**	431107.00	424899.50	414992.50	427915.00

Tabla B.4: Resultados del problema DMCLP para la instancia de 1800 nodos de demanda y 100 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

		2000				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**353960.00**	347410.00	340373.00	332573.00	345526.50
	70	**371622.50**	367997.50	359291.00	345373.50	361091.50
	75	**387171.00**	380431.50	376150.50	362794.50	377140.50
	80	**401614.50**	398666.00	388280.00	376321.00	391529.00
	85	**415020.50**	409525.50	401563.00	388170.50	403581.50
4.5	65	**405789.00**	400704.50	393890.50	378753.50	394768.00
	70	**421406.00**	415248.50	408560.00	397206.50	409807.00
	75	**434733.00**	430952.50	423037.00	409251.50	424770.50
	80	**448983.00**	445910.00	434043.00	419635.50	436892.50
	85	**458419.50**	453279.50	444094.00	432454.50	448665.00
5	65	**447069.00**	443353.50	431172.50	417855.00	435609.50
	70	**460030.00**	452614.00	448677.00	432544.50	451025.50
	75	**470494.50**	463125.00	456636.00	444158.50	460758.50
	80	**478949.00**	473471.00	466633.50	456445.00	469364.50
	85	**485588.50**	481635.00	474252.00	464127.50	477437.50

Tabla B.5: Resultados del problema DMCLP para la instancia de 2000 nodos de demanda y 150 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

		2500				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**437324.00**	431133.00	420958.00	407108.50	423010.50
	70	**460114.50**	455992.00	444001.00	428015.00	446867.50
	75	**479731.00**	476301.00	463668.50	450759.00	466067.00
	80	**498172.50**	490866.00	482638.50	466283.50	483482.50
	85	**515783.00**	509885.50	497219.50	483932.00	501694.50
4.5	65	**503979.50**	496848.50	486515.00	472150.50	490849.50
	70	**523634.50**	519317.00	503896.50	488095.00	509434.50
	75	**543369.00**	536852.00	524647.00	508498.00	527776.00
	80	**558604.50**	552475.50	539435.00	523146.00	545193.50
	85	**571280.00**	564091.50	554278.50	537638.50	555338.00
5	65	**559131.00**	553880.50	541499.50	524913.00	545774.00
	70	**574818.00**	571400.00	556454.00	541764.00	560480.50
	75	**589151.00**	578945.50	572152.00	553415.50	576756.50
	80	**598561.00**	593336.50	585247.00	564903.00	587540.00
	85	**608565.00**	600788.50	588087.50	577937.00	597745.00

Tabla B.6: Resultados del problema DMCLP para la instancia de 2500 nodos de demanda y 200 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

B.2. Resultados DMCLP con costo de apertura y cierre

		1800				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**74.34**	62.48	61.30	61.55	73.03
	70	**76.64**	64.09	62.78	62.86	75.18
	75	**78.59**	65.41	64.09	63.98	76.95
	80	**80.25**	66.27	65.21	65.31	78.75
	85	**82.20**	67.23	66.25	66.56	80.55
4.5	65	**81.75**	68.86	67.49	67.82	80.20
	70	**83.46**	70.62	69.51	69.11	82.18
	75	**85.31**	71.67	70.18	70.52	83.79
	80	**86.99**	72.86	71.72	71.75	85.46
	85	**88.59**	74.32	72.80	72.63	87.01
5	65	**88.35**	75.55	73.17	73.35	86.96
	70	**89.80**	76.52	74.77	74.95	88.20
	75	**91.00**	78.11	76.10	75.77	89.54
	80	**92.12**	78.55	77.11	76.87	90.80
	85	**93.11**	79.75	78.07	78.09	91.95

Tabla B.7: Resultados del problema DMCLP-AC para la instancia de 1800 nodos de demanda y 100 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		2000				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**76.30**	64.50	63.66	63.67	75.04
	70	**78.66**	66.11	65.19	65.41	76.93
	75	**80.53**	67.79	66.75	66.60	78.93
	80	**82.31**	69.16	68.08	68.01	80.74
	85	**83.66**	70.00	69.20	69.32	82.19
4.5	65	**83.24**	71.16	69.89	69.77	81.72
	70	**85.21**	72.75	71.27	71.29	83.77
	75	**86.94**	74.01	72.91	72.73	85.10
	80	**88.63**	75.66	74.10	74.12	86.86
	85	**89.91**	76.46	75.59	75.35	88.64
5.0	65	**89.07**	76.85	75.05	75.05	87.46
	70	**90.69**	78.59	76.80	76.63	89.34
	75	**91.84**	79.39	77.93	77.98	90.48
	80	**92.86**	81.20	79.29	79.26	91.45
	85	**93.95**	82.03	80.39	80.27	92.57

Tabla B.8: Resultados del problema DMCLP-AC para la instancia de 2000 nodos de demanda y 150 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		2500				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**75.86**	65.71	64.53	64.33	74.63
	70	**78.24**	67.19	66.18	66.19	76.94
	75	**80.22**	68.67	67.88	67.63	78.99
	80	**82.19**	70.21	69.24	69.13	80.41
	85	**83.72**	71.30	70.65	70.48	82.43
4.5	65	**83.09**	72.62	70.35	70.77	81.78
	70	**85.26**	73.71	72.14	72.36	83.71
	75	**86.88**	75.17	73.83	73.62	85.37
	80	**88.75**	76.39	75.13	75.17	87.17
	85	**89.93**	78.02	76.65	76.49	88.60
5.0	65	**89.02**	77.75	76.19	76.32	87.80
	70	**90.63**	79.21	77.90	77.87	89.40
	75	**92.25**	80.60	79.31	79.32	90.81
	80	**93.31**	81.84	80.58	80.63	92.11
	85	**94.26**	83.44	81.60	81.77	92.64

Tabla B.9: Resultados del problema DMCLP-AC para la instancia de 2500 nodos de demanda y 200 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		1800				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**0.743**	0.625	0.613	0.616	0.730
	70	**0.766**	0.641	0.628	0.629	0.752
	75	**0.786**	0.654	0.641	0.640	0.770
	80	**0.803**	0.663	0.652	0.653	0.787
	85	**0.822**	0.672	0.663	0.666	0.806
4.5	65	**0.818**	0.689	0.675	0.678	0.802
	70	**0.835**	0.706	0.695	0.691	0.822
	75	**0.853**	0.717	0.702	0.705	0.838
	80	**0.870**	0.729	0.717	0.717	0.855
	85	**0.886**	0.743	0.728	0.726	0.870
5	65	**0.884**	0.755	0.732	0.733	0.870
	70	**0.898**	0.765	0.748	0.750	0.882
	75	**0.910**	0.781	0.761	0.758	0.895
	80	**0.921**	0.785	0.771	0.769	0.908
	85	**0.931**	0.798	0.781	0.781	0.920

Tabla B.10: Resultados del problema DMCLP-AC para la instancia de 1800 nodos de demanda y 100 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

		2000				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**0.763**	0.645	0.637	0.637	0.750
	70	**0.787**	0.661	0.652	0.654	0.769
	75	**0.805**	0.678	0.668	0.666	0.789
	80	**0.823**	0.692	0.681	0.680	0.807
	85	**0.837**	0.700	0.692	0.693	0.822
4.5	65	**0.832**	0.712	0.699	0.698	0.817
	70	**0.852**	0.728	0.713	0.713	0.838
	75	**0.869**	0.740	0.729	0.727	0.851
	80	**0.886**	0.757	0.741	0.741	0.869
	85	**0.899**	0.765	0.756	0.753	0.886
5	65	**0.891**	0.768	0.750	0.750	0.875
	70	**0.907**	0.786	0.768	0.766	0.893
	75	**0.918**	0.794	0.779	0.780	0.905
	80	**0.929**	0.812	0.793	0.793	0.914
	85	**0.939**	0.820	0.804	0.803	0.926

Tabla B.11: Resultados del problema DMCLP-AC para la instancia de 2000 nodos de demanda y 150 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

		2500				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**0.759**	0.657	0.645	0.643	0.746
	70	**0.782**	0.672	0.662	0.662	0.769
	75	**0.802**	0.687	0.679	0.676	0.790
	80	**0.822**	0.702	0.692	0.691	0.804
	85	**0.837**	0.713	0.706	0.705	0.824
4.5	65	**0.831**	0.726	0.703	0.708	0.818
	70	**0.853**	0.737	0.721	0.724	0.837
	75	**0.869**	0.752	0.738	0.736	0.854
	80	**0.888**	0.764	0.751	0.752	0.872
	85	**0.899**	0.780	0.767	0.765	0.886
5	65	**0.890**	0.778	0.762	0.763	0.878
	70	**0.906**	0.792	0.779	0.779	0.894
	75	**0.923**	0.806	0.793	0.793	0.908
	80	**0.933**	0.818	0.806	0.806	0.921
	85	**0.943**	0.834	0.816	0.818	0.926

Tabla B.12: Resultados del problema DMCLP-AC para la instancia de 2500 nodos de demanda y 200 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

B.3. Resultados DMCLP con localizaciones fijas

		1800				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4	65	**68.99**	65.72	66.19	66.20	67.08
	70	**72.33**	69.49	69.19	69.58	70.51
	75	**75.47**	72.72	72.24	72.38	73.85
	80	**78.64**	75.17	74.85	75.73	76.60
	85	**81.85**	78.25	77.62	78.64	79.20
4.5	65	**80.25**	76.29	76.58	77.37	78.33
	70	**83.48**	79.98	80.18	80.60	81.41
	75	**86.56**	82.75	82.71	83.43	84.10
	80	**89.38**	85.94	85.63	85.30	86.42
	85	**91.62**	87.33	87.82	87.94	88.82
5	65	**89.72**	86.78	86.88	86.85	87.69
	70	**92.02**	88.83	89.43	89.77	90.13
	75	**94.36**	91.11	91.40	91.46	92.25
	80	**96.03**	92.51	93.09	93.07	93.88
	85	**97.30**	94.30	94.38	94.57	95.48

Tabla B.13: Resultados del problema DMCLP-LocF para la instancia de 1800 nodos de demanda y 100 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		2000				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**71.90**	68.34	68.51	68.76	69.42
	70	**75.44**	71.30	71.49	71.65	72.99
	75	**78.96**	74.16	75.22	75.16	76.01
	80	**82.00**	77.08	77.38	77.61	78.73
	85	**84.72**	80.55	79.87	79.92	81.39
4.5	65	**83.04**	78.41	78.54	78.23	79.75
	70	**85.88**	81.64	81.57	82.20	83.07
	75	**88.71**	84.64	84.27	84.58	85.78
	80	**91.30**	87.45	86.80	86.93	88.29
	85	**93.67**	88.82	89.19	88.92	89.97
5.0	65	**91.21**	86.94	86.73	87.39	88.13
	70	**93.80**	89.86	89.38	89.94	90.80
	75	**95.57**	92.35	91.77	91.82	93.10
	80	**96.91**	93.88	93.81	93.63	94.64
	85	**97.87**	95.19	95.10	94.48	95.95

Tabla B.14: Resultados del problema DMCLP-LocF para la instancia de 2000 nodos de demanda y 150 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		2500				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**71.04**	67.55	67.34	67.14	68.40
	70	**74.94**	70.90	70.07	70.79	71.66
	75	**78.41**	74.34	74.02	74.00	74.85
	80	**81.77**	77.02	76.01	76.63	77.75
	85	**84.17**	79.44	78.62	79.19	80.27
4.5	65	**82.80**	77.61	77.42	78.32	79.14
	70	**86.69**	80.66	80.79	80.95	81.91
	75	**88.87**	83.95	83.61	84.14	84.53
	80	**91.33**	86.79	86.02	85.85	86.95
	85	**93.26**	88.65	87.77	87.88	89.33
5.0	65	**90.91**	87.28	86.63	86.78	87.49
	70	**93.61**	88.97	88.89	89.00	90.17
	75	**95.67**	91.01	91.78	91.60	92.39
	80	**97.39**	93.50	93.45	93.21	94.11
	85	**98.30**	94.76	94.52	94.77	95.69

Tabla B.15: Resultados del problema DMCLP-LocF para la instancia de 2500 nodos de demanda y 200 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es el porcentajes de demanda cubierto con respecto al total de demanda. El mejor resultado en cada caso está marcado en negrita.

		1800				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**312148.50**	297325.00	299477.00	299521.00	303492.50
	70	**327268.00**	314402.00	313043.00	314801.00	319005.00
	75	**341429.50**	329022.50	326820.00	327452.00	334140.00
	80	**355806.00**	340094.50	338665.50	342637.50	346578.00
	85	**370324.50**	354043.00	351201.50	355815.50	358328.00
4.5	65	**363088.50**	345148.00	346479.00	350038.00	354410.00
	70	**377681.00**	361870.50	362766.00	364682.50	368307.00
	75	**391629.50**	374378.00	374198.00	377444.00	380514.00
	80	**404377.00**	388810.50	387430.00	385916.00	390993.50
	85	**414541.50**	395088.00	397319.00	397875.00	401846.00
5.0	65	**405917.00**	392622.00	393093.00	392936.50	396757.00
	70	**416345.00**	401880.00	404632.50	406146.50	407759.50
	75	**426922.00**	412205.50	413534.50	413802.00	417387.00
	80	**434455.50**	418526.50	421148.50	421064.00	424746.00
	85	**440207.00**	426666.00	427023.00	427872.50	431996.00

Tabla B.16: Resultados del problema DMCLP-LocF para la instancia de 1800 nodos de demanda y 100 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

		2000				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**361184.50**	343333.00	344191.00	345439.00	348726.50
	70	**378961.50**	358167.00	359156.50	359918.00	366662.50
	75	**396670.50**	372539.50	377881.00	377580.00	381845.00
	80	**411926.00**	387233.00	388704.00	389895.00	395520.50
	85	**425577.00**	404661.00	401242.50	401488.50	408855.00
4.5	65	**417171.00**	393920.50	394571.00	392988.50	400632.00
	70	**431418.50**	410125.00	409792.50	412958.00	417321.00
	75	**445622.50**	425211.50	423333.50	424896.00	430932.50
	80	**458668.50**	439299.00	436040.50	436687.50	443536.00
	85	**470560.00**	446200.50	448073.50	446688.00	451950.00
5.0	65	**458194.00**	436743.00	435695.00	439022.50	442735.00
	70	**471217.00**	451403.00	449023.50	451813.50	456152.50
	75	**480109.00**	463949.00	461013.50	461272.00	467690.00
	80	**486830.00**	471637.50	471268.50	470366.50	475455.50
	85	**491649.50**	478179.50	477745.00	474647.00	482000.50

Tabla B.17: Resultados del problema DMCLP-LocF para la instancia de 2000 nodos de demanda y 150 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

		2500				
C	Inst	PF	RS-E	RS-H	RS-L	EE
4.0	65	**448355.50**	426323.50	424986.00	423739.00	431671.00
	70	**472947.00**	447448.00	442205.00	446764.00	452276.00
	75	**494871.00**	469155.00	467124.00	466993.50	472412.00
	80	**516080.50**	486074.00	479712.50	483626.50	490668.50
	85	**531180.00**	501376.00	496204.00	499784.50	506602.00
4.5	65	**522563.00**	489825.00	488592.50	494285.50	499460.00
	70	**547128.00**	509044.50	509895.50	510890.00	516957.00
	75	**560890.00**	529826.50	527694.50	531025.50	533495.00
	80	**576370.50**	547732.50	542866.50	541801.50	548769.00
	85	**588575.50**	559459.50	553924.50	554609.50	563758.00
5.0	65	**573730.00**	550825.50	546738.00	547647.50	552135.50
	70	**590777.00**	561505.50	561018.50	561670.00	569051.00
	75	**603792.00**	574360.00	579250.00	578126.00	583110.00
	80	**614615.00**	590071.50	589796.00	588265.00	593925.50
	85	**620375.50**	598033.50	596538.00	598101.50	603933.50

Tabla B.18: Resultados del problema DMCLP-LocF para la instancia de 2500 nodos de demanda y 200 nodos de instalaciones, para cada algoritmo y las diferentes combinaciones de cobertura e instalaciones a ubicar. El resultado que se muestra es la métrica promedio de la evaluación en la Función Objetivo. El mejor resultado en cada caso está marcado en negrita.

Bibliografía

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[2] I. G. Amo, D. A. Pelta, J. R. González, and P. Novoa. An analysis of particle properties on a multi-swarm pso for dynamic optimization problems. In P. Meseguer, L. Mandow, and R. Gasca, editors, *Current Topics in Artificial Intelligence*, volume 5988 of *Lecture Notes in Computer Science*, pages 32–41. Springer, 2010.

[3] P. Angeline. Tracking extrema in dynamic environments. In P. Angeline, R. Reynolds, J. McDonnell, and R. Eberhart, editors, *Evolutionary Programming VI*, volume 1213 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 1997.

[4] M. Ayob and G. Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Placement Machine, INTECH 2003 Thailand*, pages 132–141, 2003.

[5] A. Başar, B. Çatay, and T. Ünlüyurt. A multi-period double coverage approach for locating the emergency medical service stations in istanbul. *Journal of the Operational Research Society*, 62(4):627–637, 2010.

[6] O. Berman and D. Krass. The generalized maximal covering location problem. *Computers & Operations Research*, 29:563–581, 2002.

[7] M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Studies in Computational Intelligence. Springer, 2005.

[8] T. Blackwell. Particle swarm optimization in dynamic environments. In S. Yang, Y.-S. Ong, and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, pages 29–49. Springer, 2007.

[9] T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.

[10] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[11] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Evolutionary Computation (CEC'99)*, volume 3, pages 1875–1882, 1999.

[12] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[13] J. Branke, M. Orbayi, and S. Uyar. The role of representations in dynamic knapsack problems. In F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. Moore, J. Romero, G. Smith, G. Squillero, and H. Takagi, editors, *Applications of Evolutionary Computing*, volume 3907 of *Lecture Notes in Computer Science*, pages 764–775. Springer, 2006.

[14] J. Branke and H. Schmeck. Designing evolutionary algorithms for dynamic optimization problems. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing*, Natural Computing Series, pages 239–262. Springer, 2003.

[15] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[16] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[17] S. Chawathe. Organizing hot-spot police patrol routes. In *Intelligence and Security Informatics, 2007 IEEE*, pages 79–86, May 2007.

[18] R. Church and C. ReVelle. The maximal covering location problem. *Papers of the Regional Science Association*, 32(1):101–118, 1974.

[19] C. C. Coello, G. B. Lamont, and D. A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer Science, 2007.

[20] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2001.

[21] T. G. Crainic. *Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search*, chapter Parallel computation, cooperation, tabu search, pages 283–302. Kluwer Academic Publishers, 2005.

[22] T. G. Crainic and M. Toulouse. Explicit and emergent cooperation schemes for search algorithms. In *Learning and Intelligent Optimization: Second International Conference (LION II)*, volume 5313 of *Lecture Notes in Computer Science*, pages 95–109, 2008.

[23] Y. Crama and M. Schyns. Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research*, 150(3):546 – 571, 2003.

[24] C. Cruz, J. R. González, and D. A. Pelta. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448, 2011.

[25] C. Cruz and D. Pelta. Soft computing and cooperative strategies for optimization. *Applied Soft Computing*, 9(1):30 – 38, 2009.

[26] H. Dam, C. Lokan, and H. Abbass. Evolutionary online data mining: An investigation in a dynamic environment. *Studies in Computational intelligence*, 51:153–178, 2007.

[27] F. O. de França, F. J. Von Zuben, and L. N. de Castro. An artificial immune network for multimodal function optimization on dynamic environments. In *Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 289–296, New York, USA, 2005. ACM.

[28] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions Evolutionary Computation*, 6(2):182–197, 2002.

[29] I. G. del Amo, D. A. Pelta, J. R. González, and A. D. Masegosa. An algorithm comparison for dynamic optimization problems. *Applied Soft Computing*, 12(10):3176 – 3192, 2012.

[30] K. Doerner. *Metaheuristics: Progress in complex systems optimization*, volume 39. Springer Verlag, 2007.

[31] K. Dowold, A. Aderhold, A. Scheidler, and M. Middendorf. Performance evaluation of artificial bee colony optimization and new selection schemes. *Memetic Computing*, 3:149–162, 2011.

[32] S. Droste. Analysis of the (1+1) EA for a dynamically bitwise changing onemax. In *Genetic and Evolutionary Computation Conference (GECCO'03)*, volume 2723 of *Lecture Notes in Computer Science*, pages 909–921. Springer, 2003.

[33] W. Du and B. Li. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, 178(15):3096 – 3109, 2008.

[34] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, Oct 1995.

[35] P. Eeles. Layering strategies. In *Rational Architecture Workshop*, 2001.

[36] A. Elshamli, H. Abdullah, and S. Areibi. Genetic algorithm for dynamic path planning. In *Canadian Conference on Electrical and Computer Engineering*, 2004.

[37] J. Fajardo, A. D. Masegosa, and D. Pelta. Algorithm portfolio based scheme for dynamic optimization problems. *International Journal of Computational Intelligence Systems*, 8(4):667–689, 2015.

[38] J. Fajardo and A. Rosete. Algoritmo multigenerador de soluciones, para la competencia y colaboración de generadores metaheurísticos. *Revista Internacional de Investigación de Operaciones*, 1:57–63, 2011.

[39] W. Feng, T. Brune, L. Chan, M. Chowdhury, C. Kuek, and Y. Li. Benchmarks for testing evolutionary algorithms. Technical report, Center for System and Control, University of Glasgow, 1997.

[40] C. Fernandes, J. Laredo, A. Rosa, and J. Merelo. The sandpile mutation genetic algorithm: an investigation on the working mechanisms of a diversity-oriented and self-organized mutation operator for non-stationary functions. *Applied Intelligence*, 39(2):279–306, 2013.

[41] H. Finner. On a monotonicity problem in step-down multiple test procedures. *Journal of the American Statistical Association*, 88(423):920–923, 1993.

[42] H. Fisher and G. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference, Carnegie Institue of Technology*, 1961.

[43] C. Fonseca and P. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *5th International Conference on Genetic Algorithms*, pages 416–423, San Mateo, CA, 1993.

[44] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[45] M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2006.

[46] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[47] S. García, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[48] P. Garrido and C. Castro. Stable solving of cvrps using hyperheuristics. In *Genetic and Evolutionary Computation Conference (GECCO'09)*, pages 255–262. ACM, 2009.

[49] M. Gendreau, G. Laporte, and F. Seme. A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. *Parallel Computing*, 27(12):1641–1653, 2001. Applications of parallel computing in transportation.

[50] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[51] F. Glover and B. Melián. Búsqueda tabú. *Revista Iberoamericana de Inteligencia Artificial*, 7(19):29–48, 2003.

[52] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.

[53] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[54] U. Gönül, B. Kiraz, S. Etaner-Uyar, and E. Özcan. A framework to hybridize pbil and a hyper-heuristic for dynamic environments. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 358–367. Springer, 2012.

[55] J. González, C. Cruz, I. Amo, and D. Pelta. An adaptive multiagent strategy for solving combinatorial dynamic optimization problems. In D. Pelta, N. Krasnogor, D. Dumitrescu, C. Chira, and R. Lung, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2011)*, volume 387 of *Studies in Computational Intelligence*, pages 41–55. Springer, 2011.

[56] J. R. González, A. D. Masegosa, and I. G. del Amo. A cooperative strategy for solving dynamic optimization problems. *Memetic Computing*, 3:3–14, 2011.

[57] R. Greiner. Probabilistic hill-climbing: Theory and applications. In *Procedings of the bienial Conference-Canadian Society of Computational Studies of Intelligence (CSCSI-92)*, pages 60–67. Morgan Kaufmann Publishers, Inc, 1992.

[58] G. Gunawardane. Dynamic versions of set covering type public facility location problems. *European Journal of Operational Research*, 10(2):190–195, 1982.

[59] M. Guntsch, M. Middendorf, and H. Schmeck. An ant colony optimization approach to dynamic TSP. In *Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 860–867, San Francisco, CA, USA, 2001.

[60] J. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[61] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[62] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.

[63] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[64] O. Karasakal and E. K. Karasakal. A maximal covering location model in the presence of partial coverage. *Computers & Operations Research*, 31:1515–1526, 2004.

[65] J. Karimi, H. Nobahari, and S. Pourtakdoust. A new hybrid approach for dynamic continuous optimization problems. *Applied Soft Computing*, 12(3):1158 – 1167, 2012.

[66] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[67] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, IEEE International Conference*, volume 4, pages 1942–1948, 1995.

[68] B. Kiraz, A. S. Uyar, and E. Özcan. Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769, 2013.

[69] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[70] C. Li and S. Yang. Fast multi-swarm optimization for dynamic optimization problems. In *Fourth International Conference on Natural Computation, IEEE Computer Society*, volume 7, pages 624 – 628, 2008.

[71] C. Li and S. Yang. A comparative study on particle swarm optimization in dynamic environments. In S. Yang and X. Yao, editors, *Evolutionary Computation for Dynamic Optimization Problems*, volume 490 of *Studies in Computational Intelligence*, pages 109–136. Springer, 2013.

[72] C. Li, S. Yang, T. Nguyen, E. Yu, X. Yao, Y. Jin, H. Beyer, and P. Suganthan. Benchmark generator for CEC 2009 competition on dynamic optimization. Technical report, 2008.

[73] C. S. Lim, R. Mamat, and T. Braunl. Impact of ambulance dispatch policies on performance of emergency medical services. *Intelligent Transportation Systems, IEEE Transactions on*, 12(2):624–632, June 2011.

[74] R. I. Lung and D. Dumitrescu. Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing*, 9(1):83–94, 2009.

[75] Y. Mack, T. Goel, W. Shyy, and R. Haftka. Surrogate model-based optimization framework: A case study in aerospace design. *Studies in Computational Intelligence*, 51:323–342, 2007.

[76] A. Masegosa, F. Mascia, D. Pelta, and M. Brunato. Cooperative strategies and reactive search: A hybrid model proposal. In *Learning and Intelligent Optimization*, volume 5851 of *Lecture Notes in Computer Science*, pages 206–220. Springer, 2009.

[77] A. D. Masegosa, D. Pelta, and I. G. del Amo. The role of cardinality and neighborhood sampling strategy in agent-based cooperative strategies for dynamic optimization problems. *Applied Soft Computing*, 14, Part C(0):577 – 593, 2014.

[78] D. C. Mattfeld and C. Bierwirth. An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research*, 155(3):616 – 630, 2004.

[79] M. Mavrovouniotis and S. Yang. A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing*, 15(7):1405–1425, 2011.

[80] Z. Michalewicz, M. Schmidt, M. Michalewicz, and C. Chiriac. Adaptive business intelligence: Three case studies. *Studies in Computational Intelligence*, 51:179–196, 2007.

[81] R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Natural Computing Series. Springer, 2004.

[82] R. Morrison and K. De Jong. A test problem generator for non-stationary environments. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, volume 3, pages 2047–2053, Washington D.C., USA, 1999. IEEE Press.

[83] R. W. Morrison. Performance measurement in dynamic environments. In *Genetic and Evolutionary Computation Conference (GECCO'03)*, pages 99–102, 2003.

[84] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[85] T. Nguyen and X. Yao. Solving dynamic constrained optimisation problems using repair methods. *IEEE Trans. on Evol. Comput.*, 2011.

[86] T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6(0):1 – 24, 2012.

[87] D. Pelta, C. Cruz, and J. R. González. A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems. *International Journal of Intelligent Systems*, 24:844–861, 04/2009 2009.

[88] D. Pelta, A. Sancho-Royo, C. Cruz, and J. L. Verdegay. Using memory and fuzzy rules in a cooperative multi-thread strategy for optimization. *Information Sciences*, 176(13):1849 – 1868, 2006.

[89] F. Peng, K. Tang, G. Chen, and X. Yao. Population-based algorithm portfolios for numerical optimization. *IEEE Transactions on Evolutionary Computation*, 14(5):782–800, 2010.

[90] D. Pisinger. Where are the hard knapsack problems? *Computers and Operations Research*, 32(9):2271–2284, 2005.

[91] F. Quintao, F. Nakamura, and G. Mateus. Evolutionary algorithms for combinatorial problems in the uncertain environment of the wireless sensor networks. *Studies in Computational Intelligence*, 51:197–222, 2007.

[92] A. R. R. Montemanni, L. Gambardella and A. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. In *In Second International Workshop on Freight Transportation and Logistics*, 2003.

[93] G. Raidl and J. Gottlieb. *Evolutionary computation in combinatorial optimization*. Springer, 2005.

[94] G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida, M. J. B. Aguilera, C. Blum, J. M. Moreno-Vega, M. P. Pérez, A. Roli, and M. Sampels, editors, *3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin, 2006.

[95] W. Rand and R. Riolo. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: a case study using the shaky ladder hyperplane-defined functions. In *Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 32–38, New York, USA, 2005. ACM.

[96] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[97] R.-J. E. Reeves, Colin. *Genetic Algorithms: Principles and Perspectives*. Operations Research/Computer Science Interfaces Series. Springer, 2002.

[98] G. Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag Berlin, 1994.

[99] S. Remde, P. Cowling, K. Dahal, and N. Colledge. Exact/heuristic hybrids using rvns and hyperheuristics for workforce scheduling. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2007)*, volume 4446, pages 188–197. Springer-Verlag, 2007.

[100] J. F. Repede and J. J. Bernardo. Developing and validating a decision support system for locating emergency medical vehicles in Louisville, Kentucky. *European Journal of Operational Research*, 75(3):567–581, 1994.

[101] C. ReVelle and H. Eiselt. Location analysis: A synthesis and survey. *European Journal of Operational Research*, 165(1):1 – 19, 2005.

[102] C. ReVelle, M. Scholssberg, and J. Williams. Solving the maximal covering location problem with heuristic concentration. *Computers & Operations Research*, 35:427–435, 2008.

[103] P. Rohlfshagen and X. Yao. The dynamic knapsack problem revisited: A new benchmark problem for dynamic combinatorial optimisation. In M. Giacobini, A. Brabazon, S. Cagnoni, G. Caro, A. EkÃ¡rt, A. Esparcia-AlcÃ¡zar, M. Farooq, A. Fink, and P. Machado, editors, *Applications of Evolutionary Computing*, volume 5484 of *Lecture Notes in Computer Science*, pages 745–754. Springer, 2009.

[104] A. Rosete. *Una solución flexible y eficiente para el trazado de grafos basada en el Escalador de Colinas Estocástico*. PhD thesis, Instituto Superior Politécnico "José Antonio Echeverría", CUJAE: La Habana, 2000.

[105] J. Rumbaugh, I. Jacobson, and G. Booch. *El lenguaje Unificado de Modelado, Manual de Referencia*. Addison Wesley, 2000.

[106] N. Sabar, M. Ayob, R. Qu, and G. Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1):1–11, 2011.

[107] R. Santana, P. Larrañaga, and J. A. Lozano. Adaptive estimation of distribution algorithms. In C. Cotta, M. Sevaux, and K. Sörensen, editors, *Adaptive and Multilevel*

Metaheuristics, volume 136 of *Studies in Computational Intelligence*, pages 177–197. Springer, 2008.

[108] D. A. Schilling. Dynamic location modeling for public-sector facilities: A multicriteria approach. *Decision Sciences*, 11(4):714–724, 1980.

[109] V. Schmid. Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming. *European Journal of Operational Research*, 219(3):611–621, 2012.

[110] V. Schmid and K. F. Doerner. Ambulance location and relocation problems with time-dependent travel times. *European Journal of Operational Research*, 207(3):1293 – 1303, 2010.

[111] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2*. Wiley & Sons, 2000.

[112] L. Schönemann. The impact of population sizes and diversity on the adaptability of evolution strategies in dynamic environments. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1270–1277 Vol.2, June 2004.

[113] L. Schönemann. Evolution strategies in dynamic environments. In S. Yang, Y.-S. Ong, and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, pages 51–77. Springer, 2007.

[114] A. Shalloway and J. R. Trott. *Design Patterns Explained A New Perspective on Object-Oriented Design Second Edition*. Addison Wesley, 2004.

[115] N. Shukla, A. Choudhary, P. Prakash, K. Fernandes, and M. Tiwari. Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance. *International Journal of Production Economics*, 141(1):146 – 166, 2013.

[116] N. Shukla, M. K. Tiwari, and D. Ceglarek. Genetic-algorithms-based algorithm portfolio for inventory routing problem with stochastic demand. *International Journal of Production Research*, 51(1):118–137, 2013.

[117] R. Sri and N. Balaji. Meta-heuristic hybrid dynamic task scheduling in heterogeneous computing environment. In *Computer Communication and Informatics (ICCCI), 2013 International Conference on*, pages 1–6, Jan 2013.

[118] S. A. Stanhope and J. M. Daida. (1+1) genetic algorithm fitness dynamics in a changing environment. In *Proceedings of the 1999 Congress of Evolutionary Computation, CEC 2009*, volume 3, pages 1851–1858, Washington D.C., USA, 1999.

[119] P. N. Suganthan, N. Hansen, J. J. Liang, , K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005.

[120] E. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[121] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, Inc., 2009.

[122] K. Tanaka. Maximum flow-covering location and service start time problem and its application to tokyo metropolitan railway network. *Journal of the Operations Research Society of Japan*, 54(4):237–258, 2011.

[123] H. Terashima, J. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In *Genetic and Evolutionary Computation Conference (GECCO'08)*, Atlanta, Georgia, USA, 2008.

[124] H. Terashima-Marín, C. Zárate, P. Ross, and M. Valenzuela-Rendón. Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem. In *Genetic and Evolutionary Computation Conference (GECCO'07)*, pages 2182–2189. ACM, 2007.

[125] M. Tezuka, M. Munetomo, and K. Akama. Genetic algorithm to optimize fitness function with sampling error and its application to financial optimization problem. *Studies in Computational Intelligence*, 51:417–434, 2007.

[126] R. Tinós and S. Yang. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. *Genetic Programming and Evolvable Machines*, 8(3):255–286, 2007.

[127] H. R. Topcuoglu, A. Ucar, and L. Altin. A hyper-heuristic based framework for dynamic optimization problems. *Applied Soft Computing*, 19(0):236 – 251, 2014.

[128] M. Toulouse, T. G. Crainic, B. Sanso, and K. Thulasiraman. Self-organization in cooperative tabu search algorithms. In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2379–2385. Omnipress, 1998.

[129] A. M. Turky and S. Abdullah. A multi-population harmony search algorithm with external archive for dynamic optimization problems. *Information Sciences*, 272(0):84 – 95, 2014.

[130] A. K. Vatsa. Multi-period facility location problem with an uncertain number of servers. IIMA Working Papers WP2014-02-06, Indian Institute of Management Ahmedabad, Research and Publication Department, 2014.

[131] J. Verdegay, R. R. Yager, and P. P. Bonissone. On heuristics as a fundamental constituent of soft computing. *Fuzzy Sets and Systems*, 159:846–855, 2008.

[132] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, USA, 1998.

[133] H. Wang, D. Wang, and S. Yang. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing*, 13(8-9):763–780, 2009.

[134] K. Weicker. Performance measures for dynamic environments. In *Parallel Problem Solving from Nature, PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 64–73. Springer, 2002.

[135] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, 1945.

[136] D. H. Wolpert and W. G. Macready. No free lunch theorems for search, 1995.

[137] S. R. Yadav, R. R. M. Muddada, M. Tiwari, and R. Shankar. An algorithm portfolio based solution methodology to solve a supply chain optimization problem. *Expert Systems with Applications*, 36(4):8407–8420, 2009.

[138] S. Yang, H. Cheng, and F. Wang. Genetic algorithms with immigrants and memory schemesfor dynamic shortest path routing problems in mobile ad hoc networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(99):52–63, 2010.

[139] S. Yang, Y. Jiang, and T. T. Nguyen. Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, 24(4):451–480, 2013.

[140] S. Yang, Y. Ong, and Y. Jin. *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*. Springer, 2007.

[141] S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11):815–834, 2005.

[142] J. Yaochu and J. Branke. Evolutionary optimization in uncertain environments: a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.

[143] S. Yuen and X. Zhang. On composing an algorithm portfolio. *Memetic Computing*, pages 1–12, 2015.

[144] M. H. F. Zarandia, S. Davaria, and S. A. H. Sisakht. The large-scale dynamic maximal covering location problem. *Mathematical and Computer Modelling*, 57:710–719, 2013.