Universidad de Málaga

Tesis Doctoral

Doctorado en Tecnologías Informáticas

Métodos *anytime* para problemas de optimización combinatoria multiobjetivo con aplicación a Ingeniería del Software

---

Autor:
Miguel Ángel Domínguez Ríos

Directores:
José Francisco Chicano García
Enrique Alba Torres

Enero 2022

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

AUTOR: Miguel Ángel Domínguez Ríos

https://orcid.org/0000-0002-1972-0169

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:
http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode
Cualquier parte de esta obra se puede reproducir sin autorización
pero con el reconocimiento y atribución de los autores.
No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): riuma.uma.es

UNIVERSIDAD DE MÁLAGA

Declaración de autoría

D. Miguel Ángel Domínguez Ríos, estudiante del programa de doctorado **Tecnologías Informáticas**, de la Universidad de Málaga, autor de esta tesis, presentada para la obtención del título de doctor con mención internacional por la Universidad de Málaga, titulada

Métodos anytime *para problemas de optimización combinatoria multiobjetivo con aplicación a Ingeniería del Software*,

realizada bajo la dirección de **D. José Francisco Chicano García** y **D. Enrique Alba Torres** y tutorización de D. Enrique Alba Torres,

DECLARA QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asume, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 8 de enero de 2022

Firmado por DOMINGUEZ RIOS MIGUEL ANGEL - el día 08/01/2022 con un certificado emitido por AC FNMT Usuarios

Fdo.: Miguel Ángel Domínguez Ríos

CHICANO GARCIA JOSE FRANCISCO - Firmado digitalmente por CHICANO GARCIA JOSE FRANCISCO - Fecha: 2022.01.10 16:08:06 +01'00'

Fdo.: José Francisco Chicano García

ALBA TORRES ENRIQUE - Firmado digitalmente por ALBA TORRES ENRIQUE - Fecha: 2022.01.10 15:36:17 +01'00'

Fdo.: Enrique Alba Torres

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Agradecimientos

Y por fin llegó el culmen de mi trayectoria académica. Más de seis años de duro trabajo que me han hecho madurar en el ámbito de la investigación. Un aprendizaje continuo y un largo camino lleno de obstáculos, que han sido necesarios ir superando. Evidentemente, nada de esto habría sido posible sin la ayuda recibida por parte de mis directores de tesis, Francisco Chicano y Enrique Alba.

Compaginar el trabajo laboral con la realización de una tesis doctoral no es una tarea baladí, más aún cuando has superado ya los cuarenta. Por esto, y por toda la ayuda recibida, quiero dar las gracias a todos los que directa o indirectamente me han ayudado a conseguir este logro.

En primer lugar agradezco a Francisco Chicano toda la ayuda que me ha ofrecido y todo lo que de él he aprendido. Su prácticamente total disponibilidad a la hora de resolver problemas, el haberme enseñado a ser paciente, a controlar mis impulsos a la hora de tomar decisiones, su profesionalidad y su sobrada aptitud, son solo algunas de las múltiples características o cualidades que podría mencionar. Él ha sido con quien más contacto directo he tenido, quien me ha levantado el ánimo en mis momentos más difíciles y a quien estaré eternamente agradecido por todo lo que ha hecho de mi.

En segundo lugar agradezco también a Enrique Alba su ayuda en los momentos en los que la he necesitado. Destaco su capacidad de síntesis a la hora de revisar los manuscritos y su talento para detectar errores de estructura o de concepto con un simple vistazo, cuando yo ya había repasado varias veces el documento y creía que todo era correcto. Jamás olvidaré aquellos archivos *pdf* con más de cien comentarios para revisar.

Por supuesto, agradezco enormemente también el apoyo de mi familia. A mis tesoros, mi mujer Eva y mi hijo Miguel Ángel, por su paciencia y por entender mi pasión por la investigación. A mi hermana Ana María y a mi madre Nety, por darme palabras de ánimo en aquellos momentos difíciles en los que la tesis quedaba estancada. A mi padre Paco y a mi hermano José Francisco, quienes creo que en ocasiones magnificaban demasiado mi trabajo.

Además me gustaría mencionar a aquellos amigos que han sabido valorar el esfuerzo que supone ejecutar este proyecto. Me vienen a la mente algunos compañeros: Salvador Domínguez, de la carrera de matemáticas en la Universidad de Málaga; Felipe Rosa, del Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna; Nicolás Guillén, compañero de trabajo en el instituto de educación secundaria; Marcelo de Souza, estudiante de doctorado a quién conocí durante mi estancia en la Universidad de Manchester; algunos de mis amigos que conservo desde la juventud, como Antonio Chamizo, Francisco Valenzuela o Juanjo Arjona; los miembros del grupo de investigación NEO, así como el resto de familiares y amigos que en algún momento han mostrado su interés por la evolución de mi tesis.

Por último, me gustaría dedicar unas palabras de agradecimiento a mi *golden retriever*, Luna, compañera fiel e inseparable, quien más horas estuvo a mi lado durante mi trabajo, y a quien siempre tendré en mi corazón.

¡Gracias a todos!

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Índice general

Declaración de autoría III

Agradecimientos VII

1. Introducción 1
1.1. Contexto . . . 1
1.2. Objetivos y metodología . . . 3
1.3. Contribuciones . . . 4
1.4. Resumen del estado de la cuestión . . . 4
1.5. Organización de esta tesis doctoral . . . 5

I Fundamentos 7

2. Introducción a la optimización combinatoria 9
2.1. Complejidad algorítmica . . . 9
2.2. Optimización combinatoria . . . 11
2.3. Programación lineal . . . 16
2.4. Optimización multiobjetivo . . . 21

3. Técnicas de resolución de problemas multiobjetivo 27
3.1. Métodos exactos . . . 27
3.1.1. Técnicas de escalarización . . . 28
3.1.2. Formulación genérica que engloba las técnicas de escalarización . . . 33
3.1.3. Algoritmo genérico que calcula el frente de Pareto completo . . . 34
3.1.4. Uso de CPLEX como resolutor de problemas multiobjetivo . . . 36
3.2. Métodos aproximados. Metaheurísticas . . . 37
3.3. Metaheurísticas híbridas . . . 40
3.4. Evaluación de las soluciones obtenidas . . . 41
3.4.1. Indicadores de calidad . . . 42
3.4.2. Métodos de comparación . . . 45
3.4.3. Algoritmos *anytime* . . . 48
3.4.4. Contrastes de hipótesis no paramétricos . . . 49

II Propuestas algorítmicas 53

4. Algoritmo *TPA* 55
4.1. Algoritmo genérico para problemas MOCO . . . 55
4.2. Revisión de la literatura . . . 60
4.3. Algoritmo *TPA* . . . 63
4.3.1. Elección del programa matemático . . . 63
4.3.2. Contribuciones . . . 64
4.3.3. Propuesta algorítmica . . . 68
4.4. Resultados computacionales . . . 70
4.4.1. Instancias, métricas y parámetros de CPLEX . . . 70
4.4.2. Análisis de los resultados computacionales . . . 72
4.4.3. Validación estadística . . . 76
4.5. Conclusiones y trabajo futuro . . . 78

UNIVERSIDAD DE MÁLAGA

5. Algoritmos híbridos en optimización combinatoria multiobjetivo **79**
5.1. Algoritmo *MOFeLS* . . . 79
5.1.1. Introducción . . . 79
5.1.2. Propuesta algorítmica . . . 80
5.1.3. Resultados computacionales . . . 84
5.2. Algoritmo *HBOXES* . . . 88
5.2.1. Descripción de la idea y justificación . . . 88
5.2.2. Propuesta algorítmica . . . 90
5.2.3. Resultados computacionales . . . 90
5.3. Conclusiones y trabajo futuro . . . 96

III Aplicaciones **99**

6. Problema de Partición de Módulos Software **101**
6.1. Descripción del problema . . . 101
6.2. Resolución del problema *SMC* mediante ramificación y poda . . . 103
6.3. Resolución del problema *SMC* mediante programación matemática . . . 105
6.4. Resultados computacionales . . . 112
6.5. Investigación ampliada . . . 114
6.6. Conclusiones . . . 116

7. Problema de Selección de Requisitos **119**
7.1. Descripción del problema . . . 119
7.2. *NRP* monoobjetivo . . . 121
7.2.1. Algoritmo híbrido usando CMSA . . . 122
7.2.2. Resultados computacionales . . . 124
7.3. *NRP* biobjetivo . . . 128
7.3.1. Algoritmos clásicos que resuelven el *NRP* biobjetivo . . . 128
7.3.2. Algoritmos *anytime* eficientes que resuelven el *NRP* biobjetivo . . . 129
7.3.3. Resultados computacionales . . . 134
7.4. Conclusiones . . . 141

IV Conclusiones y trabajo futuro **143**

V Apéndices **149**

A. Relación de publicaciones que sustentan la tesis **151**
A.1. Publicaciones en revistas . . . 151
A.2. Publicaciones en congresos nacionales e internacionales . . . 151

B. Summary of this thesis in English **153**
B.1. Organization of the thesis . . . 153
B.2. Introduction to combinatorial optimization . . . 155
B.3. Techniques to solve multiobjective combinatorial optimization problems . . . 155
B.4. *TPA* algorithm . . . 157
B.5. Hybrid algorithms based on multiobjective combinatorial optimization . . . 158
B.6. Software Module Clustering problem . . . 159
B.7. Next Release Problem . . . 160
B.8. Conclusions and future work . . . 161

Símbolos, siglas y acrónimos **165**

Bibliografía **165**

Índice de figuras **177**

Índice de tablas **179**

UNIVERSIDAD DE MÁLAGA

A todo lo que amo, y en particular, a mi familia y a la ciencia,
por todo lo bueno que recibo de ellas

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Capítulo 1

Introducción

1.1. Contexto

Los problemas de optimización combinatoria [29] se resuelven cuando encontramos una solución, de entre un conjunto finito de ellas, que minimice o maximice una determinada función objetivo, que asigna a cada una de las soluciones un número real. Dos ejemplos clásicos son el problema del camino más corto en un grafo [73] y el problema del viajante de comercio (del inglés *Travelling Salesperson Problem*, TSP) [4]. Los problemas de optimización combinatoria se encuentran presentes en casi todos los ámbitos de la sociedad. Su resolución es importante cuando una entidad u organización desea minimizar los costes y/o aumentar los beneficios. Por ejemplo, las empresas de reparto de mercancías tratan de minimizar el coste de transporte empacando de forma adecuada la mercancía en los camiones y eligiendo el orden apropiado de entrega para que el consumo de combustible o el tiempo requerido para hacer el reparto sea mínimo [125]. Pero también encontramos ejemplos de estos problemas en nuestra vida cotidiana. Si tenemos prisa para llegar a un destino concreto, intentaremos dirigirnos por el camino más rápido de entre todos los que nos conectan con nuestro destino, y para ello podríamos resolver una instancia del problema del *camino más corto*.

Dada la importancia que conlleva resolver problemas de optimización combinatoria, no es de extrañar que existan multitud de congresos, revistas y libros dedicados a esta cuestión. Algunos ejemplos son: *International Conference on Combinatorial Optimization and Applications*[1] (COCOA), congreso que se celebra anualmente; la revista *Journal of Combinatorial Optimization*,[2] de Springer; o la enciclopédica obra *Handbook on Combinatorial Optimization* [144], editada por Panos M. Pardalos, Ding-Zhu Du y Ronald L. Graham.

Muchos de los problemas de optimización a los que debe enfrentarse la sociedad no tienen una única función objetivo para optimizar, sino varias. Dichos problemas se denominan *multiobjetivo* [58]. Cuando dos o más objetivos son contrapuestos no existe, en general, una única solución que los minimice o los maximice simultáneamente a todos. En este contexto es posible utilizar el concepto de *dominancia de Pareto*, para establecer un orden parcial entre las soluciones. Se dice que una solución x domina a otra solución y cuando el vector objetivo asociado a x es mejor que el vector objetivo asociado a y en alguna componente y no es peor en las demás. Desde el punto de vista del usuario (o decisor) que quiere resolver el problema multiobjetivo, la solución y es descartable. En estos problemas lo interesante es conocer un conjunto de soluciones que no puede mejorarse, es decir, un conjunto de soluciones no dominadas, también llamado conjunto óptimo de Pareto. La imagen en el espacio objetivo de este conjunto se conoce como *frente de Pareto* y resulta ser una herramienta muy importante para que el decisor pueda elegir la solución que más le interesa de entre un conjunto de soluciones, en el que todas ellas podrían considerarse óptimas. Esta decisión final dependerá de circunstancias que rodean al decisor, incluyendo aspectos personales, y que no aparecen reflejados en la formulación matemática del problema.

Algunos problemas de optimización combinatoria pueden resolverse hasta la optimalidad con un algoritmo apropiado utilizando un tiempo polinomial que depende del tamaño del problema. Tal es el caso del problema del camino más corto, que puede resolverse con el algoritmo de Dijkstra [46] en un tiempo $O(A \log N)$, donde A y N son el número de aristas y vértices en

[1] `http://www.wikicfp.com/cfp/program?id=522&f=Conference`
[2] `http://www.springer.com/mathematics/journal/10878`

UNIVERSIDAD DE MÁLAGA

grafo[3], respectivamente. Sin embargo, la resolución de un problema de optimización combinatoria, de manera general, es un problema NP-difícil o $\mathcal{NP}$-*Hard* [74]. Hasta el momento no se conoce ningún algoritmo que pueda resolver los problemas NP-difíciles en tiempo polinomial y la gran mayoría de los expertos en la cuestión opinan que podría no existir tal algoritmo[4]. Si esto es cierto para los problemas con un solo objetivo, se cumple igualmente, y con más razón, para problemas multiobjetivo, ya que el descubrimiento de cada solución no dominada puede ser un problema NP-difícil en sí y, además, el número de soluciones en el frente de Pareto puede crecer de forma exponencial con el tamaño del problema [58].

Los algoritmos usados para resolver los problemas de optimización pueden ser clasificados en dos categorías principales: *exactos* y *aproximados*. Los algoritmos exactos, como los de ramificación y poda, encuentran soluciones óptimas a los problemas, mientras que los aproximados no aseguran encontrar dichas soluciones. Dentro de los aproximados podemos distinguir las heurísticas *ad hoc*, que son algoritmos específicamente diseñados para problemas concretos, y las metaheurísticas [14], que están diseñadas para trabajar sobre cualquier problema de optimización. Un ejemplo de heurística *ad hoc* es el algoritmo Lin-Kernighan [117] para el TSP, mientras que un ejemplo de metaheurística es un *algoritmo genético* [90].

La principal ventaja de los algoritmos exactos es que garantizan encontrar el óptimo global de cualquier problema. Por tanto, constituyen la mejor opción cuando el tiempo requerido para resolver el problema es razonablemente corto. El progreso en algoritmos exactos en los últimos años ha dado lugar a algunos ejemplos destacados de algoritmos que resultan rápidos incluso cuando las instancias que deben resolver son grandes. Por ejemplo, el algoritmo *Pulse*, propuesto por Duque et al. para resolver el problema biobjetivo del camino más corto [54], es capaz de encontrar un conjunto óptimo de Pareto en pocos segundos para grafos con millones de vértices y ejes, que bien permiten modelar los mapas de carreteras de las ciudades más grandes del mundo.

Sin embargo, los algoritmos exactos tienen el inconveniente de que el tiempo requerido para resolver las instancias de un problema puede crecer exponencialmente en el tamaño del mismo. En cambio, los algoritmos aproximados son normalmente bastante rápidos, pero no pueden garantizar que la solución encontrada sea óptima y, en muchos casos, la calidad de las mismas está lejos de ser óptima. Un ejemplo de este conflicto entre tiempo de ejecución y optimalidad lo podemos encontrar en las competiciones de SAT y MAX-SAT[5], donde los algoritmos completos resultan ser más lentos que los algoritmos incompletos, aunque ambos encuentran soluciones de la misma calidad en muchos casos.

Dadas las ventajas e inconvenientes de los algoritmos exactos y aproximados, parece lógico que una combinación acertada de ambos podría dar lugar a un algoritmo aproximado con soluciones óptimas (o casi) o a un algoritmo exacto más rápido. En cualquier caso, la combinación de ambos tipos de algoritmos puede resultar beneficiosa. De hecho, en la literatura científica abundan los ejemplos. En [129], Maslov et al. utilizan una búsqueda local iterada para encontrar una solución aproximada de calidad para el problema del *máximo clique de un grafo*, que se usa posteriormente en un algoritmo de ramificación y poda para incrementar el número de podas. El resultado es un algoritmo exacto más rápido. El resolutor Concorde[6], utilizado para resolver el problema TSP, combina técnicas de búsqueda local como el operador de Lin-Kernighan-Helsaum [88] junto con programación lineal entera para obtener un algoritmo exacto que, hoy en día, forma parte del estado del arte en el TSP. En el otro sentido, el algoritmo CMSA de Blum et al. [12] utiliza una heurística a alto nivel, en la que varios componentes de la solución son fijados antes de resolver un problema de programación lineal entera que obtiene los valores óptimos para el resto de componentes (no fijados).

En los problemas con un solo objetivo, una solución aproximada es aquella que, no siendo óptima, se encuentra cerca, en calidad, de una que sí lo es. En el caso multiobjetivo, un frente aproximado puede ser uno formado por soluciones posiblemente dominadas pero cercanas a las soluciones no dominadas del problema. Pero también podemos hablar de frente aproximado cuando nos referimos a un conjunto de soluciones no dominadas que son un subconjunto propio del frente de Pareto. En este caso, es posible aplicar técnicas exactas para encontrar cada una de las soluciones del frente y usar una estrategia aproximada global de búsqueda

[3] Algunas implementaciones del algoritmo de Dijkstra pueden hacerlo en $O(N \log N + A)$.
[4] Véase la página `http://www.win.tue.nl/~gwoegi/P-versus-NP.htm` sobre la cuestión P *vs* NP.
[5] `https://maxsat-evaluations.github.io`.
[6] `http://www.math.uwaterloo.ca/tsp/concorde/index.html`

UNIVERSIDAD DE MÁLAGA

para encontrar soluciones dispersas en el espacio objetivo que aseguren cierta calidad del frente de Pareto en cualquier momento de la búsqueda. Estos algoritmos son denominados *anytime* y a esta categoría pertenecen la mayoría de los algoritmos desarrollados en esta tesis doctoral.

1.2. Objetivos y metodología

Dentro del contexto expuesto anteriormente, esta tesis doctoral tiene como **objetivo** principal estudiar y proponer nuevos algoritmos que resuelvan problemas de optimización combinatoria multiobjetivo. En particular, se estudiarán problemas de Ingeniería del Software [85], incluso en su versión monoobjetivo. También se utilizarán para los experimentos computacionales *benchmarks* de problemas pertenecientes a problemas de asignación, problemas de tipo mochila, de mochila multidimensional, de programación lineal entera o de recubrimiento mínimo, todos ellos en su versión multiobjetivo. Los algoritmos propuestos pueden ser exactos o aproximados. Aunque nos centramos en algoritmos multiobjetivo, dejamos también abierta la posibilidad de que en algunos problemas o aplicaciones, optemos finalmente por un diseño monoobjetivo, si dado el caso, esta formulación funciona mejor que la formulación multiobjetivo.

Este objetivo principal se puede descomponer en los siguientes subobjetivos:

- Proponer algoritmos exactos o aproximados para resolver los problemas identificados.
- Utilizar la técnica de hibridación entre algoritmos exactos y aproximados para obtener un rendimiento mejor en los resultados.
- Comparar los algoritmos propuestos con los algoritmos del estado de la cuestión y determinar las condiciones en las que el algoritmo propuesto resulta ser mejor que los existentes.
- Analizar los problemas y obtener resultados teóricos que permitan mejorar los algoritmos existentes.

Para poder abordar con éxito los subobjetivos anteriormente enumerados, proponemos una **metodología** basada en las distintas fases del método científico.

En la fase de *observación*, la metodología utilizada comprende las siguientes tareas:

- Estudio de las técnicas de optimización exactas y comprensión del funcionamiento de las heurísticas existentes para la optimización combinatoria multiobjetivo.
- Identificación de problemas de optimización multiobjetivo en el dominio de la Ingeniería del Software.
- Estudio en profundidad de los problemas identificados hasta conocer y comprender los mejores algoritmos existentes para dichos problemas.

En la fase de *inducción* podemos establecer el siguiente item:

- Análisis de los algoritmos de resolución existentes y de los problemas para deducir resultados teóricos que permitan reducir el tiempo de cómputo empleado para alcanzar una solución, en los problemas de optimización identificados.

El siguiente paso del método científico comprende el establecimiento de la *hipótesis*:

- Tomamos como hipótesis de partida la idea de que la combinación de algoritmos exactos y heurísticos para problemas concretos mejoran el rendimiento de los algoritmos exactos y de los algoritmos heurísticos utilizados de forma aislada.

No queremos decir con esto que cualquier combinación de un algoritmo heurístico y uno exacto para un problema dé lugar a un algoritmo mejor que aquellos que lo componen. Por otro lado, tampoco esperamos que la combinación de algoritmos exactos y heurísticos mejore en todos los casos los algoritmos existentes para resolver cualquier problema de optimización

UNIVERSIDAD DE MÁLAGA

combinatoria, o clases muy generales de problemas (como la programación lineal entera), ya que el teorema *No Free Lunch* [182] afirma que todos los algoritmos tienen el mismo rendimiento promedio cuando se consideran todos los problemas de optimización.

A continuación pasamos a la fase de *experimentación*, que contendrá las siguientes actividades:

- Implementar los algoritmos propuestos poniendo especial cuidado para conseguir la mayor eficiencia posible en su ejecución.
- Comparar los algoritmos propuestos con los algoritmos existentes en el estado de la cuestión del problema.

El último aspecto metodológico a detallar comprende la *tesis*, que puede concentrarse en:

- Analizar los resultados obtenidos y delimitar las condiciones en las cuales los algoritmos propuestos son mejores que los existentes.
- Publicar los resultados como artículos científicos en conferencias y revistas internacionales de alto impacto.
- Publicar los software desarrollados en repositorios de código abierto (GitHub), para su reproducción y validación por parte de la comunidad científica.

1.3. Contribuciones

Las contribuciones de esta tesis doctoral se pueden concretar en las siguientes:

- Diseño de un algoritmo exacto, denominado *TPA*, que resuelve cualquier problema de optimización combinatoria multiobjetivo, y cuyas soluciones obtenidas están bien dispersas por el espacio objetivo en cualquier momento de la ejecución, es decir, es *anytime*.
- Diseño de un algoritmo híbrido *anytime*, denominado *MOFeLS*, que usa un resolutor ILP y una búsqueda local, y que resuelve de manera aproximada cualquier problema de optimización multiobjetivo con variables binarias y funciones objetivo y restricciones lineales.
- Implementación de un algoritmo *anytime* aproximado, llamado *HBOXES*, que en función de los parámetros de entrada puede convertirse en exacto, y que resuelve cualquier problema de optimización combinatoria multiobjetivo.
- Resolución del problema de Partición de Módulos Software mediante un algoritmo de ramificación y poda.
- Modelado y resolución del problema de Partición de Módulos Software mediante programación matemática.
- Modelado de dos algoritmos híbridos, llamados *cmsa_r* y *cmsa_s*, que resuelven de manera aproximada el problema de Selección de Requisitos en su versión monoobjetivo.
- Formulación de tres nuevos algoritmos *anytime* exactos, llamados *AnyHybrid*, *MixHT* y *MixSHT*, que resuelven el problema de Selección de Requisitos en su versión biobjetivo.

1.4. Resumen del estado de la cuestión

Por comodidad para el lector, se desplaza el estado de la cuestión de cada tópico a los correspondientes capítulos, para poder introducir previamente la notación y lenguaje apropiados, y permitir una mejor compresión de dicho estado de la cuestión. Concretamente, esta revisión bibliográfica se encuentra en las Secciones 4.2, 5.1.1, 6.1, 7.1, y 7.2.1.

UNIVERSIDAD DE MÁLAGA

1.5. Organización de esta tesis doctoral

Salvando el presente capítulo introductorio, esta tesis doctoral se divide en cinco partes bien diferenciadas. La primera de ellas se dedica a los fundamentos teóricos, donde en los dos capítulos que contiene, se introduce formalmente el concepto de optimización combinatoria, y se establecen las distintas técnicas que resuelven los problemas combinatorios multiobjetivo, incluyendo la optimización monoobjetivo como caso particular. La segunda parte establece las propuestas algorítmicas, que constituyen las contribuciones aportadas en este campo. Se divide también en dos capítulos, donde se formula, desarrolla y experimenta con un total de tres algoritmos que resuelven problemas combinatorios. El primero de ellos es un algoritmo exacto que resuelve cualquier problema combinatorio multiobjetivo, en general. El segundo es un algoritmo híbrido restringido a problemas binarios multiobjetivo. El tercero es un algoritmo híbrido, si bien en función de los parámetros de entrada utilizados, podría considerarse también como algoritmo exacto. La tercera parte muestra dos nuevos capítulos dedicados a aplicaciones concretas para problemas de Ingeniería del Software. Aunque esta tesis trata sobre optimización multiobjetivo, se ha constatado que en algunos casos la formulación monoobjetivo del problema ha proporcionado resultados de mejor calidad. Así, se establecen algoritmos, tanto exactos como aproximados, que resuelven el problema de Partición de Módulos Software y el de Selección de Requisitos. En la cuarta parte se reflejan las conclusiones de esta tesis, sus limitaciones, así como las líneas de investigación futuras. La última parte contiene dos apéndices, uno con la relación de publicaciones que sustenta la tesis, incluyendo publicaciones en revistas, congresos internacionales y nacionales; y la segunda parte, que contiene un resumen de esta tesis en inglés.

A continuación mostramos un detalle algo más pormenorizado de cada una de las partes que contiene esta tesis doctoral.

- **Parte I. Fundamentos**
  - El Capítulo 2 introduce los conceptos fundamentales utilizados en esta tesis, empezando por la definición de complejidad algorítmica, definiendo las clases de problemas, así como el concepto de optimización combinatoria. Tras una clasificación de los modelos de programación matemática, se definen algunos problemas combinatorios clásicos. Se dedica una sección a la explicación de la técnica de ramificación y poda para resolver problemas lineales enteros, y se emplea la última parte a la definición de optimización combinatoria multiobjetivo, mostrando también algunos ejemplos clásicos.
  - El Capítulo 3 muestra las técnicas de escalarización más usadas a la hora de resolver problemas combinatorios multiobjetivo, así como un algoritmo genérico que resuelve dichos problemas, y que sirve como esqueleto o referencia de algunos algoritmos que serán expuestos en sucesivos capítulos. Se citan someramente algunas de las metaheurísticas más conocidas y utilizadas, así como algunas metaheurísticas híbridas existentes en la literatura. Una nueva sección define los indicadores de calidad que serán utilizados en los experimentos computacionales asociados a los algoritmos. El capítulo finaliza con la definición de algunos test no paramétricos, que serán utilizados para validar los experimentos computacionales.
- **Parte II. Propuestas algorítmicas**
  - El Capítulo 4 propone un algoritmo exacto que resuelve cualquier problema combinatorio multiobjetivo. Tras una revisión de la literatura en este campo, se establece dicha propuesta, junto con los resultados computacionales obtenidos, así como su validación estadística y conclusiones.
  - El Capítulo 5 establece dos algoritmos. El primero de ellos es un algoritmo aproximado que resuelve problemas multiobjetivo lineales con variables binarias. El segundo es un algoritmo híbrido, válido para resolver cualquier problema combinatorio multiobjetivo. Ambos algoritmos conllevan sus propios experimentos computacionales, a la vez que son comparados entre sí, y comparados con el algoritmo descrito en el Capítulo 4.
- **Parte III. Aplicaciones**

UNIVERSIDAD DE MÁLAGA

- El Capítulo 6 describe el problema de Partición de Módulos Software, y plantea su resolución de dos maneras: mediante ramificación y poda, y mediante la resolución de un problema de programación lineal mixto (con variables continuas y enteras). Este problema, resuelto en su versión monoobjetivo, ha sido también estudiado como problema con más de un objetivo, entre otras líneas de investigación que se muestran al final del capítulo.
- El Capítulo 7 describe otro problema del campo de la Ingeniería del Software, conocido como Problema de Selección de Requisitos. Se abordan dos versiones de este problema: monoobjetivo y biobjetivo. En la versión monoobjetivo se proponen dos algoritmos híbridos para su resolución. En la versión biobjetivo se proponen cinco algoritmos que lo resuelven de manera exacta. Algunos de ellos no son más que aplicaciones a este problema de métodos ya existentes, y otros resultan nuevas contribuciones en este campo. En las dos versiones descritas se muestran los correspondientes resultados computacionales.

- **Parte IV. Conclusiones y trabajo futuro**
  Se establecen las conclusiones, algunas limitaciones y las líneas abiertas de investigación futura para cada uno de los problemas tratados.

- **Parte V. Apéndices**

  - El Apéndice A muestra la relación de publicaciones que sustenta la tesis, incluyendo publicaciones en revistas, en congresos internacionales y en congresos nacionales.
  - El Apéndice B contiene un resumen de la tesis doctoral en inglés.

UNIVERSIDAD DE MÁLAGA

Parte I

Fundamentos

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Capítulo 2

Introducción a la optimización combinatoria

El presente capítulo define los conceptos básicos de la optimización combinatoria, así como de la optimización combinatoria multiobjetivo. Durante el mismo, se describen además todas las definiciones de los conceptos más importantes en este campo de investigación. Contiene cuatro partes completamente estructuradas. En una primera parte se clasifican los problemas de optimización en función de su complejidad, permitiendo así catalogar cómo de difíciles podría resultar resolverlos. El segundo apartado clasifica los problemas de optimización y define los modelos combinatorios, estableciendo ejemplos y describiendo algunos problemas clásicos. La tercera parte está dedicada a la programación lineal y a la programación lineal entera, introduciendo sus nociones básicas, aunque sin entrar en detalle en los algoritmos que resuelven estos problemas. En el último apartado se definen la optimización multiobjetivo y la optimización combinatoria multiobjetivo como caso particular, así como todos los conceptos que serán usados en el resto de la tesis.

2.1. Complejidad algorítmica

A modo introductorio, y sin ánimo de profundizar en la *Teoría de la complejidad* [41], podríamos definir la *complejidad algorítmica* como aquel concepto que estudia cómo crece el coste computacional a la hora de resolver un determinado problema en relación a lo que crece el tamaño de dicho problema. Este coste computacional se suele medir en tiempo de ejecución o en memoria consumida.

Para cualquier conjunto finito de símbolos, Σ (que llamamos *alfabeto*), decimos que Σ^* es un *lenguaje* cuando Σ^* es el conjunto de todas las cadenas finitas de símbolos en Σ. Un problema de decisión se puede formalizar como el problema de decidir si una cadena pertenece a un lenguaje. Por ejemplo, supongamos que nos dan un número natural y queremos determinar si es primo o no (dicho de otra forma, si el número pertenece al conjunto de números primos). En ocasiones, este término se simplifica diciendo que un problema de decisión es aquel aquel cuya respuesta es *sí* o *no*. La mayoría de los problemas pueden reducirse a uno de decisión.

Un *algoritmo* es un conjunto específico de instrucciones para llevar a cabo un procedimiento o resolver un problema, generalmente con el requisito de que el procedimiento finalice en algún momento [176]. Existe una relación biunívoca entre los problemas de decisión y los lenguajes formales [99]. Si existe un algoritmo que resuelve un cierto problema de decisión decimos que el problema es *decidible*. En caso contrario decimos que es *indecidible* [176]. El análisis de algoritmos se dedica a estudiar qué recursos necesita un algoritmo para ejecutarse. Podemos estimar el consumo de recursos para un algoritmo dado, lo que nos permite comparar también la eficiencia entre distintos algoritmos. La unidad de medida para estimar el rendimiento de un algoritmo es la *operación básica*, que no depende de valores particulares de los operandos (por ejemplo: sumar, comparar, etc). Generalmente, el factor más importante a considerar es el tamaño de los datos de entrada, n[1]. El tiempo de ejecución de un algoritmo, $T(n)$, se expresa en función de este parámetro de entrada. Calcular la expresión analítica de la función $T(n)$ podría resultar muy complicado o incluso imposible, por lo que se recurre al uso del análisis matemático para obtener tanto cotas inferiores como superiores de dicha función.

[1] Cantidad de datos de entrada necesaria para describir la instancia, [74].

UNIVERSIDAD DE MÁLAGA

Lo mismo podemos decir en relación al espacio en memoria, otro recurso necesario a tener en cuenta, si bien se considera menos importante que el tiempo de ejecución, debido a que los actuales recursos tecnológicos nos permiten obtener gran capacidad de almacenamiento.

En la Tabla 2.1 se muestran ejemplos de algunos problemas y una cota superior del tiempo de ejecución para un algoritmo que lo soluciona. Podemos considerar que la complejidad de un problema es la complejidad del mejor algoritmo que lo resuelve.

Tabla 2.1: Ejemplos de cota superior del tiempo computacional empleado en la resolución de ciertos problemas.

Cota	Tipo cota	Ejemplo
$O(1)$	constante	Acceder a un elemento concreto en un vector
$O(\log n)$	logarítmico	Acceder a un elemento concreto en un árbol binario de n nodos
$O(n)$	lineal	Obtener el mayor elemento en un vector de n elementos
$O(n \cdot \log n)$	cuasi-lineal	Ordenar los elementos de un vector de n elementos usando el algoritmo *heapsort*
$O(n^2)$	polinomial	Ordenar n elementos de un vector usando el algoritmo de la burbuja
$O(2^n)$	exponencial	Calcular todos los subconjuntos de un conjunto de n elementos

Es evidente que cuanto menor sea la cota superior del tiempo de ejecución obtenida por un algoritmo que resuelve un problema, mejor será considerado dicho algoritmo. De manera informal podríamos decir que los problemas *tratables* son aquellos que pueden resolverse en tiempo razonable. Por otro lado, se dice que son *intratables* cuando son tan duros que posiblemente no exista algoritmo que lo resuelva en tiempo polinomial [74]. La justificación de estos adjetivos puede deducirse observando la Tabla 2.2, donde se muestra cómo la función exponencial crece mucho más rápido que un polinomio.

Tabla 2.2: Comparativa de crecimiento de la función potencial y la función exponencial.

n	n^2	n^5	2^n	3^n
5	25	3125	32	243
10	1000	10000	1024	59049
20	400	$3.2 \cdot 10^6$	$1.04 \cdot 10^6$	$3.49 \cdot 10^9$
40	1600	$1.02 \cdot 10^8$	$1.1 \cdot 10^{12}$	$1.22 \cdot 10^{19}$

Se definen a continuación cuatro clases de problemas para, en cierta manera, poder clasificarlos en función de su complejidad algorítmica.

Definición 2.1. Decimos que un problema de decisión está en la clase $\mathcal{P}$ cuando existe un algoritmo que lo resuelve en tiempo polinomial.

Podríamos decir que la clase $\mathcal{P}$ contiene a todos aquellos problemas 'fáciles' de resolver. Lo ideal sería demostrar que todos los problemas de decisión se encuentran en esta clase, es decir, confirmar que siempre existirá un algoritmo con complejidad polinomial que lo resuelva.

Definición 2.2. Decimos que un problema de decisión está en la clase $\mathcal{NP}$ cuando existe un algoritmo que determina en tiempo polinomial si cierta configuración es una solución de dicho problema.

En 1937, Alan Turing definió un modelo matemático computacional, llamado *máquina de Turing*, que demostraba la existencia de problemas indecidibles [170], como el caso del problema de *la parada*. En este trabajo se dio respuesta al problema de *Entscheidungs*, planteado por David Hilbert en 1928: *encontrar un algoritmo que determine, dado un conjunto de axiomas y una proposición, si dicha proposición puede ser probada a partir de esos axiomas.* Finalmente, Turing demostró que esto no era posible, en general.

La máquina de Turing tiene al menos dos versiones: la versión determinista y la no determinista. Se puede establecer una relación entre cada una de las máquinas y las dos definiciones

UNIVERSIDAD DE MÁLAGA

anteriores de manera que diríamos que un problema está en la clase $\mathcal{P}$ si puede ser resuelto en tiempo polinomial en una máquina de Turing determinista, y está en la clase $\mathcal{NP}$ si puede ser resuelto en tiempo polinomial en una máquina de Turing no determinista [74].

Es fácil observar que cualquier problema que esté en la clase $\mathcal{P}$ está también en la clase $\mathcal{NP}$. La gran cuestión, todavía no resuelta en la fecha de escritura de esta tesis, es determinar si también ocurre que cualquier problema en $\mathcal{NP}$ está en $\mathcal{P}$. De ser así, ambas clases serían iguales y estaría resuelto uno de los problemas matemáticos más importantes del milenio [17].

En la búsqueda de métodos para probar o refutar la conjetura $\mathcal{P} \neq \mathcal{NP}$, fueron descubiertas nuevas relaciones entre problemas, estableciéndose una nueva clase llamada *NP-Completo*, o también $\mathcal{NP}$-*Complete*. La idea es considerar una serie de problemas de referencia, dentro de la clase $\mathcal{NP}$, de manera que cualquier problema en $\mathcal{NP}$ pueda reducirse a uno de esta clase de referencia. A modo de ejemplo, supongamos que disponemos de dos problemas. El primero, de ordenación, consistente en una secuencia de n enteros como entrada, y cuya salida es otra secuencia de n enteros, permutación de los anteriores, de manera que aparezcan ordenados, por ejemplo, de menor a mayor. Por otro lado disponemos de un algoritmo de emparejamiento, cuya entrada consiste en dos secuencias de n enteros cada una, sean X e Y, y cuya salida consiste en emparejar el menor valor de X con el menor valor de Y, el segundo menor valor de X con el segundo menor valor de Y, y así sucesivamente. Una forma de resolver el problema de emparejamiento podría consistir en aplicar el algoritmo de ordenación para las dos secuencias por separado, y después emparejar los elementos en base a su posición en los dos *arrays*. De esta forma, diríamos que el problema de emparejamiento se *reduce* al problema de ordenación.

Definición 2.3. Un problema X pertenece a la clase $\mathcal{NP}$-*Complete* si verifica que $X \in \mathcal{NP}$ y además cualquier otro problema de la clase $\mathcal{NP}$ puede ser reducido a X en tiempo polinomial.

Analizando esta nueva definición, podemos concluir que si se encuentra algún algoritmo polinomial para un problema de la clase $\mathcal{NP}$-*Complete*, se podrían resolver todos los de dicha clase mediante reducciones. Los fundamentos de la teoría de la *NP*-completitud se presentaron en 1971 por Stephen Cook [28]. Este autor probó que cualquier problema $\mathcal{NP}$ podía ser reducido en tiempo polinomial al problema de satisfacibilidad de expresiones lógicas (SAT [2]). En este sentido el problema SAT es considerado el primer problema $\mathcal{NP}$-*Complete*, que es la clase de los problemas más duros que hay en $\mathcal{NP}$, y por tanto, los menos probables de estar en $\mathcal{P}$. Si se consigue demostrar que un problema $\mathcal{NP}$-*Complete* está en $\mathcal{P}$ (o que no está), quedaría demostrado que $\mathcal{P} = \mathcal{NP}$ (o que $\mathcal{P} \neq \mathcal{NP}$).

A continuación definimos una última clase de problemas, conocida como *NP-Difícil* y denotada por $\mathcal{NP}$-*Hard*.

Definición 2.4. Decimos que un problema de decisión H está en la clase $\mathcal{NP}$-*Hard* si todo problema $L \in \mathcal{NP}$ puede ser transformado polinomialmente en H.

La clase $\mathcal{NP}$-*Hard* es aquella que contiene a los problemas de decisión que son tan difíciles o más que los problemas de $\mathcal{NP}$. El problema de *la parada* está en $\mathcal{NP}$-*Hard* pero no está en $\mathcal{NP}$, por lo que estas clases son distintas. Se puede observar que $\mathcal{NP}$-*Complete* $= \mathcal{NP}$-*Hard* $\cap \mathcal{NP}$. Si se encuentra un algoritmo que resuelva un problema en $\mathcal{NP}$-*Hard* en tiempo polinomial, nuevamente podríamos concluir que $\mathcal{P} = \mathcal{NP}$.

En la Figura 2.1 resumimos de manera gráfica la relación existente entre las 4 clases definidas. Téngase en cuenta que si finalmente se demostrase que $\mathcal{P} = \mathcal{NP} = \mathcal{NP}$-*Complete*, habría que readaptar dicha figura.

2.2. Optimización combinatoria

En numerosas ocasiones el ser humano se ha visto en la tesitura de decidir qué acción realizar, bajo ciertas circunstancias, de manera que el resultado obtenido pueda ser considerado, de alguna forma, el mejor posible. Según la Real Academia Española (RAE), el significado de *optimizar* es: *buscar la mejor manera de realizar una actividad*. El estudio de los problemas de optimización se encuentra dentro de la rama de las matemáticas conocida como *Investigación*

[2] Dada una expresión lógica con variables y sin cuantificadores, el problema SAT determina si existe alguna asignación de valores para sus variables que hace que la expresión sea verdadera.

UNIVERSIDAD DE MÁLAGA

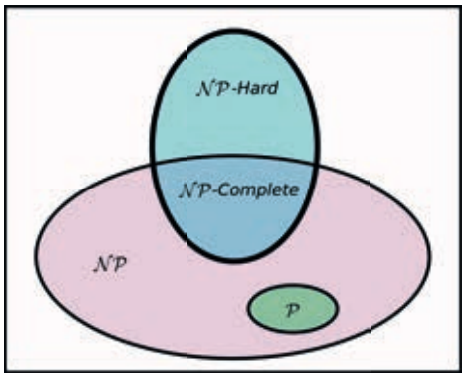


FIGURA 2.1: Clases $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-*Complete* y $\mathcal{NP}$-*Hard*.

de operaciones (OR, del inglés *Operational Research*) o también *Investigación Operativa*. Los problemas de optimización que requieren herramientas matemáticas son conocidos bajo el término de *programación matemática*.

Un programa matemático es una clase especial de problemas de optimización donde usamos de manera eficiente una serie de recursos limitados para satisfacer nuestros objetivos [160]. Este objetivo a conseguir generalmente responde al de maximizar o minimizar una cierta función, que llamaremos *función objetivo*, y está sujeto a una serie de restricciones, expresadas generalmente por medio de ecuaciones o inecuaciones matemáticas, que modelan los recursos de los que disponemos [136]. Es importante resaltar que la programación matemática no es lo mismo que la *programación informática*. En el primer caso el término 'programación' se refiere a un sentido de *planificación* [181].

Como norma general, podemos expresar un problema de optimización de la siguiente forma:

$$\begin{aligned} \text{mín} \quad & f(x), \\ s.a. \quad & x \in \mathcal{X}. \end{aligned} \tag{2.1}$$

La función $f(x)$ es la función objetivo que queremos minimizar. El conjunto $\mathcal{X}$ se llama *conjunto factible* y contiene todos aquellos vectores x que satisfacen las restricciones del problema. La función objetivo, $f : \mathcal{X} \longrightarrow \mathbb{R}$, nos proporciona un indicador de calidad asociado a cada vector de decisión, que en este caso es un valor real el cual queremos minimizar. El conjunto $Im(f)$ es un subconjunto del *espacio objetivo*, que es la recta real. En el caso de querer maximizar una cierta función objetivo, sin pérdida de generalidad, podemos usar la propiedad $\text{máx}\, f(x) = -\,\text{mín}(-f(x))$. Dada la estrecha relación entre ambos tipos, usaremos generalmente la forma de 'minimización'. Cuando la función $f(x)$ y las restricciones dadas por $\mathcal{X}$ vengan expresadas mediante fórmulas matemáticas, diremos que el problema de optimización es un problema de programación matemática.

Veamos un ejemplo, sencillo pero ilustrativo, de un problema de optimización.

Ejemplo 2.1. *Se desea invertir 15000 euros en dos tipos de acciones, A_1 y A_2. La acción tipo A_1 renta al 9 % anual, mientras que la de tipo A_2 lo hace al 5 %. ¿Cuánto hay que invertir en cada tipo de acción si queremos optimizar la rentabilidad, sabiendo además que el máximo en A_1 es de 9000 euros, el mínimo en A_2 es de 3000 euros, y que la inversión en A_1 no se puede ser menor que en A_2?*

Este problema se puede modelar sencillamente usando programación matemática. Si llamamos x_1 al número de euros invertidos en acciones de tipo A_1 y x_2 al número de euros invertidos en acciones de tipo A_2, una formulación del modelo a resolver sería:

UNIVERSIDAD DE MÁLAGA

$$\begin{aligned} \text{máx} \quad & (0.09x_1 + 0.05x_2), \\ s.a. \quad & x_1 + x_2 \leq 15000, \\ & x_1 \geq x_2, \\ & 0 \leq x_1 \leq 9000, \\ & x_2 \geq 3000. \end{aligned}$$

La función objetivo (a maximizar) indica la rentabilidad anual obtenida en función de la cantidad ingresada (despreciando costes de gestión o fiscales, por simplicidad). Cada inecuación representa una de las distintas restricciones del problema. El espacio de decisiones es $\mathbb{R}^2$ y el conjunto factible está formado por todos los vectores de decisión $x = (x_1, x_2)$ que satisfacen las restricciones. La solución del problema es $x = (9000, 6000)$ y proporciona una rentabilidad de 1110 euros, que es la óptima en este caso.

Los problemas de optimización se usan en prácticamente todas las disciplinas científicas, tanto técnicas como aplicadas. En particular, resuelven problemas de logística [185], planificación [8], de Ingeniería del Software [57], finanzas [34], biología [80], problemas de localización [67], redes de tráfico [26], etc. En esta tesis nos centraremos en un tipo particular muy concreto, los problemas combinatorios, que describiremos más adelante.

Clasificación de los problemas de programación matemática

Existen diversas clasificaciones de los problemas de programación matemática atendiendo a distintos criterios. Algunas de las más significativas se muestran a continuación.

- Según el número de objetivos a optimizar: *monoobjetivo* o *multiobjetivo*.
- Según el carácter de las variables de decisión: *discreto*, *continuo* o *mixto*.
- Según el carácter analítico de la función objetivo: *lineal* o *no lineal*.
- Según el grado de certeza con el cuál se conocen los parámetros: *determinista* o *estocástico*.

La optimización o programación matemática *multiobjetivo*, que será introducida en la Sección 2.4, utiliza más de un objetivo a optimizar, y éstos no poseen, a priori, ninguna relación funcional entre sí. Decimos que una variable de decisión es *discreta* si solamente puede tomar un número finito o infinito numerable[3] de valores. En caso contrario la variable es *continua*. Los modelos *mixtos* poseen variables de ambos tipos: discretas y continuas. Dentro de los modelos discretos, cuando las variables solo toman valores enteros hablamos de *programación entera*, y cuando las variables son binarias, *programación binaria*. La programación *lineal* será brevemente introducida en la Sección 2.3. La programación *no lineal* estudia problemas en los que, o bien las restricciones, o bien la función objetivo, no tienen una expresión completamente lineal. Dentro de este campo merece especial mención la programación *convexa* [16]. En un modelo *determinista*, el resultado obtenido es siempre el mismo si las condiciones iniciales se mantienen. En los modelos *estocásticos* los parámetros son inciertos y se expresan mediante variables aleatorias, por lo que los resultados finales podrían variar aún cuando las condiciones de partida fuesen las mismas.

Problemas combinatorios

No es sencillo encontrar una definición formal de lo que es un problema de optimización combinatoria. Algunas referencias bibliográficas describen la combinatoria como la disciplina encargada de hallar la cantidad de objetos que poseen una determinada propiedad dentro de un conjunto finito. En otras palabras, hay que enumerar o contar de alguna forma los elementos de dicho conjunto. Si el problema combinatorio es de optimización, entonces habrá que encontrar aquél elemento (que podría no ser único) que, satisfaciendo las propiedades exigidas, maximice o minimice cierta función objetivo o de calidad. Basándonos en la definición dada por Ehrgott [58] en el año 2005, lo definiremos de una manera algo más rigurosa.

[3] Un conjunto es *infinito numerable* si existe una biyección entre dicho conjunto y $\mathbb{N}$.

UNIVERSIDAD DE MÁLAGA

Definición 2.5. Sea E un conjunto finito. Definamos un coste asociado a cada elemento del conjunto, $c : E \longrightarrow \mathbb{Z}$. Una instancia de un problema de *optimización combinatoria* es un par $(f, \mathcal{X})$ donde $\mathcal{X} \in \mathscr{P}(E)$, siendo $\mathscr{P}$ el conjunto *partes de un conjunto*[4], y $f : \mathcal{X} \longrightarrow \mathbb{Z}$ una función objetivo a minimizar.

Analizando la definición anterior, podemos observar en primer lugar que los problemas combinatorios también son válidos en el caso en que la función de costes esté definida para los números racionales, $\hat{c} : E \longrightarrow \mathbb{Q}$, sin más que definir $c(x) = \kappa \cdot \hat{c}(x)$, siendo κ el mínimo común múltiplo de todos los denominadores de los costes $x \in E$. Aunque la función de costes podría tomar incluso cualquier valor real, se mantiene esta definición, ya que al trabajar con números irracionales, la precisión del computador terminaría aproximándolos siempre a números racionales.

En segundo lugar, observamos que los modelos combinatorios tienen un número finito de soluciones factibles. Por tanto, estos modelos siempre pueden ser resueltos mediante enumeración exhaustiva, si bien es lógico pensar que resulta intratable hacerlo cuando crece el número de elementos en E.

Ejemplo 2.2. Resolver mediante enumeración un problema de optimización combinatoria con 70 variables lógicas requeriría analizar 2^{70} elementos. Aunque fuésemos capaces de computar cada uno de ellos en un nanosegúndo e ignorásemos el tiempo empleado en ejecutar las restantes instrucciones, necesitaríamos más de 37.400 años para resolver el problema.

En muchos casos los problemas de optimización combinatoria se pueden modelar mediante programación lineal binaria. Si $E = \{e_1, \ldots, e_n\}$ y definimos el vector binario $x \in \{0,1\}^n$ como $x_i = 1$ si $e_i \in x$; $x_i = 0$ en otro caso, y esto lo hacemos $\forall i = 1, \ldots, n$, obtenemos una biyección entre $\mathscr{P}(E)$ y $\{0,1\}^n$. Dado que cualquier variable entera se puede expresar como combinación lineal de variables binarias, también podemos concluir que los problemas de programación lineal entera con región factible acotada son problemas combinatorios.

Mostramos a continuación algunos ejemplos clásicos de problemas de optimización combinatoria.

Ejemplo 2.3. Problema de asignación (*AP*, *Assignment Problem*).

Dados n agentes o recursos, queremos ejecutar n tareas al menor coste posible sabiendo que cada recurso se destina a una sola tarea, y que cada tarea es ejecutada por un único recurso. Formalmente, si A es el conjunto de agentes y T el conjunto de tareas, disponemos de una función de costes $C : A \times T \longrightarrow \mathbb{R}$. Queremos encontrar una biyección $\varphi : A \longrightarrow T$ tal que el coste total de la asignación sea mínimo: $\text{mín}_{x \in A}\, C(x, \varphi(x))$.

La función de costes puede ser representada mediante una matriz cuadrada $C = (c_{ij})$, y el problema AP se transforma en un problema de optimización binario con restricciones de igualdad. Si definimos $\forall i, j \in \{1, \ldots, n\}$ las variables $x_{ij} = 1$, si el agente i tiene asignada la tarea j; $x_{ij} = 0$, en otro caso, el modelo a resolver es

$$\text{mín} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{2.2}$$

$$s.a. \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, \ldots, n, \tag{2.3}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, n, \tag{2.4}$$

$$x_{ij} \in \{0,1\} \quad \forall i, j = 1, \ldots, n. \tag{2.5}$$

Dada la estructura particular del programa matemático dado por las ecuaciones (2.2)–(2.5), aunque siempre tendríamos la opción de comprobar las $n!$ combinaciones posibles, se han encontrado algoritmos que lo resuelven en tiempo polinomial, como el algoritmo Húngaro [108], que es $O(n^3)$, por lo que este problema combinatorio está en la clase $\mathcal{P}$.

[4] El conjunto *partes de un conjunto* es aquel cuyos elementos son todos los posibles subconjuntos de dicho conjunto.

UNIVERSIDAD DE MÁLAGA

Ejemplo 2.4. Problema de la mochila (*KP*, *Knapsack Problem*).

Dados n objetos, enumerados en el conjunto $\{1, \ldots, n\}$, cada objeto i tiene un peso w_i y un beneficio o valor v_i. Tanto los pesos como los beneficios se expresan mediante números reales positivos. Disponemos de una mochila que soporta un peso máximo $W > 0$ y queremos introducir en la mochila varios de estos objetos de manera que se maximice el beneficio, sin superar la capacidad máxima de la mochila.

El problema de la mochila, también conocido como *0-1 Knapsack problem*, es sencillo de modelar mediante programación lineal entera. Si definimos $\forall i \in \{1, \ldots, n\}$ las variables x_i, de forma que $x_i = 1$, si el objeto i es introducido en la mochila; $x_i = 0$, si no lo está, tenemos que

$$\text{máx} \quad \sum_{i=1}^{n} v_i x_i, \tag{2.6}$$

$$s.a. \quad \sum_{i=1}^{n} w_i x_i \leq W, \tag{2.7}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n. \tag{2.8}$$

A pesar de que este problema solo tiene una restricción, y es de tipo lineal, el problema de la mochila es uno de los 21 problemas de la clase $\mathcal{NP}$-*Complete* (véase Karp [98]).

Ejemplo 2.5. Problema de la mochila multidimensional (*MKP*, *Multidimensional Knapsack Problem*).

Disponemos de m mochilas con capacidades $c_j > 0$, $\forall j = 1, \ldots, m$. Dados n objetos, cada objeto i consume una cantidad recursos $w_{ij} > 0$ asociado a la mochila j. El objeto i también produce un beneficio o valor $v_i > 0$ si finalmente es seleccionado. Queremos obtener un subconjunto de objetos que maximice el beneficio sin superar las capacidades de las mochilas.

Se puede ver que este ejemplo es una generalización del problema de la mochila (ambos problemas coinciden cuando $m = 1$). Definiendo nuevamente $x_i = 1$, si el item i es seleccionado; $x_i = 0$, en caso contrario, su formulación mediante programación lineal entera puede expresarse como:

$$\text{máx} \quad \sum_{i=1}^{n} v_i x_i, \tag{2.9}$$

$$s.a. \quad \sum_{i=1}^{n} w_{ij} x_i \leq c_j \quad \forall j = 1, \ldots, m, \tag{2.10}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n. \tag{2.11}$$

El problema *MKP* se encuentra en la clase $\mathcal{NP}$-*Hard* (véase [153]).

Ejemplo 2.6. Problema de asignación cuadrática (*QAP*, *Quadratic Assignment Problem*).

Consideremos el problema de asignar n instalaciones en n localizaciones, donde el coste es una función que depende de la distancia y el flujo entre las instalaciones más el coste asociado a asignar la instalación en dicha localización. El objetivo es asignar las instalaciones de manera que el coste total empleado sea mínimo.

Uno de los capítulos del famoso *Handbook of combinatorial optimization* [145] define formalmente este problema, basándose en la formulación de Koopmans y Beckmann [107]. Consideramos tres matrices cuadradas reales de orden n, $F = (f_{ij})$, $D = (d_{kl})$ y $B = (b_{ik})$, donde f_{ij} indica el flujo entre la instalación i y j; d_{kl} es la distancia entre la localización k y l; y b_{ik} es el coste de asignar la instalación i a la localización k. El problema *QAP* puede ser formulado de la siguiente manera:

$$\min_{\varphi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} + \sum_{i=1}^{n} b_{i\varphi(i)}, \tag{2.12}$$

UNIVERSIDAD DE MÁLAGA

donde S_n es el conjunto de las permutaciones $\varphi : \mathbb{N} \longrightarrow \mathbb{N}$. Aunque el modelo de la Ecuación (2.12) no tiene restricciones en su versión original, *QAP* es considerado uno de los problemas combinatorios más duros de resolver, aún existiendo modelos de programación matemática que linealizan el problema [183].

Ejemplo 2.7. Problema del recubrimiento mínimo (*SCP*, *Set Covering Problem*).

Es uno de los problemas pertenecientes a la clase $\mathcal{NP}$-*Complete*. Dado un universo U con un número finito de elementos y una colección de subconjuntos de U, denotada por S, donde cada elemento de S tiene asociado un coste positivo (no se permite que contenga al subconjunto vacío), el problema del recubrimiento mínimo consiste en encontrar un subconjunto $C \subseteq S$ cuya unión de elementos sea el conjunto U y tenga coste mínimo.

Formalmente, y utilizando programación lineal, para un universo U con n elementos, y colección de subconjuntos S con m elementos, el modelo es el siguiente:

$$\text{mín} \quad \sum_{j=1}^{m} c_j x_j, \tag{2.13}$$

$$s.a. \quad \sum_{j=1}^{m} a_{ji} x_j \geq 1 \quad \forall i = 1, \ldots, n, \tag{2.14}$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, m, \tag{2.15}$$

donde x_j toma el valor 1 si el subconjunto j de S es finalmente seleccionado, y 0 en otro caso. El valor c_j representa el coste del elemento j en S. Finalmente, el valor a_{ji} vale 1 si y solo si el subconjunto j de S contiene al elemento i de U, y vale 0 en otro caso.

2.3. Programación lineal

La Investigación Operativa utiliza el método científico en la búsqueda de soluciones óptimas, y estas sirven de apoyo en la toma de decisiones. El término OR se comienza a utilizar sobre el año 1940, durante la segunda guerra mundial, donde se crearon grupos de investigación que empezaron a resolver problemas de control de inventarios, asignación de recursos, reducción de costes y estrategias de defensa. No obstante, el origen real de esta disciplina se remonta al siglo XVII, durante la Revolución Industrial. Uno de los campos de estudio en la OR es la Programación Lineal (LP, del inglés *Linear Programming*). En 1947, George B. Dantzig diseñó un algoritmo general capaz de resolver cualquier problema de programación lineal, conocido como algoritmo *simplex*. Este método, publicado en 1951 (véase [38]) marcó un antes y un después en el mundo de la investigación operativa, dada la enorme utilidad y aplicabilidad, y aún hoy día se sigue usando como algoritmo de referencia. Numerosas empresas han desarrollado paquetes software que resuelven problemas LP con cientos de miles de variables. Un ejemplo lo tenemos en la empresa ILOG, que desarrolló el paquete comercial CPLEX, muy usado en entornos de investigación, y cuya pequeña descripción se puede encontrar en la Sección 3.1.4. El matemático húngaro John Von Neumann desarrolló la teoría de la dualidad, que define un problema dual basado en el problema original, llamado primal. Existen multitud de libros y guías didácticas relacionados con la programación lineal. Algunos libros interesantes son [39], [173] y [79].

Antes de explicar someramente en qué consiste el método simplex, definamos previamente una serie de conceptos, extraídos del libro de Boyd and Vandenberghe [16].

Definición 2.6. Un conjunto C es *convexo* si el segmento lineal que une cualesquiera dos puntos en C está formado por puntos en C, es decir, si para cualesquiera $x_1, x_2 \in C$ y cualquier θ con $0 \leq \theta \leq 1$, tenemos que $\theta x_1 + (1 - \theta) x_2 \in C$.

Definición 2.7. Una función $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ es *convexa* si su dominio es un conjunto convexo y para todo x, y del dominio de f se tiene que $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$, siendo $0 \leq \theta \leq 1$.

Los problemas de programación convexa son aquellos en los que tanto la función objetivo como las restricciones son funciones convexas. Estos problemas pueden resolverse utilizando

UNIVERSIDAD DE MÁLAGA

el método de *mínimos cuadrados*. De hecho, los problemas LP son problemas de optimización convexa, aunque el método simplex use una técnica distinta a la de mínimos cuadrados para la resolución.

Definición 2.8. Definimos un *hiperplano* como el conjunto de puntos de la forma $\{x \mid a^T x = b\}$, donde $a \in \mathbb{R}^n$, $a \neq 0$, y $b \in \mathbb{R}$.

Analíticamente podríamos decir que un hiperplano es el conjunto de soluciones no triviales en un sistema de ecuaciones lineales. Geométricamente podríamos interpretar un hiperplano como un conjunto con una dimensión una unidad menor que la del espacio afín considerado. Por ejemplo, para la recta real $\mathbb{R}$ un punto sería un hiperplano; para $\mathbb{R}^2$ un hiperplano es una recta; para el espacio $\mathbb{R}^3$ un hiperplano es un plano en el espacio. Es fácil comprobar que los hiperplanos son conjuntos convexos. Un hiperplano divide el espacio en dos *semiespacios*: $\{x \mid a^T x \leq b\}$ y $\{x \mid a^T x > b\}$. Estos semi-espacios se representan analíticamente mediante desigualdades.

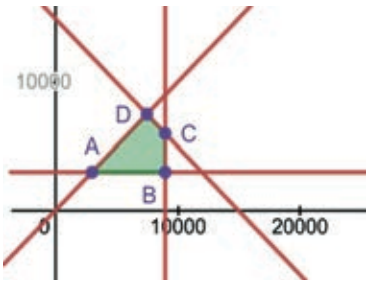
Definición 2.9. Un *poliedro* es el conjunto solución de un número finito de ecuaciones o inecuaciones lineales. Equivalentemente, es la intersección de un número finito de hiperplanos o semiespacios. Un poliedro acotado recibe el nombre de *politopo*.

A estas alturas el lector habrá podido deducir que las restricciones de un problema de programación lineal son hiperplanos del espacio afín $\mathbb{R}^n$, y que por tanto, la intersección de todas ellas, conocida como región factible, es un poliedro. Dado que la intersección de conjuntos convexos es otro conjunto convexo, concluimos que cualquier poliedro es convexo.

El método simplex explora una serie de puntos de un poliedro y trata de encontrar aquel cuya función objetivo toma el valor óptimo. El conocido como *teorema de la programación lineal* establece que dicho óptimo, en caso de existir, debe encontrarse en un vértice del poliedro. Así, Dantzig diseñó un método algebraico en el que, a partir de una solución factible inicial, nos dirigimos hacia un vértice del poliedro y nos vamos moviendo por distintos vértices (uno en cada iteración) hasta que encontremos uno cuyo valor objetivo no pueda ser mejorado, y que será la solución óptima del problema. De manera teórica, el método simplex tiene complejidad exponencial, aunque en la práctica se ha determinado que en el caso promedio está complejidad es mucho menor, polinomial en la mayoría de los casos [162].

Cuando varios vértices del poliedro tienen el mismo valor objetivo, y son el óptimo, entonces cualquier punto determinado por el segmento lineal que une dichos vértices es también una solución óptima. En este caso decimos que hay *óptimos alternativos*. También puede ocurrir que no exista solución alguna al problema, en cuyo caso decimos que el modelo es *infactible*. Un último caso es el de problema *no acotado*, que es aquel en el que tampoco obtenemos una solución óptima, ya que en la aplicación del método simplex las soluciones se pueden ir mejorando indefinidamente.

Retomemos el Ejemplo 2.1 para ilustrar gráficamente este problema. Dado que solo tenemos dos variables, podemos representar el espacio de decisiones de manera gráfica. En la Figura 2.2 podemos observar que al representar las cuatro restricciones, la región factible es un poliedro convexo. Según el teorema de la programación lineal, el óptimo debe encontrarse en un vértice del poliedro, por lo que, aún sin conocer el funcionamiento interno del método simplex, bastaría con evaluar la función objetivo en sus cuatro vértices para hallar el óptimo. Tras los cálculos, concluimos que es necesario invertir 9000 euros en las acciones de tipo A_1 y 6000 euros en las de tipo A_2, para obtener una rentabilidad máxima de 1110 euros.



$$f(x_1, x_2) = 0.09x_1 + 0.05x_2$$
$$f(A) = f(3000, 3000) = 420$$
$$f(B) = f(9000, 3000) = 960$$
$$f(\mathbf{C}) = f(9000, 6000) = \mathbf{1110}$$
$$f(D) = f(7500, 7500) = 1050$$

Figura 2.2: Resolución gráfica de un problema de programación lineal.

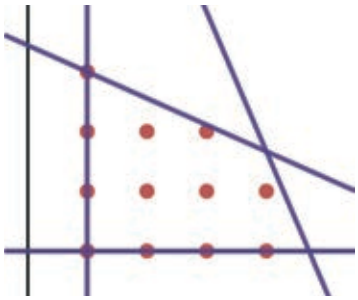
UNIVERSIDAD DE MÁLAGA

Programación lineal entera

Cuando las variables de decisión son discretas, estamos ante un problema de programación lineal entera (ILP, del inglés *Integer Linear Programming*). En este caso, la región factible estaría formada por puntos aislados que no forman un conjunto convexo, por lo que no podríamos usar el método del simplex para la resolución. Véase el ejemplo ilustrativo de la Figura 2.3, donde se define un modelo ILP y se representa su región factible.

$$
\begin{aligned}
\text{máx} \quad & 9x_1 + 20x_2, \\
s.a. \quad & 4x_1 + 9x_2 \leq 31, \\
& 7x_1 + 3x_2 \leq 33, \\
& x_1 \geq 1, \\
& x_1, x_2 \in \mathbb{N}.
\end{aligned}
$$

(a) Modelo ILP.



(b) Región factible.

Figura 2.3: Ejemplo ilustrativo de problema de programación lineal entera.

Una vez identificada la región factible, podríamos calcular el valor objetivo de cada uno de los puntos y obtener de esa manera la solución óptima. Obviamente, esto es precisamente un método enumerativo, y resulta inviable su aplicación cuando crece el número de variables de decisión y, consecuentemente, el número de puntos que componen la región factible.

Técnicas de resolución de problemas ILP

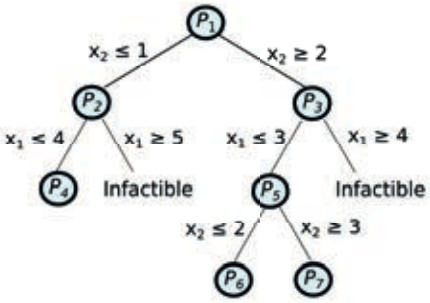
Una de las técnicas más usuales que se emplean a la hora de resolver problemas ILP se conoce como *ramificación y acotación*. Se explicará brevemente en qué consiste este método y se aplicará en la resolución del problema del ejemplo anterior. Antes de ello definamos en qué consiste la *relajación lineal* de un modelo ILP. Consiste en eliminar las restricciones de integralidad de las variables, manteniendo inalteradas las restantes restricciones. Nótese que la relajación lineal es un modelo LP, y por tanto, podemos aplicar el método simplex. Si al aplicar el simplex obtenemos que la solución óptima tiene componentes enteras, entonces dicha solución será también solución de nuestro problema entero. En la Figura 2.4c podemos observar gráficamente la región factible obtenida mediante la relajación lineal del modelo ILP de la Figura 2.3b. El poliedro obtenido tiene vértices A, B, C y D, dos de los cuales no tienen todas sus coordenadas enteras. De hecho, el valor óptimo se obtiene para el vértice $C(4, 5/3)$, que no es una solución factible de nuestro problema. En esta situación cabría pensar que el valor óptimo podría encontrarse en la solución enterá más próxima a nuestro vértice C. Esto no tiene por qué ser así, como comprobaremos más adelante. Obsérvese que en el caso de problemas de maximización podemos garantizar que el óptimo obtenido en su relajación lineal tiene un valor objetivo mayor o igual que el óptimo del problema entero. Esto es evidente si tenemos en cuenta que el problema entero tiene una región factible que es un subconjunto del problema original (por tanto, el relajado es menos restrictivo). Equivalentemente, en el caso de minimización, el valor objetivo óptimo obtenido en su relajación lineal será una cota inferior del valor objetivo óptimo del problema entero.

Cuando la solución óptima del problema relajado no tiene coordenadas enteras (lo que ocurrirá en la mayoría de los casos), procedemos a ramificar nuestro problema, que consiste dividir el modelo en dos nuevos, cada uno de ellos con una restricción adicional, de manera que podamos garantizar que el óptimo al resolver estos dos modelos será también el óptimo del problema original. En el proceso de ramificación se elige una coordenada no entera para el vector de decisión óptimo obtenido en el problema relajado. Por ejemplo, si $x_i = a$, siendo $a \in \mathbb{Q} - \mathbb{Z}$, elegimos la variable x_i para ramificar. El primer subproblema será igual que el anterior, pero añadiendo la restricción $x_i \leq \lfloor a \rfloor$. Al segundo subproblema le añadimos la restricción $x_i \geq \lfloor a \rfloor + 1$. Así, dada la integralidad de las variables de decisión, garantizamos

UNIVERSIDAD DE MÁLAGA

que el óptimo se encuentra en alguno de los dos subproblemas. Si al resolver la relajación lineal de cualquiera de estos dos nuevos problemas volvemos a obtener un óptimo con alguna coordenada no entera, repetimos el mismo proceso de ramificación.

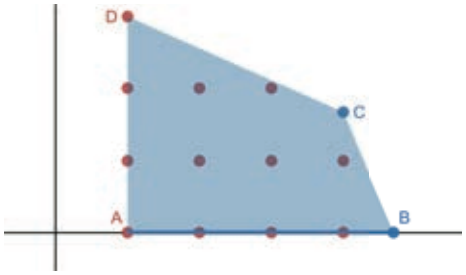
Para una mejor comprensión del método de ramificación y acotación podemos pensar en el *árbol de búsqueda* del procedimiento, donde cada nodo representa un subproblema a resolver, y cada arista relaciona un subproblema con otro si ambos difieren solo en una restricción. El nodo inicial, o nodo *padre* del árbol de búsqueda, representa la relajación lineal del modelo inicial. La ramificación consiste, por tanto, en crear dos nuevos nodos a partir de un nodo concreto, y relacionar ambos con el nodo de partida mediante una arista. Resumiendo, el método empleado consiste en resolver una serie de problemas de programación lineal (recorrido del árbol de búsqueda) hasta que, una vez finalizada la exploración, nos quedemos con el valor óptimo. La exploración de este árbol se hace en tiempo finito, como justificaremos a continuación.



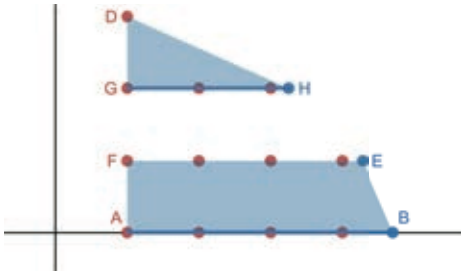
(a) Árbol de búsqueda.

Nodo	Extremos poliedro	Mejor vértice	Objetivo	Óptimo
P_1	A,B,C,D	$C(4,5/3)$	$208/3$	-
P_2	A,B,E,F	$E(30/7,1)$	$410/7$	-
P_3	D,G,H	$H(13/4,2)$	$277/4$	-
P_4	A,F,I,J	$J(4,1)$	56	J
P_5	D,G,K,L	$L(3,19/9)$	$623/9$	J
P_6	G,K	$K(3,2)$	67	K
P_7	D	$D(1,3)$	69	D

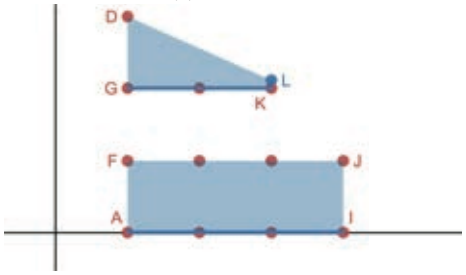
(b) Recorrido en amplitud.



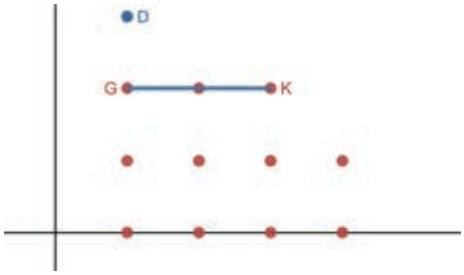
(c) Nodo P_1.



(d) Nodos P_2 y P_3.



(e) Nodos P_4 y P_5.



(f) Nodos P_6 y P_7.

Figura 2.4: Resolución de un problema de PLE usando ramificación y corte.

Cada vez que exploramos un nodo (es decir, cada vez que resolvemos un problema LP) puede ocurrir lo siguiente:

- Obtenemos una solución óptima con coordenadas enteras, en cuyo caso no exploramos más esa rama.
- El problema es infactible, en cuyo caso no generamos más nodos a partir de él.
- Obtenemos un óptimo con alguna coordenada no entera. En ese caso seleccionamos una coordenada y ramificamos en base a ella, creando dos nuevos nodos a explorar.

UNIVERSIDAD DE MÁLAGA

En el tercer caso, observamos que las restricciones $x_i \leq \lfloor a \rfloor$ y $x_i \geq \lfloor a \rfloor + 1$ reducen el espacio de búsqueda, ya que suprimimos la región $\lfloor a \rfloor < x_i < \lfloor a \rfloor + 1$, que sabemos que es no vacía, porque el valor a no es entero. Por tanto, si la región factible es acotada, lo cual ocurre para los problemas combinatorios, podemos garantizar que en algún momento terminaremos de explorar el árbol de búsqueda y hallar el óptimo.

Recorrer el árbol de búsqueda puede resultar muy costoso, ya que puede requerir de un tiempo que es exponencial en base al número de variables de decisión. Aquí es donde entra en juego la técnica de la acotación, que permite no tener que ramificar ciertos nodos del árbol de búsqueda cuando sabemos que yendo por esa zona no vamos a encontrar nunca la solución óptima. Para la técnica de acotación tenemos en cuenta que al resolver un problema lineal relajado, la solución obtenida es una cota superior (o inferior, según el caso) del verdadero óptimo de nuestro problema. Para el caso de maximización, si la cota superior obtenida es menor que la mejor solución encontrada hasta el momento, es obvio que podemos desechar ese nodo y no ramificarlo más.

Aplicamos la técnica de ramificación y acotación para el ejemplo de la Figura 2.3. Para una mejor comprensión de este método, mostramos el árbol de búsqueda en la Figura 2.4a, así como una resumen en la Tabla 2.4b.

Inicialmente, el nodo P_1 resuelve la relajación lineal, que como ya vimos, nos proporciona como óptimo el punto $C(4, 5/3)$. Debido a que la segunda variable verifica $1 < 5/3 < 2$, generamos dos nuevos nodos, P_2 y P_3 añadiendo las restricciones $x_2 \leq 1$ y $x_2 \geq 2$, respectivamente. Existen distintas técnicas a la hora de explorar el árbol de búsqueda (en profundidad, en amplitud, aleatoriamente, etc). En este ejemplo, que tiene solo carácter ilustrativo, realizaremos una búsqueda en amplitud. Las regiones factibles de las relajaciones lineales de los problemas P_2 y P_3 podemos verlas en la Figura 2.4d. Para P_2 obtenemos un nuevo óptimo con coordenadas no enteras, $E(30/7, 1)$, por lo que tenemos que crear dos nuevos subproblemas, esta vez ramificando la variable x_1. Lo mismo ocurre a la hora de resolver el problema P_3. El nodo P_4 es el primero cuya relajación lineal proporciona un óptimo que es factible en el problema original (punto J). Su valor objetivo, 56, será guardado como el mejor valor obtenido hasta el momento, por lo que si algún nodo nos proporciona un valor objetivo menor a 56, podemos desecharlo y no seguir ramificando esa parte. El resto de la exploración del árbol de búsqueda se puede ver en las figuras 2.4e y 2.4f, donde el mejor óptimo encontrado hasta el momento se actualiza en dos ocasiones, hasta concluir que el óptimo final se alcanza en el punto $D(1, 3)$ cuyo valor objetivo es 69.

Obsérvese que este problema ha resuelto un total de 9 relajaciones lineales, y que además, la solución óptima está 'lejos' del óptimo del problema relajado inicial. Nótese también como todas las regiones factibles en el árbol de búsqueda son conjuntos convexos, incluso en los casos en que la región factible es un segmento, o un punto (véase la Figura 2.4f).

El método de ramificación y acotación es aplicable también a problemas de programación lineal mixta, que contienen variables discretas y continuas. En este caso es obvio que la ramificación siempre se producirá para aquellas variables que sean enteras. Cuanto mayor sea el número de variables enteras en un modelo, 'mayor' esfuerzo tendrá que realizarse en la exploración del árbol de búsqueda. Sin embargo, en algunos casos se ha constatado que añadir nuevas restricciones (nuevos cortes) a un modelo, puede reducir considerablemente el número de vértices del poliedro a explorar, y por tanto, podría obtenerse una mejora en el tiempo computacional empleado hasta obtener la solución óptima. Existen en la literatura muchos trabajos que estudian los poliedros y proporcionan métodos *ad-hoc* eficientes, como por ejemplo en [126], dedicado a resolver el problema de la mochila.

El resolutor CPLEX es capaz de resolver problemas ILP. Para ello, utiliza una técnica conocida como *ramificación y corte*. Este método es un híbrido entre el método de ramificación y acotación y el *método de los planos de corte*. Concretamente, como método de plano de corte utliza los cortes de Gomory [78]. Detectada una solución óptima en un problema relajado, un corte de Gomory consiste en añadir una nueva restricción (hiperplano) de manera que el óptimo del problema relajado no verifica la condición del hiperplano (por lo que queda excluido desde ese momento), y todas las soluciones enteras del problema satisfacen la restricción del nuevo hiperplano. En la práctica computacional, podrían producirse errores en los cálculos debido a los redondeos que utiliza la máquina, de manera que se podrían omitir soluciones válidas. Un buen trabajo que solventa este problema puede verse en [30].

UNIVERSIDAD DE MÁLAGA

2.4. Optimización multiobjetivo

En este apartado introducimos los conceptos básicos asociados a los problemas de decisión multicriterio desde el punto de vista de la optimización combinatoria multiobjetivo. Cuando varios objetivos se deben optimizar a la vez, es común que no pueda hacerse para todos ellos simultáneamente. Esto quiere decir que tomar una decisión que mejore el valor para un objetivo, normalmente implica empeorar el valor en otro objetivo. En función del tipo de problema, el decisor podría disponer de una serie de preferencias para ciertos objetivos, o no tenerlas, y tras aplicar el algoritmo concreto, se le mostrarán una serie de alternativas, todas ellas 'óptimas' en cierto sentido.

Para los problemas multiobjetivo (MOO, del inglés *MultiObjective Optimization*), estamos interesados en optimizar varios objetivos a la vez, todo ello sujeto a una serie de restricciones. Un problema de *optimización multiobjetivo* puede definirse de la siguiente forma:

$$\min f(x) = (f_1(x), \ldots, f_p(x)), \tag{2.16}$$
$$s.a. \quad x \in \mathcal{X}. \tag{2.17}$$

En este caso, disponemos de p funciones objetivo a optimizar. A diferencia del caso de optimización monoobjetivo (ver Ecuación (2.1)), ahora nuestra función objetivo es una función de varias variables, $f : \mathcal{X} \longrightarrow \mathbb{R}^p$, siendo $\mathcal{X}$ el conjunto de soluciones factibles al problema. Si se deseara maximizar alguno de los objetivos, sea i, aplicamos la propiedad $\max f_i(x) = -\min(-f_i(x))$, multiplicando por -1 cada coeficiente, y volvemos a tomar el opuesto una vez obtenido el valor óptimo. La región factible seguirá siendo un subconjunto de $\mathbb{R}^n$, siendo n el número de variables de decisión, pero en este caso el espacio objetivo es un subconjunto de $\mathbb{R}^p$.

Antes de formalizar qué significa minimizar una función que está definida en $\mathbb{R}^p$ necesitamos definir una serie de conceptos.

Definición 2.10. Consideramos un conjunto S sobre el que tenemos definido una relación binaria, R. Decimos que la relación R es:

- Reflexiva, si $x\,R\,x$ para todo $x \in S$.
- Antisimétrica, si para cualesquiera $x, y \in S$ tales que $x\,R\,y$ e $y\,R\,x$, entonces forzosamente $x = y$.
- Transitiva, si para cualesquiera $x, y, z \in S$ tales que $x\,R\,y$ e $y\,R\,z$, entonces se verifica que $x\,R\,z$.
- Asimétrica, si para cualesquiera $x, y \in S$ tales que $x\,R\,y$, entonces $y \not R\, x$[5].
- Conexa, si para cualesquiera $x, y \in S$ con $x \neq y$, se verifica que $x\,R\,y$ o que $y\,R\,x$.

Definición 2.11. Consideramos el conjunto $\mathbb{R}^p$ y una relación binaria $\preceq$ definida sobre él. Decimos que el par $(\mathbb{R}^p, \preceq)$ es

- un orden total, si $\preceq$ verifica las relaciones reflexiva, antisimétrica, transitiva, y conexa;
- un orden parcial, si $\preceq$ verifica las relaciones reflexiva, antisimétrica y transitiva;
- un orden parcial estricto, si $\preceq$ verifica las relaciones asimétrica y transitiva.

Cuando no sea necesario explicitar el tipo, referiremos el término relación de *orden* para cualesquiera de los tres casos definidos anteriormente. En la Tabla 2.3, se muestran las relaciones más conocidas en el conjunto $\mathbb{R}^p$ siendo $p > 1$.

De forma análoga se podrían definir las relaciones $\geqq$, $\geq$, $>$, $\geq_{lex}$, $>_{lex}$ $\geq_{max}$ y $>_{max}$. La relación $\leq_{lex}$ recibe el nombre de *orden lexicográfico*, la relación $<_{lex}$ es llamada *orden lexicográfico estricto* y la relación $\leq_{max}$ se conoce como *max-orden* (del inglés, *max-ordering*), aunque no sea un orden propiamente dicho.

[5] $y \not R\, x$ significa que no se verifica $y\,R\,x$.

UNIVERSIDAD DE MÁLAGA

Tabla 2.3: Algunas relaciones de orden en $\mathbb{R}^p$ con $p > 1$.

Relación	Definición	Clasificación
$x \leqq y$	$x_i \leq y_i \ \forall i = 1, \ldots, p$	orden parcial
$x \leq y$	$x \leqq y$, y además $x \neq y$	orden parcial estricto
$x < y$	$x_i < y_i \ \forall i = 1, \ldots, p$	orden parcial estricto
$x \leq_{lex} y$	$x = y$, o $x_k < y_k$ para $k = \text{mín}\{i \mid x_i \neq y_i\}$	orden total
$x <_{lex} y$	$x \leq_{lex} y$, y además $x \neq y$	orden parcial estricto
$x \leq_{max} y$	$\max\limits_{i=1,\ldots,p} x_i \leq \max\limits_{i=1,\ldots,p} y_i$	-
$x <_{max} y$	$\max\limits_{i=1,\ldots,p} x_i < \max\limits_{i=1,\ldots,p} y_i$	orden parcial estricto

Si generalizásemos las relaciones definidas en la Tabla 2.3 incluyendo el caso $p = 1$, obtenemos que las relaciones $\leqq$, $\leq_{lex}$ y $\leq_{max}$ son iguales entre sí e iguales a la relación 'menor o igual' en $\mathbb{R}$, y que las relaciones $\leq$, $<$, $<_{lex}$ y $<_{max}$ son iguales a la relación 'menor estricto' en $\mathbb{R}$.

Vistas las definiciones anteriores, estamos ya en condiciones de definir el significado de minimización para el caso multiobjetivo.

Definición 2.12. Sea $\preceq$ una relación de orden definida en $\mathbb{R}^p$. Decimos que $x^* \in \mathcal{X}$ es una solución óptima para el problema dado por las Ecuaciones (2.16)–(2.17) si no existe $x \in \mathcal{X}$ con $x \neq x^*$ tal que $f(x) \preceq f(x^*)$.

La definición de óptimo para el caso multiobjetivo viene a significar que será óptimo aquel vector que no sea mejorado por ningún otro si comparamos sus imágenes en el espacio objetivo. La propia definición sugiere la posibilidad de existencia de múltiples óptimos para estos problemas.

Mostramos un ejemplo ilustrativo de problema de optimización multicriterio, que será resuelto considerando tres tipos distintos de optimización.

Ejemplo 2.8. Consideremos el problema de decisión multicriterio

$$\min_{x \in \mathcal{X}} f(x) = (f_1(x), f_2(x), f_3(x)),$$

siendo $\mathcal{X} = \{a, b, c, d\}$. Supongamos que las imágenes en el espacio objetivo de los puntos de la región factible son las siguientes: $f(a) = (3, 5, 6)$, $f(b) = (3, 5, 7)$, $f(c) = (5, 3, 2)$, $f(d) = (3, 7, 2)$.

- Si consideramos el orden $\leq$, el conjunto de soluciones óptimas es $\{a, c, d\}$, y son llamadas soluciones *Pareto óptimas* o *eficientes*.
- Si consideramos el orden $\leq_{lex}$, el conjunto óptimo sería $\{a\}$, que es llamado *óptimo lexicográfico*.
- Si consideramos el orden $<_{max}$, el conjunto óptimo sería $\{b, d\}$, y son los llamados *max-orden* óptimos.

La gran mayoría de los trabajos de investigación existentes en la literatura consideran el par $(\mathbb{R}^p, \leq)$, conocido como orden de Pareto, a la hora de optimizar un problema multiobjetivo. En esta tesis consideraremos siempre este tipo de optimización, salvo que se indique lo contrario. Las definiciones que vienen a continuación están basadas en problemas de minimización (véanse Ecuaciones (2.16)–(2.17)).

Definición 2.13. (Dominancia)

- Dados dos vectores $x, y \in \mathbb{R}^p$, decimos que x es *débilmente dominado* por y o que y *domina débilmente* a x si $y \leqq x$.
- Si además se cumple que $y \leq x$, entonces decimos que y *domina* a x o que x es *dominado* por y.
- Si $y < x$, entonces y *domina estrictamente* a x o x es *estrictamente dominado* por y.

UNIVERSIDAD DE MÁLAGA

Definición 2.14. (Eficiencia) Consideramos $x \in \mathcal{X}$, una solución factible al problema MOO.

- x es *eficiente* si no existe $y \in \mathcal{X}$ tal que $f(y)$ domine a $f(x)$.
- x es *débilmente eficiente* si no existe $y \in \mathcal{X}$ tal que $f(y)$ domine estrictamente a $f(x)$.

Obsérvese que dada una solución eficiente x, su imagen $f(x)$ es un punto no dominado en el espacio objetivo (mismo razonamiento para las soluciones débilmente eficientes). Generalmente usaremos el término de soluciones eficientes o puntos no dominados, aunque en ocasiones se permite usar indistintamente el vocablo 'solución', refiriéndonos a cualquiera de los dos espacios (decisión y objetivo) si no se produce confusión alguna en su interpretación.

Definición 2.15. Al conjunto de puntos eficientes se le denota $\mathcal{X}_E$ y se le llama *conjunto eficiente*. Al conjunto $f(\mathcal{X}_E)$ se le llama *frente de Pareto*, y se le denota por PF (del inglés, *Pareto Front*).

En la mayoría de los problemas multiobjetivos a resolver, estaremos interesados en obtener el frente de Pareto, o una aproximación del mismo. Pueden existir muchos puntos eficientes que den lugar a un mismo punto del frente de Pareto. Es por ello que, en la práctica, se suele determinar una única antiimagen para cada punto del frente calculado o aproximado.

Definición 2.16. Dado un conjunto discreto de puntos, X, definimos su *envoltura convexa* como el conjunto intersección de todos los conjuntos convexos que contienen a X.

Geométricamente, la envoltura convexa de un conjunto de puntos es bastante sencilla de obtener: es el menor subconjunto convexo que contiene a los puntos del conjunto (ver Figura 2.5). Sin embargo, calcular la envoltura convexa de manera analítica para un conjunto discreto de puntos podría resultar muy complicado.

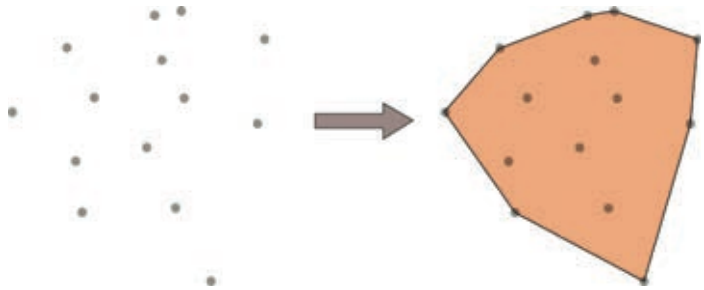


Figura 2.5: Conjunto discreto y su envoltura convexa.

Definición 2.17. Una solución eficiente es *soportada* si su imagen pertenece a la frontera de la envoltura convexa del frente de Pareto. El *frente de Pareto soportado* (*Supported Pareto Front*), $SPF \subset PF$ está formado por todas las soluciones eficientes soportadas. El *frente de Pareto no soportado* (*Non-Supported Pareto Front*) es $NSPF = PF - SPF$.

Definición 2.18. Los óptimos lexicográficos son todos aquellos puntos óptimos según la relación $\leq_{lex}$, considerando cualquier posible permutación de las funciones objetivos. Estos puntos son también llamados *puntos extremos*.

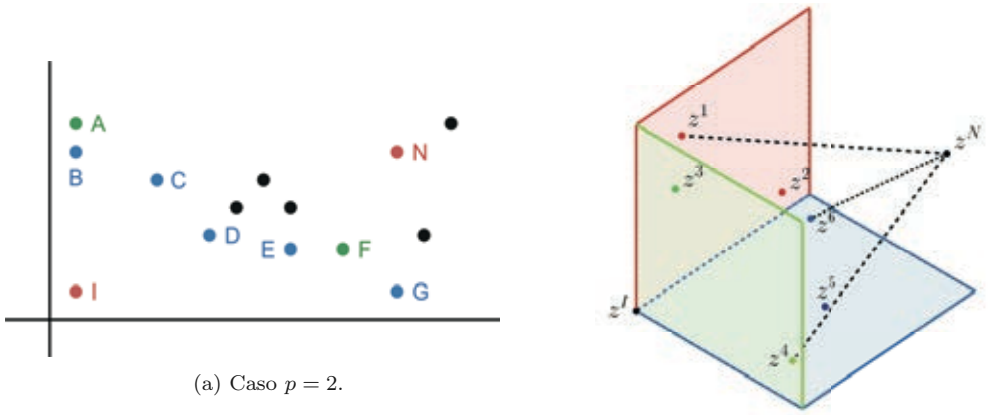
Teniendo en cuenta que existen $p!$ permutaciones de los objetivos, concluimos que en un problema MOO existe un máximo de $p!$ puntos lexicográficos óptimos (téngase en cuenta que la relación $\leq_{lex}$ permite la igualdad entre vectores, por lo que, en la práctica, algunos puntos podrían repetirse). Cualquier óptimo lexicográfico es un punto eficiente. Por otro lado, un óptimo max-orden es un punto débilmente eficiente.

Por último, definimos dos puntos que acotan inferior y superiormente al frente de Pareto.

Definición 2.19. Dado un problema MOO con frente de Pareto PF, definimos el *punto ideal* como $z^I = (z_1^I, z_2^I, \ldots, z_p^I)$, donde $z_i^I = \min_{x \in \mathcal{X}} f_i(x) = \min_{z \in PF} z_i, \quad \forall i = 1, \ldots, p$. El *punto nadir* es $z^N = (z_1^N, z_2^N, \ldots, z_p^N)$, y verifica $z_i^N = \max_{x \in \mathcal{X}_E} f_i(x) = \max_{z \in PF} z_i, \ \forall i = 1, \ldots, p$.

UNIVERSIDAD DE MÁLAGA

De las definiciones de punto ideal y punto nadir se deduce que ambos puntos son una cota inferior y superior, respectivamente, del frente de Pareto. Si el frente es conocido, calcular ambos puntos es muy sencillo. Si se desea calcular estos puntos sin conocer el frente, necesitamos resolver p problemas monoobjetivo para calcular el punto ideal (cada uno de ellos proporcionará una coordenada). Sin embargo, el cálculo del punto nadir implica la optimización sobre el conjunto eficiente $\mathcal{X}_E$, y su obtención, en general, es un problema $\mathcal{NP}$-*Hard* para el caso $p > 2$ (véase Ehrgott y Tenfelde-Podehl [64]).



(a) Caso $p = 2$.

(b) Caso $p = 3$.

FIGURA 2.6: Puntos dominados, eficientes, débilmente eficientes, ideal, nadir y óptimos lexicográficos.

En la Figura 2.6a se muestra un ejemplo para el caso de optimización biobjetivo. Suponemos que el problema es de minimización. Los puntos que no están etiquetados corresponden a puntos dominados (ambas coordenadas son dominadas por otro punto), por tanto, no forman parte del frente de Pareto. Los puntos A y F son puntos débilmente dominados (pueden ser mejorados en una coordenada), y tampoco pertenecen al frente de Pareto. De hecho, B domina débilmente a A, y E domina débilmente a F. El frente de Pareto es $PF = \{\text{B,C,D,E,G}\}$. El frente soportado es $SPF = \{\text{B,D,G}\}$ y el frente no soportado es $NSPF = \{\text{C,E}\}$. Los puntos óptimos lexicográficos son B y G, y los puntos nadir e ideal (que no son puntos del frente de Pareto) vienen representados por $z^N = \text{N}$ y $z^I = \text{I}$. Obsérvese que para el caso $p = 2$ siempre habrá dos puntos óptimos lexicográficos que formarán una caja (rectángulo) cuyos vértices opuestos son dichos puntos, y los otros vértices son el punto nadir e ideal. En el caso de que $z^N = z^I$, entonces el espacio objetivo estaría formado por un único punto eficiente, que además sería el único óptimo lexicográfico.

En la Figura 2.6b se ha querido mostrar un caso triobjetivo con seis puntos eficientes, que además son extremos. Para una mejor comprensión de por qué pueden existir 6 óptimos lexicográficos en el caso $p = 3$, hemos agrupado dichos puntos por parejas. Cada pareja $\{z^1, z^2\}$, $\{z^3, z^4\}$ y $\{z^5, z^6\}$ comparte el valor numérico de una de sus coordenadas, y es por ello que las dos coordenadas restantes dan libertad para generar dos nuevos puntos eficientes. En general, para el caso de p objetivos, si fijamos una coordenada, hay libertad para permutar las $p-1$ restantes, por ello existirá un máximo de $p!$ puntos extremos. Los hiperplanos en $\mathbb{R}^3$ son en realidad planos y se han representado aquellos que contienen a cada pareja e intersecan en el punto ideal. Se ha representado también el punto nadir, tomando los valores máximos para cada coordenada en el frente de Pareto. En este caso, los seis puntos extremos formarían también el conjunto *SPF*.

Como se ha indicado anteriormente, el cálculo del punto nadir cuando el conjunto PF no es conocido a priori, podría resultar muy costoso, hablando en términos de tiempo computacional, especialmente para el caso $p > 2$. En ocasiones necesitamos conocer una cota superior del frente de Pareto. Una posible forma de calcular dicha cota sería $\overline{z}^N = (\overline{z}_1^N, \overline{z}_2^N, \ldots, \overline{z}_p^N)$, siendo $\overline{z}_i^N = \max_{x \in \mathcal{X}} f_i(x)\ \forall i = 1, \ldots, p$. Se verifica que $z^N \leqq \overline{z}^N$, aunque estos dos puntos podrían estar a una distancia elevada, como se muestra en el Capítulo 2 de [58].

UNIVERSIDAD DE MÁLAGA

Problemas combinarios multiobjetivo

Cerramos este capítulo con la definición formal de problema combinatorio multiobjetivo y describiendo algunos ejemplos clásicos.

Definición 2.20. Un problema de la forma expresada en las Ecuaciones (2.16)–(2.17) es combinatorio multiobjetivo cuando el espacio de decisiones es discreto, finito, y la minimización se realiza en el sentido de Pareto.

Ejemplo 2.9. Problema de asignación multiobjetivo (*MOAP*, *MultiObjective Assignment Problem*).

Es una generalización del problema *AP* para p objetivos. Ahora, la función de costes estaría compuesta por una matriz para cada objetivo, por lo que se podría considerar que se trata de un *array* tridimensional del orden $p \times n \times n$, siendo n el número de agentes y de tareas. Si definimos $\forall i, j \in \{1, \ldots, n\}$ las variables $x_{ij} = 1$ si el agente i tiene asignada la tarea j; $x_{ij} = 0$ en otro caso, su formulación es la siguiente:

$$\text{mín} \quad f(x) = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} c_{1ij} x_{ij}, \sum_{i=1}^{n} \sum_{j=1}^{n} c_{2ij} x_{ij}, \ldots, \sum_{i=1}^{n} \sum_{j=1}^{n} c_{pij} x_{ij} \right), \tag{2.18}$$

$$s.a. \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, \ldots, n, \tag{2.19}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, n, \tag{2.20}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j = 1, \ldots, n. \tag{2.21}$$

Ejemplo 2.10. Problema de la mochila multiobjetivo (*MOKP*, *MultiObjective Knapsack Problem*).

Es una generalización del problema *KP*. Ahora, cada objeto de la mochila, además de tener un peso, tendrá un beneficio v_{ip} que dependerá de cada una de las p características que tendrá el objeto, por lo que su formulación sería la siguiente:

$$\text{máx} \quad f(x) = \left(\sum_{i=1}^{n} v_{1i} x_i, \sum_{i=1}^{n} v_{2i} x_i, \ldots, \sum_{i=1}^{n} v_{pi} x_i \right), \tag{2.22}$$

$$s.a. \quad \sum_{i=1}^{n} w_i x_i \leq W, \tag{2.23}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n. \tag{2.24}$$

Ejemplo 2.11. Problema de la mochila multidimensional multiobjetivo (*MOMKP*, *MultiObjective Multidimensional Knapsack Problem*).

Es una generalización del problema *MKP*. Ahora, disponemos de n objetos y m mochilas, donde cada objeto tiene un beneficio dependiente de cada una de sus p características. La formulación del modelo es la siguiente:

$$\text{máx} \quad f(x) = \left(\sum_{i=1}^{n} v_{1i} x_i, \sum_{i=1}^{n} v_{2i} x_i, \ldots, \sum_{i=1}^{n} v_{pi} x_i \right), \tag{2.25}$$

$$s.a. \quad \sum_{i=1}^{n} w_{ij} x_i \leq c_j \quad \forall j = 1, \ldots, m, \tag{2.26}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n. \tag{2.27}$$

Ejemplo 2.12. Modelo de programación lineal entera multiobjetivo genérico (MOILP, del inglés *MultiObjective Integer Linear Programming*).

Disponemos de un modelo MOILP con n variables, m restricciones y p objetivos. El vector x representa el vector de decisión y tiene sus coordenadas no negativas y enteras. Los costes de la función objetivo se pueden representar mediante una matriz C de orden $p \times n$. La matriz de

UNIVERSIDAD DE MÁLAGA

coeficientes en las restricciones se denota por $A = (a_{ij})$ y el vector de términos independientes (*right-hand side value*) se representa por $B = (b_i)$. Suponiendo que queremos maximizar, el modelo genérico sería:

$$\text{máx} \quad f(x) = \left(\sum_{i=1}^{n} c_{1i}x_i, \sum_{i=1}^{n} c_{2i}x_i, \ldots, \sum_{i=1}^{n} c_{pi}x_i \right), \tag{2.28}$$

$$s.a. \quad \sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad \forall i = 1, \ldots, m, \tag{2.29}$$

$$x_i \geq 0 \text{ y entero} \quad \forall i = 1, \ldots, n. \tag{2.30}$$

Ejemplo 2.13. Problema multiobjetivo del recubrimiento mínimo (*MOSCP*, *MultiObjective Set Covering Problem*).

Es una generalización del *SCP* para el caso de varios objetivos. Dado un universo U con n elementos, y una colección de m subconjuntos de U, denotada por S, donde cada elemento de S tiene asociado un coste positivo (no se permite que contenga al subconjunto vacío) para cada una de las p funciones objetivo, el problema multiobjetivo del recubrimiento mínimo se puede formular de la siguiente manera:

$$\text{mín} \quad f(x) = \left(\sum_{j=1}^{m} c_j^1 x_j, \sum_{j=1}^{m} c_j^2 x_j, \ldots, \sum_{j=1}^{m} c_j^p x_j \right), \tag{2.31}$$

$$s.a. \quad \sum_{j=1}^{m} a_{ji}x_j \geq 1 \quad \forall i = 1, \ldots, n, \tag{2.32}$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, m, \tag{2.33}$$

donde x_j toma el valor 1 si el subconjunto j de S es finalmente seleccionado, y 0 en otro caso. El valor c_j representa el coste del elemento j en S. El valor a_{ji} vale 1 si y solo si el subconjunto j de S contiene al elemento i de U, y vale 0 en otro caso. Finalmente, c_j^k es el coste asociado al elemento j de U para la función objetivo k.

UNIVERSIDAD DE MÁLAGA

Capítulo 3

Técnicas de resolución de problemas multiobjetivo

Tras la revisión general de los conceptos básicos en optimización combinatoria multiobjetivo, en este capítulo haremos un breve repaso sobre las distintas técnicas de resolución para este tipo de problemas. Cabe destacar que existen tres tipos de algoritmos que resuelven los problemas de optimización combinatoria multiobjetivo (en inglés *MultiObjective Combinatorial Optimization*, MOCO). Diferenciaremos cada uno de ellos en una sección diferente. En primer lugar hablaremos de los métodos exactos. Las técnicas de escalarización son las más comunes a la hora de resolver este tipo de problemas. Mostraremos un algoritmo genérico capaz de calcular el frente de Pareto, y que es independiente de la técnica de escalarización utilizada. En segundo lugar haremos un muy breve repaso sobre las principales técnicas metaheurísticas, para continuar con las metaheurísticas híbridas, que combinan métodos exactos con metaheurísticas. Las técnicas híbridas han sido objeto de un importante auge en el campo de la investigación operativa en los últimos años, ya que aprovechan las ventajas de los métodos exactos y los aproximados y mitigan algunas de las desventajas que tienen si los consideramos individualmente. Finalmente incluimos una sección en la que mostramos cómo evaluar las soluciones obtenidas tras la ejecución de un algoritmo multiobjetivo. Es claro que de alguna manera se hará necesario cuantificar dichas soluciones. De esto se encargan los indicadores de calidad. Los métodos de comparación modelan teóricamente el uso de indicadores de calidad, y serán brevemente descritos para concluir que ninguna métrica unaria será lo suficientemente buena a la hora de describir la calidad del conjunto de soluciones. Los algoritmos *anytime* son aquellos que proporcionan un buen conjunto de soluciones, bien disperso por el espacio objetivo, y esto lo hacen en cualquier momento durante su ejecución. Claramente esta es una característica deseada para cualquier algoritmo que resuelva problemas de optimización multiobjetivo, máxime cuando dicho problema sea tan duro que no seamos capaces de calcular todas las soluciones en un tiempo razonable. Cerraremos la última sección con un apartado dedicado a los contrastes de hipótesis no paramétricos, que nos ayudan confirmar estadísticamente (para un nivel de significación α) si un algoritmo tiene mejor rendimiento que otro, basándonos en los indicadores de calidad.

3.1. Métodos exactos

La mayoría de los métodos de resolución exacta para problemas multiobjetivo usan *escalarización*. Esta técnica consiste en transformar la función objetivo del espacio $\mathbb{R}^p$ en una nueva función de un solo objetivo (monoobjetivo). La escalarización es una función $\Omega : \mathbb{R}^p \longrightarrow \mathbb{R}$ que normalmente tiene en cuenta a todas las funciones objetivo, aunque veremos que esto no tiene siempre por qué ser así, ya que la información proporcionada por las funciones objetivo también puede ir incorporada al nuevo problema en forma de restricción. En cualquier caso, la meta final es realizar la transformación para poder revolver un problema con un único objetivo. El resolutor se encargará de aplicar el método simplex o cualquier técnica de resolución de problemas ILP. Una vez resuelto, lo ideal sería que la solución óptima del problema monoobjetivo sea además una solución eficiente (o al menos débilmente eficiente) en el problema multiobjetivo. De no ser así, dicha solución en el espacio objetivo sería dominada y la escalarización no tendría ningún sentido práctico.

UNIVERSIDAD DE MÁLAGA

Esta sección se divide en cuatro partes. Comenzaremos revisando las técnicas de escalarización más usadas a la hora de resolver problemas combinatorios multiobjetivo. Cada uno de estos tipos se ilustrará mediante un ejemplo gráfico en dos dimensiones. En la segunda parte mostraremos una formulación genérica que contendrá a todas las técnicas explicadas (y algunas otras más) en la primera parte. A continuación mostraremos un algoritmo genérico capaz de calcular el frente de Pareto en un problema MOILP (en particular, un problema combinatorio multiobjetivo). Este algoritmo se presenta en este capítulo ya que será utilizado como referencia en los distintos capítulos que componen la parte de propuestas algorítmicas. Finalizaremos la sección con un breve comentario sobre el resolutor CPLEX, indicando sus parámetros y funciones más usuales, que también serán descritos en capítulos posteriores.

3.1.1. Técnicas de escalarización

La escalarización puede ser aplicada en cualquier problema multiobjetivo, en particular, en un problema combinatorio multiobjetivo. Partiendo del modelo dado por las Ecuaciones (2.16)–(2.17), definimos a continuación algunas de las escalarizaciones más comunes, indicando qué tipo de solución es para el problema multiobjetivo el óptimo de dicha escalarización. Tomaremos como referencia práctica el siguiente ejemplo ilustrativo.

Ejemplo 3.1. Supongamos un problema biobjetivo en cuyo espacio objetivo solo hay cuatro puntos no dominados y dos débilmente dominados (ver Figura 3.1). Los puntos vienen dados por las coordenadas: $z^1 = (4, 13)$, $z^2 = (8, 10)$, $z^3 = (14, 8)$, $z^4 = (20, 4)$, $a^2 = (8, 11)$ y $a^3 = (17, 8)$.

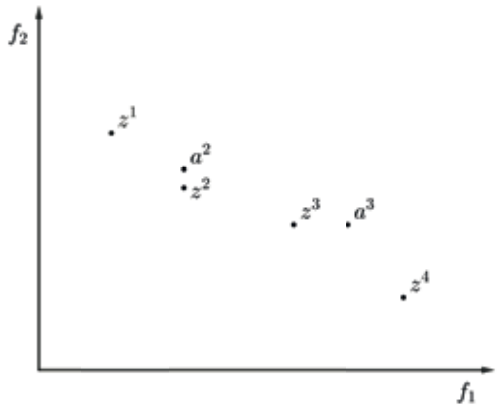


Figura 3.1: Ejemplo de espacio objetivo para $p = 2$.

Suma ponderada (*Weighted Sum*)

A la hora de optimizar varias funciones objetivo, intuitivamente parece razonable optimizar una media ponderada de cada una de ellas. En esto consiste el método de la suma ponderada. Cada función objetivo tendrá un peso o importancia y la distribución de los pesos podría influir en la solución final obtenida. Así, el modelo de escalarización viene dado por

$$\begin{aligned} \text{mín} \quad & \sum_{i=1}^{p} \lambda_i f_i(x), \\ s.a. \quad & x \in \mathcal{X}, \end{aligned} \tag{3.1}$$

donde λ_i representa el peso del objetivo i en la escalarización. Siempre consideramos $\lambda_i \geq 0$, $\forall i = 1 \dots p$. En ocasiones es necesario o deseable normalizar la expresión. En esos casos los pesos se definen de manera que $\sum_{i=1}^{p} \lambda_i = 1$.

Proposición 3.1. Sea $\hat{x}$ una solución óptima del modelo escalarizado de la Ecuación (3.1). Consideremos el vector de pesos $\lambda = (\lambda_1, \dots, \lambda_p)$. Si $\lambda \geq 0$, la solución $\hat{x}$ es débilmente eficiente, y si $\lambda > 0$, la solución $\hat{x}$ es eficiente.

UNIVERSIDAD DE MÁLAGA

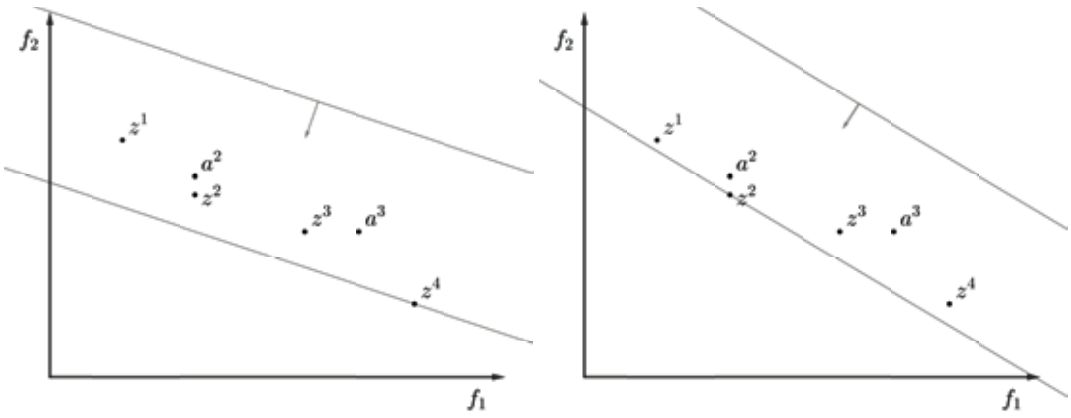
Demostración. Ver página 71 de [58]. □

Queda claro tras esta proposición la necesidad de considerar pesos positivos si queremos garantizar que la solución obtenida sea eficiente (y su imagen, por tanto, sea un punto del frente de Pareto). De hecho, distintas parametrizaciones podrían dar lugar a distintos puntos eficientes (aunque algunas veces habrá repeticiones). Cabe ahora preguntarse por el recíproco: dada una solución eficiente, $\hat{x}$, ¿existe una parametrización que tenga a $\hat{x}$ como valor óptimo? Recordemos en este punto que estamos interesados en obtener el frente de Pareto y una antiimagen para cada punto de dicho frente, por tanto, cuando decimos si existe una parametrización que tiene a $\hat{x}$ como valor óptimo, en realidad queremos decir que existe un vector $\overline{x}$ tal que $f(\overline{x}) = f(\hat{x})$, es decir, nos importa más el punto del frente que el punto eficiente en el espacio de decisiones. La respuesta a lo anterior es negativa, y esto es así debido a que pueden existir soluciones en el espacio objetivo que no están en la frontera de la envoltura convexa (ver Definición 2.16). Lo que sí podemos garantizar es que las soluciones obtenidas por el método de suma ponderada son siempre soportadas, y recíprocamente, cada solución del frente de Pareto soportado, *SPF*, se puede obtener mediante una combinación apropiada de los pesos en la suma ponderada (ver [60] y [35]). De hecho, en ocasiones se define el concepto de solución soportada de manera equivalente a la dada en la Definición 2.17:

Definición 3.1. Una solución $x \in \mathcal{X}$ es soportada si minimiza una suma ponderada de sus objetivos con todos los pesos positivos.

Dado que la ecuación $\sum_{i=1}^{p} \lambda_i f_i(x)$ es la de un hiperplano en el espacio objetivo, el vector de pesos, λ, representa un vector normal asociado a dicho hiperplano. Veamos esto último mediante un ejemplo sencillo para el caso $p = 2$. En la función objetivo escalarizada $\lambda_1 f_1(x) + \lambda_2 f_2(x)$ puede considerarse que ambos pesos sumen 1. Así, tendríamos que minimizar $\lambda f_1(x) + (1 - \lambda) f_2(x)$, siendo $0 < \lambda < 1$. El vector $(\lambda, 1 - \lambda)$ es el vector normal asociado a la recta del plano definida por la función objetivo. En función de los valores de λ, siguiendo la trayectoria de la normal a dicha recta, podremos obtener distintos puntos eficientes, pero siempre soportados. Para el caso $p = 3$ visualizar esto gráficamente es más complicado (aún más cuando $p > 3$), aunque la idea subyacente es la misma: variando los valores del vector λ podemos ir obteniendo distintos puntos del frente soportado de Pareto.

Si en el Ejemplo 3.1 parametrizamos la función objetivo con el vector de pesos $w = (1, 3)$ obtendríamos como punto no dominado el z^4. Si por el contrario la escalarización fuese dada con el vector $w = (3, 5)$, entonces obtendríamos el punto z^2 (ver Figuras 3.2a y 3.2b). Obsérvese cómo la variación de pesos origina distintas soluciones, pero sus imágenes siempre serán puntos del frente de Pareto soportado (*SPF*). Por ejemplo, nunca podremos encontrar el punto z^3 utilizando este método.



(a) Suma ponderada $(\lambda_1, \lambda_2) = (1, 3)$. (b) Suma ponderada $(\lambda_1, \lambda_2) = (3, 5)$.

Figura 3.2: Escalarización mediante suma ponderada.

UNIVERSIDAD DE MÁLAGA

Epsilon constraint (ε-*constraint*)

El método ε-constraint (o también *epsilon-constraint*) es probablemente la técnica más conocida para resolver problemas multiobjetivo. Consiste en minimizar únicamente una de sus funciones objetivo, y transformar el resto en restricciones, siendo acotadas cada una de ellas por un cierto valor. Inicialmente fue introducido por Haimes et al. [84] para el caso biobjetivo, para ser posteriormente generalizado a cualquier número de objetivos (véase por ejemplo Chankong y Haimes [20]). Su formulación viene dada por

$$\begin{aligned} \text{mín} \quad & f_i(x), \\ s.a. \quad & f_j(x) \leq \varepsilon_j \quad \forall j \neq i, \\ & x \in \mathcal{X}. \end{aligned} \tag{3.2}$$

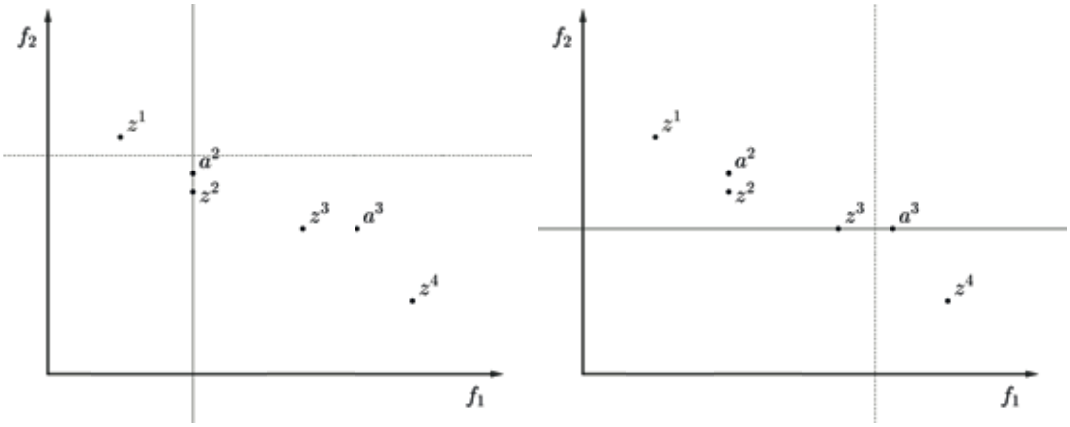
Existen p formas distintas de formular este modelo. Elegimos el objetivo $i \in \{1, \ldots, p\}$ a minimizar, y los restantes son añadidos como restricciones, siendo $\varepsilon \in \mathbb{R}^p$ el vector de acotación. Obsérvese que el vector ε tiene $p-1$ coordenadas útiles, siendo indiferente el valor que tome ε_i.

Proposición 3.2. Sea $\hat{x}$ una solución óptima del modelo de la Ecuación (3.2). Entonces $\hat{x}$ es débilmente eficiente.

Demostración. Ver página 99 de [58]. □

Si el óptimo del modelo dado por la Ecuación (3.2) es único, sí podríamos garantizar que la solución obtenida es eficiente (página 100 de [58]). Sin embargo, en la práctica, el resolutor finaliza su ejecución cuando encuentra una solución óptima, y podría ser muy costoso, en general, realizar una búsqueda de óptimos alternativos, si los hubiere. Para evitar la desventaja de que la solución obtenida por este método es débilmente eficiente, se puede recurrir a otra variante del método ε-constraint en la que sí se puede garantizar que la solución es eficiente, pero para ello se necesitarían resolver dos modelos matemáticos. La idea de esta variante puede verse en [10] y también será descrita en el Capítulo 7 de esta tesis.

Volviendo al Ejemplo 3.1, aplicamos el método ε-constraint usando sus dos variantes. Si minimizamos f_1 y ponemos como cota $\varepsilon_2 = 12$ para el segundo objetivo (ver Figura 3.3a), podemos observar que el resolutor podría proporcionarnos cualquiera de las soluciones a_2 o z_2. Es por ello que solo podemos garantizar que la solución obtenida es débilmente eficiente. Sin embargo, si minimizamos f_2 sujeto a la cota para el otro objetivo dada por $\varepsilon_1 = 16$ (Figura 3.3b), obtendríamos como solución el punto z^3, que sí es eficiente en este caso (aunque tampoco podríamos garantizarlo a no ser que comprobásemos que el resolutor nos confirme que el óptimo es único).



(a) ε-constraint minimizando f_1 con $\varepsilon_2 = 12$. (b) ε-constraint minimizando f_2 con $\varepsilon_1 = 16$.

Figura 3.3: Escalarización mediante el método ε-constraint.

UNIVERSIDAD DE MÁLAGA

Método híbrido

Este método combina los dos métodos expuestos anteriormente. Por un lado, utiliza como escalarización una suma ponderada de todos los objetivos y, además, añade restricciones asociadas a cada función objetivo. Fue propuesto por Guddat et al. [82] y aplicado por primera vez para el caso biobjetivo por Chalmet et al. [19]. Su formulación viene dada por

$$\begin{aligned} \text{mín} \quad & \sum_{i=1}^{p} \lambda_i f_i(x), \\ s.a. \quad & f_i(x) \leq \varepsilon_i \quad \forall i = 1, \ldots, p, \\ & x \in \mathcal{X}. \end{aligned} \tag{3.3}$$

Obsérvese la diferencia entre el método ε-constraint y el método híbrido. Mientras el anterior añadía $p-1$ restricciones, este método añade p nuevas restricciones, además de realizar una suma ponderada para escalarizar la función objetivo.

Proposición 3.3. Una solución factible $x \in \mathcal{X}$ del modelo dado por la Ecuación (3.3) con vector de pesos $\lambda > 0$, es una solución óptima si y solo si el punto x es eficiente.

Demostración. Ver Guddat et al. [82]. □

El método híbrido es aplicado en Domínguez et al. [49] y una versión *anytime* del mismo se comentará en el Capítulo 7. Aplicando esta técnica al Ejemplo 3.1, y considerando una escalarización con pesos $w = (1, 4)$ y vector de acotación $\varepsilon = (19, 16)$ observamos que el método híbrido es capaz de obtener incluso soluciones eficientes no soportadas, en este caso el punto no dominado z^3 (ver Figura 3.4).

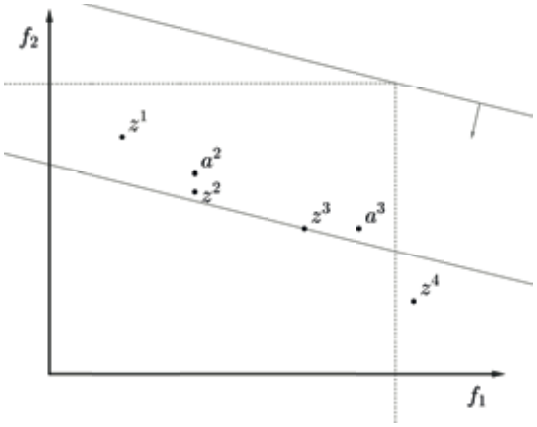


FIGURA 3.4: Método híbrido usando $w = (1, 4)$ y $\varepsilon = (19, 16)$.

Método de Tchebycheff ponderado (*Weighted Tchebycheff method*)

Este método y el siguiente se engloban en los llamados *métodos del punto de referencia* (ver Andrzej P. Wierzbicki [178]). Teniendo en cuenta que en un problema multiobjetivo lo deseable sería obtener el punto ideal, z^I, y que en la práctica esto no suele ocurrir ya que los objetivos entran en conflicto entre sí y no se pueden minimizar todos a la vez, se suele tomar este punto como referencia, de manera que se obtengan soluciones tan cerca de este punto ideal como sea posible. También es posible considerar como referencia una cota inferior del punto ideal, llamado *punto utópico*. Esta idea de 'estar cerca' de un punto, implica necesariamente la consideración de una función distancia. Generalmente se utiliza como función distancia alguna de las generadas por las normas usuales, como $\|\cdot\|_1$ o $\|\cdot\|_\infty$[1].

El método de Tchebycheff ponderado consiste en minimizar una función objetivo que usa una ponderación de la métrica $\|\cdot\|_\infty$. Se define $\|x\|_\infty^w = \max_{i=1,\ldots,p}(w_i|x_i|)$, siendo $w \in \mathbb{R}^p$ el

[1] Dados $x = (x_1, \ldots, x_p)$ e $y = (y_1, \ldots, y_p)$, tenemos que $d_1(x, y) = \sum_{i=1}^{p} |x_i - y_i|$. Por otro lado, $d_\infty(x, y) = \max_{i=1}^{p} |x_i - y_i|$.

UNIVERSIDAD DE MÁLAGA

vector de pesos, con $w_i > 0$ y $\sum_{i=1}^{p} w_i = 1$. Fue introducido por Steuer y Choo [163] y toma como referencia el punto ideal:

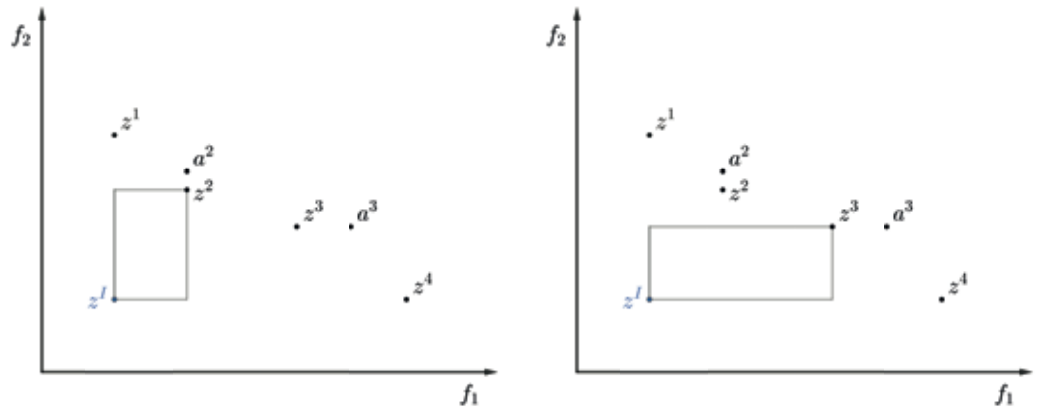
$$\begin{aligned} \text{mín} \quad & \left\| f(x) - z^I \right\|_\infty^w , \\ s.a. \quad & x \in \mathcal{X}. \end{aligned} \tag{3.4}$$

Una revisión de los métodos interactivos, entre los que se encuentran los basados en puntos de referencia, puede verse en Alves y Clímaco [2].

Proposición 3.4. Sea $\hat{x}$ una solución óptima del modelo de la Ecuación (3.4). Entonces $\hat{x}$ es débilmente eficiente.

Demostración. Ver Steuer y Choo [163]. □

Las líneas de contorno[2] definidas por la norma $\|\cdot\|_\infty$ para el caso $p = 2$ representan cuadrados en el espacio objetivo. A la hora de asignar pesos a cada dimensión, dichos cuadrados se convierten en rectángulos. Mostramos en las Figuras 3.5a y 3.5b una parte del contorno generado por la métrica de Tchebycheff (en realidad serían rectángulos centrados en z^I pero solo mostramos un cuadrante del mismo). En ambas figuras hemos aplicado la métrica de Tchebycheff al Ejemplo 3.1, con vectores de pesos $w = \left(\frac{2}{5}, \frac{3}{5}\right)$ y $w = \left(\frac{5}{7}, \frac{2}{7}\right)$, respectivamente.



(a) Método de Tchebycheff con $w = \left(\frac{2}{5}, \frac{3}{5}\right)$. (b) Método de Tchebycheff con $w = \left(\frac{5}{7}, \frac{2}{7}\right)$.

Figura 3.5: Escalarización mediante el método de Tchebycheff.

Método aumentado de Tchebycheff ponderado (*Augmented weighted Tchebycheff method*)

Presentamos un último método de escalarización, basado en puntos de referencia, y capaz de calcular soluciones eficientes. Basándonos en el caso anterior, definimos una nueva métrica que combinará las normas $\|\cdot\|_\infty^w$ y $\|\cdot\|_1$. Minimizando esta nueva norma, podremos garantizar la eficiencia de las soluciones. Dado $\gamma > 0$ y $w \in \mathbb{R}^p$ con $w_i > 0$ y $\sum_{i=1}^{p} w_i = 1$, se define la norma aumentada como $\|\cdot\|_\gamma^w = \|\cdot\|_\infty^w + \gamma\|\cdot\|_1$ y el modelo a resolver es

$$\begin{aligned} \text{mín} \quad & \left\| f(x) - z^I \right\|_\gamma^w , \\ s.a. \quad & x \in \mathcal{X}. \end{aligned} \tag{3.5}$$

Proposición 3.5. Sea $\hat{x}$ una solución óptima del modelo de la Ecuación (3.5). Entonces $\hat{x}$ es eficiente.

Demostración. Ver Steuer y Choo [163]. □

[2]Conjunto de puntos que hace constante la expresión de la escalarización.

UNIVERSIDAD DE MÁLAGA

Podemos observar que tanto en este método como en el anterior, aunque la función objetivo está escalarizada, la expresión en ambas es no lineal. Por tanto, si queremos resolver alguno de ellos usando un resolutor necesitaríamos previamente linealizar la función objetivo, lo que exigirá necesariamente añadir nuevas restricciones. Una pequeña variante de este método se usa en el trabajo de Holzmann y Smith [91] y también en el de Domínguez et al. [50]. Lo trataremos con algo más de detalle en el Capítulo 4.

En la Figura 3.6 podemos observar como la parte visible de la línea de contorno en el método aumentado de Tchebycheff es una linea poligonal (lineal a trozos y continua). Aplicándolo al Ejemplo 3.1 obtendríamos como punto no dominado el z^3. Para ampliar la información sobre este método puede consultarse el trabajo de Dächert et al. [35].

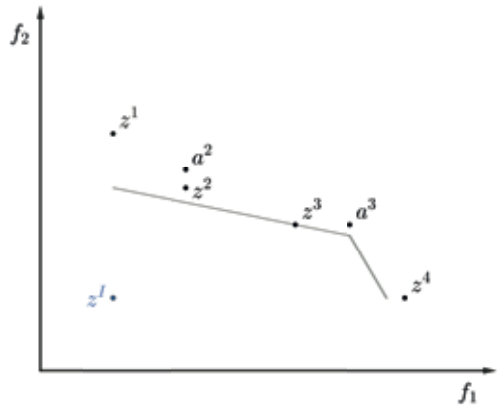


Figura 3.6: Escalarización usando el método aumentado de Tchebycheff.

Resumiendo lo expuesto en este apartado, hemos visto distintas técnicas que permiten obtener puntos (débilmente) eficientes. Algunas de ellas están basadas en la adición de restricciones del tipo $\leq$ en las que se permite acotar el espacio de búsqueda, manteniendo la garantía de que la solución obtenida será eficiente. Llegados a este punto, cabría preguntarse si sería posible acotar el espacio de búsqueda mediante restricciones del tipo $\geq$ (en problemas de minimización). La respuesta a esta cuestión es negativa. Basándonos en el Ejemplo 3.1, imaginemos que queremos minimizar cualquier suma ponderada de las funciones objetivo, sujeto a las restricciones $f_1(x) \geq 15$ y $f_2(x) \geq 5$. En ese caso obtendríamos como solución óptima el punto a^3, que no es eficiente. Dicho esto, concluimos que no podemos garantizar nunca que la solución obtenida es eficiente si se añaden restricciones de tipo $\geq$ en un problema de minimización multiobjetivo.

3.1.2. Formulación genérica que engloba las técnicas de escalarización

Matthias Ehrgott publicó en 2006 un trabajo donde discutía las distintas técnicas de escalarización para problemas MOILP. Estableció una formulación genérica para las escalarizaciones más comunes. Reproducimos a continuación dicha formulación (adaptado de [59]). Consideremos el modelo multiobjetivo en su versión matricial,

$$\begin{aligned} \text{mín} \quad & Cx, \\ s.a. \quad & x \in \mathcal{X}, \end{aligned} \tag{3.6}$$

donde $C = (c_{ki})$ es una matriz con coeficientes racionales, de orden $p \times n$. El vector fila c_k, de dimensión n, representa para el objetivo k los coeficientes de los costes asociados a esa función objetivo. En este caso el vector de decisiones se expresa como un vector columna, para que tenga sentido el producto Cx. Dados los vectores $\rho, \lambda, \nu, \epsilon \in \mathbb{R}^p$, siendo $\lambda, \nu \geqq 0$ vectores de

UNIVERSIDAD DE MÁLAGA

pesos, y ϵ un vector de términos independientes, la formulación genérica del modelo es

$$\begin{aligned} \text{mín} \quad & \left(\max_{k=1}^{p} \{\nu_k(c_k x - \rho_k)\} + \sum_{k=1}^{p} \lambda_k (c_k x - \rho_k) \right), \\ s.a. \quad & c_k x \leq \epsilon_k \quad \forall k = 1, \ldots, p, \\ & x \in \mathcal{X}. \end{aligned} \tag{3.7}$$

Todas las escalarizaciones propuestas en este Capítulo pueden considerarse casos particulares de la expresión general dada por la Ecuación (3.7), como puede observarse en la Tabla 3.1.

Tabla 3.1: Parámetros asociados a los distintos métodos de escalarización. Cuando es indiferente el valor del parámetro ρ, se indica medinte un guión.

Método	ρ	ν	λ	ϵ
Suma ponderada	-	$(0, \ldots, 0)$	$\lambda > 0$	$(\infty, \ldots, \infty)$
ε-constraint	-	$(0, \ldots, 0)$	$\lambda_i = 1$	$\epsilon_i = \infty$
			$\lambda_j = 0 \ \forall i \neq j$	$\epsilon_j \in \mathbb{R} \ \forall j \neq i$
Híbrido	-	$(0, \ldots, 0)$	$\lambda > 0$	$\epsilon \in \mathbb{R}^p$
Tchebycheff	z^I	$w > 0$	$(0, \ldots, 0)$	$(\infty, \ldots, \infty)$
Tchebycheff aumentado	z^I	$w > 0$	$(\gamma, \ldots, \gamma)$	$(\infty, \ldots, \infty)$

Nótese que $z_k^I \leq c_k x \ \ \forall k = 1, \ldots, p$, ya que cada coordenada del vector ideal toma el valor mínimo posible en comparación con los puntos del espacio objetivo. Es por ello que para los casos del método de Tchebycheff y el de Tchebycheff aumentado, se cumple que $|c_k x - z_k^I| = c_k x - z_k^I$ y los modelos también quedan generalizados por la Ecuacion 3.7.

3.1.3. Algoritmo genérico que calcula el frente de Pareto completo

Una vez explicadas distintas técnicas que calulan puntos eficientes, el siguiente asunto que nos ocupa es el de diseñar un algoritmo capaz de obtener el frente de Pareto completo. Existe en la literatura una gran cantidad de trabajos dedicados a este fin. Una revisión bibliográfica de los mismos se verá en el Capítulo 4. Por el momento, mostramos en este apartado un algoritmo genérico que calcula el frente de Pareto para cualquier problema multiobjetivo discreto finito (en particular, para los problemas combinatorios multiobjetivo, MOCO). Este algoritmo servirá como referencia para los tres algoritmos propuestos en los Capítulos 4 y 5.

Antes de su desarrollo, definamos lo que es una región de búsqueda, basándonos en el trabajo de Dächert et al. [37]. A dicha región le llamaremos 'caja', dado que su definición coincide con la generalización de un rectángulo a dimensiones mayores. Las cajas son hiperrectángulos en $\mathbb{R}^p$.

Definición 3.2. (**Caja**) Dados dos vectores $l, u \in \mathbb{R}^p$ con $l < u$, definimos la *caja* $[l, u]$ como:

$$[l, u] = \{x \in \mathbb{R}^p \mid l_i \leq x < u_i, \forall i = 1, \ldots, p\}. \tag{3.8}$$

Definición 3.3. Dado un problema combinatorio multiobjetivo de dimensión p, una región de búsqueda (*search region*) es un subconjunto de $\mathbb{R}^p$ donde todavía pueden existir puntos no dominados. Las regiones de búsqueda suelen estar definidas en la forma $Z = [l, u]$, es decir, mediante cajas.

Cuando una región de búsqueda es explorada y finalmente no se encuentra ningún punto dominado en ella, se considera que dicha región o caja está vacía, sin contradecir con ello su definición original. Supongamos que $\overline{z}^N$ es un punto tal que $z^N < \overline{z}^N$, siento z^N el punto nadir. Obsérvese que la caja $[z^I, \overline{z}^N]$ es una región de búsqueda que contiene a todo el frente de Pareto. La consideración del punto $\overline{z}^N$ como cota estrictamente superior del punto nadir es importante, debido a la desigualdad estricta que aparece en la definición de caja (es decir, no nos valdría considerar $[z^I, z^N]$). Cuando la cota inferior de una caja es el punto ideal en un frente de Pareto, generalmente se representa dicha caja únicamente mediante su cota superior, es decir, $[z^I, u] = [u]$.

UNIVERSIDAD DE MÁLAGA

El algoritmo presentado por Dächert y Klamroth [36], cuyo pseudocódigo mostramos en el Algoritmo 1, calcula el frente de Pareto de un problema combinatorio multiobjetivo, aunque sería válido también para cualquier problema multiobjetivo discreto con un número finito de soluciones.

Imaginemos que disponemos de un conjunto de cajas, $\mathcal{U}$, las cuales vamos explorando una a una, buscando buscando puntos del frente de Pareto. La caja inicial será $[\overline{z}^N]$, una cota estrictamente superior del punto nadir, para garantizar que exploraremos todo el espacio objetivo (Línea 2).

Mientras exista alguna caja por explorar, seleccionamos una de ellas (Línea 4) y la exploramos (Línea 5). Si el problema no tiene solución, o lo que es lo mismo, la caja está vacía, simplemente la descartamos, ya que sabemos que no habrá ninguna solución en esa región de búsqueda (Línea 10). Cada vez que encontramos un punto no dominado, lo añadimos al conjunto PF, que a la salida del programa será el frente de Pareto completo. El procedimiento de actualización (Línea 8) consiste en lo siguiente: la caja explorada se parte en distintas cajas de manera que la unión de las nuevas cajas, que también se añadirán al conjunto $\mathcal{U}$, no contendrá al punto encontrado, $f(x^*)$, ni a ningún otro dominado por él. Esta forma de dividir las cajas en realidad está reduciendo el espacio de búsqueda, ya que se están descartando el último punto encontrado y todos los dominados por él (en particular, el espacio de búsqueda se reduce en un punto, al menos). Este proceso de actualización, que también es capaz de eliminar cajas redundantes (aquellas que están completamente contenidas en otra caja), se explicará con más detalle en la Sección 4.1.

El parámetro de entrada, P, corresponde al programa matemático utilizado para encontrar soluciones no dominadas. Generalmente este parámetro es fijo, en el sentido de que la escalarización utilizada es siempre la misma, si bien sería posible la hibridación de escalarizaciones durante la ejecución. Esta escalarización usada como parámetro de entrada debe ser tal que permita explorar distintas regiones de búsqueda. Por ejemplo, el método de la suma ponderada no encontrará nunca soluciones no soportadas, por lo que no sería válido en este caso si pretendemos calcular el frente de Pareto. Los métodos ε-constraint y de Tchebycheff solo pueden garantizar que las soluciones son débilmente eficientes, por lo que tampoco los consideraremos. Analicemos el caso del método híbrido. El vector de acotación en el método híbrido se puede ver como una región de búsqueda. Así, tenemos identificada una relación entre las cajas y los parámetros a considerar en el problema matemático. Éste si sería un ejemplo válido a considerar como parámetro de entrada P. Por otro lado, si consideramos el método aumentado de Tchebycheff, una vez linealizado el modelo y configurados sus parámetros internos adecuadamente, también es posible encontrar una relación de los mismos con los de las regiones de búsqueda. De hecho, este mismo método se aplica en los trabajos de Holzmann y Smith [91] y de Domínguez et al. [50], como veremos en el Capítulo 4.

Algoritmo 1 Método general para problemas MOCO

Entrada: P
Salida: PF
1: $PF = \emptyset$
2: $\mathcal{U} \leftarrow [\overline{z}^N]$
3: **mientras** $(\mathcal{U} \neq \emptyset)$ **hacer**
4: Selecciona $B \in \mathcal{U}$
5: **si** *($P(B)$ tiene solución)* **entonces**
6: $x^* \leftarrow$ Solución óptima de $P(B)$
7: $PF = PF \cup \{f(x^*)\}$
8: Actualiza $\mathcal{U}$
9: **si no**
10: $\mathcal{U} \leftarrow \mathcal{U} - \{B\}$
11: **fin si**
12: **fin mientras**

El Algoritmo 1 cumple la propiedad de que el conjunto $\mathcal{U}$ contiene regiones de búsqueda sin puntos dominados por los puntos de PF. Dado que el número de puntos en el espacio objetivo es finito, y que el algoritmo encuentra el siguiente punto no dominado en un tiempo

UNIVERSIDAD DE MÁLAGA

finito, se puede concluir que dicho algoritmo es finito y termina encontrando el frente de Pareto completo.

3.1.4. Uso de CPLEX como resolutor de problemas multiobjetivo

En este apartado vamos a comentar brevemente algunas de las funcionalidades que ofrece el resolutor CPLEX, indicando algunos de sus parámetros. CPLEX resuelve problemas de programación matemática con restricciones lineales o cuadráticas convexas, donde la función objetivo puede ser lineal o cuadrática convexa. Las variables tratadas pueden ser continuas o enteras. Posee distintas bibliotecas adaptables a distintos lenguajes de programación o plataformas. En esta tesis doctoral utilizamos la versión 12.6.2. y la API del lenguaje C para la ejecución de los algoritmos propuestos. Este producto comenzó su desarrollo en 1997 por la empresa ILOG, para pasar a formar parte de IBM a partir de 2009. Está en continuo desarrollo y es uno de los resolutores más conocidos y usados en el ámbito científico. A partir de la versión 12.9. este software tiene implementados los modelos multiobjetivo. No obstante, es sencillo definirlos en versiones anteriores a partir de los datos del problema, y mediante la escalarización deseada.

En la Tabla 3.2 mostramos algunas de las funciones y parámetros más utilizadas. Varios de los parámetros de la tabla serán referenciados en los capítulos de la Parte II – Propuestas Algorítmicas.

Tabla 3.2: Algunas funciones y parámetros en CPLEX 12.6.2.

FUNCIONES	
CPXopenCPLEX	Inicializa el entorno de CPLEX. Es la primera rutina a llamar
CPXcreateprob	Crea un objeto problema (modelo matemático). Puede haber definidos varios en un mismo entorno CPLEX
CPXchgprobtype	Cambia el tipo de problema (lineal, entero, etc.)
CPXXchgctype	Cambia el tipo de la variable (continua, binaria, entera, etc.)
CPXreadcopyprob	Lee un modelo matemático desde archivo y lo carga en un objeto problema
CPXwriteprob	Escribe un problema en un archivo
CPXaddcols	Añade variables al modelo
CPXaddrows	Añade restricciones al modelo
CPXchgobjsen	Cambia el sentido de la función objetivo (máx / mín)
CPXchgobj	Cambia coeficientes de las variables en la función objetivo
CPXchgcoef	Cambia valor de los coeficientes de las variables en la matriz de restricciones
CPXchgrhs	Cambia valores en los términos independientes
CPXchgobjoffset	Añade/Modifica un coeficiente constante a la función objetivo
CPXmipopt	Resuelve el problema de optimización entera
CPXgetobjval	Obtiene el valor objetivo tras resolver el problema
CPXgetstat	Obtiene el tipo de solución obtenida
CPXgetx	Obtiene el vector de decisión óptimo
PARÁMETROS	
CPX_PARAM_PARALLELMODE	Determina el modo de optimización en paralelo usado por CPLEX
CPX_PARAM_THREADS	Numero máximo de hebras a ejecutar en paralelo
CPX_PARAM_MIPEMPHASIS	Controla balance entre priorizar factibilidad u optimalidad
CPX_PARAM_INTSOLLIM	Número de soluciones enteras a encontrar antes de parar la ejecución
CPX_PARAM_EPGAP	Tolerancia relativa en el *gap* entre el mejor valor objetivo y el objetivo del mejor nodo de los restantes a explorar en el árbol de búsqueda en el *Branch & Cut*
CPX_PARAM_EPAGAP	Tolerancia absoluta del *gap* entre el mejor valor objetivo y el objetivo del mejor nodo restante
CPX_PARAM_EPINT	Tolerancia de integralidad en el valor obtenido para las variables enteras
CPX_PARAM_TILIM	Tiempo límite (en segundos) empleado por el resolutor

UNIVERSIDAD DE MÁLAGA

CPLEX no es un resolutor puramente exacto, en el sentido de que durante su ejecución, cuando se alcanza una cota de error suficientemente pequeña como para considerar que la solución obtenida es la óptima, el resolutor da por finalizada su búsqueda del óptimo. Esto quiere decir por ejemplo que, en función de la configuración de los parámetros, CPLEX podría determinar que una variable que tiene valor 2.999999999999999 finalmente tendrá el valor 3.

Existen interesantes trabajos que desarrollan resolutores exactos para resolver problemas de programación lineal, pero que en la práctica no han resultado ser competitivos debido al enorme coste computacional que emplean en cuanto a tiempo. Para más información sobre estos resolutores pueden verse los trabajos de Applegate et al. [5] y de Cook et al. [31].

3.2. Métodos aproximados. Metaheurísticas

Cuando no estamos interesados en obtener una solución óptima, o cuando el tiempo empleado para obtener el óptimo pudiera ser muy elevado, es común recurrir a métodos aproximados. En estos casos, pretendemos encontrar una solución en un tiempo razonable y que sea 'buena' en el sentido de estar próxima al óptimo.

Las *heurísticas ad hoc* son algoritmos aproximados que dependen en gran medida del problema concreto para el que se han diseñado. En este caso, la técnica o idea aplicada a la resolución del problema es específica del mismo, y aunque podría ser trasladada a otros problemas, generalmente deben particularizarse para cada caso concreto. No existe una clasificación única para estos métodos. Una posible clasificación dividiría las heurísticas *ad hoc* en tres grandes grupos: los métodos de descomposición, los métodos constructivos y los métodos de búsqueda local[3].

En los métodos de descomposición el problema original se descompone en subproblemas más sencillos de resolver. En los métodos constructivos se van añadiendo componentes a la solución, paso a paso, hasta obtener una solución completa. En los métodos de búsqueda local, partimos de una solución completa y la mejoramos progresivamente. Para ello, el algoritmo realiza sucesivos movimientos con la condición de trasladarse a una solución con mejor valor objetivo (o *fitness*). La iteración finaliza cuando no existe ningún movimiento que mejore la mejor solución encontrada hasta el momento. Realizar un movimiento se conoce también con el término de explorar el *vecindario*. Un vecindario de una solución x es un conjunto de soluciones que se puede construir a partir de x aplicando un operador específico de modificación. Los óptimos en este tipo de métodos son llamados *óptimos locales*.

Los procedimientos *metaheurísticos* se sitúan conceptualmente 'por encima' de los heurísticos, en el sentido en que guían el diseño de éstos. La idea es combinar diferentes métodos heurísticos en un nivel más alto y explorar el espacio de búsqueda de una forma más eficiente, bien descartando regiones con poca garantía de encontrar el óptimo, o bien dirigiéndonos hacia regiones más prometedoras. El término metaheurística fue introducido por Glover [75]. Existen varias formas de clasificar las metaheurísticas. Si nos centramos en la clasificación basada en el número de soluciones manipuladas en cada iteración del algoritmo (Blum y Roli [14]), podemos dividirlas en dos grandes grupos: las que se basan en *trayectoria* y las que se basan en *población*. Sin describir todos los algoritmos metaheurísticos que existen, destacamos algunos de los más relevantes en el dominio.

Las metaheurísticas basadas en trayectoria parten de una solución y, tras explorar su vecindario, la actualizan. Las más conocidas dentro de este grupo son el *enfriamiento simulado*, la *búsqueda tabú*, la *búsqueda miope aleatorizada y adaptativa*, la *búsqueda con vecindario variable* y la *búsqueda local iterada*.

El enfriamiento simulado (*Simulated Annealing*, SA) simula el proceso de enfriamiento del metal y del cristal. Se permite elegir una solución con peor *fitness* que la actual para posteriormente explorar el vecindario. De esta forma, si la solución ha mejorado, hemos conseguido escapar del óptimo local en búsqueda de otras regiones posiblemente mejores (ver Kirkpatrick et al. [101]).

La búsqueda tabú (*Tabu Search*, TS) utiliza una especie de memoria a corto plazo, llamada *lista tabú*, para poder escapar de los mínimos locales, evitando así explorar repetidas veces la misma región. La lista tabú guarda las soluciones visitadas recientemente y no permite que

[3]Descripción de los algoritmos aproximados extraída de [23] y de [127].

UNIVERSIDAD DE MÁLAGA

sean seleccionadas en los siguientes movimientos, hasta que dicha solución es excluida de la lista (ver Glover et al. [77]).

El procedimiento de búsqueda miope aleatorizado y adaptativo (*Greedy Randomized Adaptive Search Procedure*, GRASP) combina heurísticas constructivas con búsqueda local. En una primera fase se construye una solución. A continuación se van añadiendo componentes aleatoriamente, dentro de una lista restringida de candidatos. En una segunda fase se mejora dicha solución mediante búsqueda local, que a su vez puede utilizar búsqueda tabú (ver Feo y Resende [68]).

La búsqueda con vecindario variable (*Variable Neighborhood Search*, VNS) utiliza una estrategia que cambia entre diferentes vecindarios durante la búsqueda. Inicialmente se define un conjunto de vecindarios. En cada iteración se elige un candidato, se mejora la solución mediante búsqueda local y finalmente se compara con la solución inicial. Si hay mejora, se inicializa el contador de vecindarios. Si no, se repite el proceso en el siguiente vecindario (ver Mladenovi y Hansen [137]).

La búsqueda local iterada (*Iterated Local Search*, ILS) perturba una solución en cada iteración, y luego aplica un método de búsqueda local. El nuevo mínimo local es aceptado si supera un test de aceptación. La clave del método de perturbación es generar una solución que sirva como inicio a la búsqueda local, pero que no esté muy lejos de la actual para que no pueda ser considerada como una solución aleatoria. El criterio de aceptación filtra las nuevas soluciones en base a la historia de búsqueda y las características del nuevo mínimo local (ver Lourenço et al. [122]).

En relación a las metaheurísticas basadas en población, las más conocidas son los *algoritmos evolutivos*, los *algoritmos de estimación de la distrbución*, la *búsqueda dispersa*, la *optimización basada en colonias de hormigas* y la *optimización basada en cúmulos de partículas*.

Los algoritmos evolutivos (*Evolutionary Algorithms*, EA) se inspiran en la teoría de la evolución. Tras la generación de una población aleatoria, el método comprende tres fases: selección, reproducción y reemplazo. En la selección se toman los 'mejores' individuos, que son cruzados o recombinados generando nuevos individuos (reproducción). Existe un proceso llamado *mutación* que puede modificar algunos 'genes' en la generación de los nuevos individuos. En la fase de reemplazo se forman nuevas soluciones a partir de las soluciones generadas. En la mayoría de los casos se descartan los 'peores' individuos, es decir, aquellos que no sobreviven a la selección natural. Este proceso es repetido hasta que se produzca un criterio de parada (ver Eiben y Smith [65]).

Los algoritmos de estimación de la distribución (*Estimation of Distribution Algorithms*, EDA) son similares a los evolutivos. La principal diferencia con éstos es que se crean distribuciones de probabilidad que se ajustan bien a las mejores soluciones de la población y se hace un muestreo de esa distribución para generar la siguiente población (ver Heinz Mühlenbein [138]).

En la búsqueda dispersa (*Scatter Search*, SS) se mantiene un conjunto pequeño de soluciones de referencia, caracterizadas por ser de calidad y diversas ('separadas' en el espacio objetivo). Tras la creación de la población inicial y del conjunto de referencia, se crean subconjuntos de soluciones, que son recombinados y mejorados iterativamente (ver Glover [76]).

Los algoritmos basados en colonias de hormigas (*Ant Colony Optimization*, ACO) se inspiran en el comportamiento real de las hormigas cuando buscan comida. Se utilizan hormigas artificiales que simulan el recorrido en un grafo. Cada hormiga avanza de manera independiente al resto. Partiendo de un nodo inicial y un rastro inicial de feromona, cada hormiga elige el siguiente nodo de acuerdo al rastro de feromona y un heurístico asociado. Repitiendo el proceso, la hormiga termina produciendo una solución candidata. Una vez construida, se actualizan los rastros de feromona por los que han pasado las hormigas. Este rastro influye en el camino que realizarán más tarde otras hormigas en la búsqueda de la solución. El rastro de feromona puede también evaporarse a lo largo del tiempo, perdiendo así fuerza de atracción y evitando que el algoritmo de búsqueda converja siempre a óptimos locales. Esta simple idea ha resultado ser muy útil al utilizarse como metaheurística (ver Dorigo y Stützle [52]).

Los algoritmos de optimización basada en cúmulos de partículas (*Particle Swarm Optimizacion*, PSO) se inspiran en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces (Kennedy [100]). Se inicializa aleatoriamente un conjunto de soluciones en el espacio de búsqueda, llamadas partículas. Cada una de ellas posee una posición y una velocidad que cambia conforme avanza la búsqueda. El movimiento de una

UNIVERSIDAD DE MÁLAGA

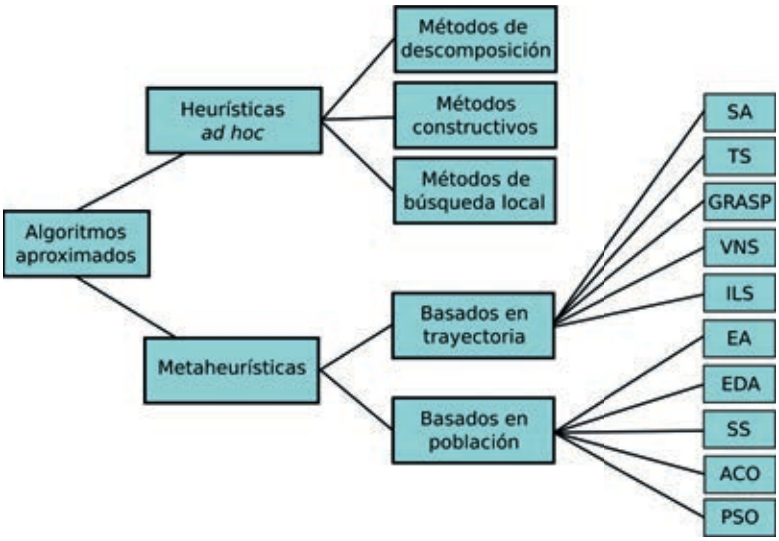


Figura 3.7: Clasificación de algoritmos aproximados.

partícula depende de su posición actual y de la posición de las partículas de su vecindario. En este caso el concepto de vecindario es distinto al expuesto anteriormente. El vecindario se define como un conjunto de partículas del cúmulo. En este contexto, el vecindario de una partícula puede ser global (todas las partículas del cúmulo se consideran vecinas), o local (solo las partículas más cercanas se consideran vecinas).

En la Figura 3.7, mostramos a modo de resumen la clasificación de los algoritmos aproximados que se acaba de describir.

Algoritmos aproximados multiobjetivo

Las metaheurísticas más usadas a la hora de resolver los problemas combinatorios multiobjetivo son los *algoritmos evolutivos multiobjetivo* (en inglés *MultiObjective Evolutionary Algorithms*, MOEAs). Varios de estos métodos han sido desarrollados desde hace varias décadas incluso para resolver problemas no convexos. La idea general en los MOEAs es aproximar el frente de Pareto mediante frentes parciales que van mejorando (o al menos no empeorando) en las distintas iteraciones del algoritmo, manteniendo en lo posible una buena dispersión de las soluciones a lo largo del espacio objetivo. Un interesante tutorial al respecto puede verse en el trabajo de Emmerich y Deutz [66]. Para profundizar en este ámbito, el lector puede remitirse al libro de Coello et al. [27] o al trabajo de Herzel et al. [89], donde se muestra un estudio sobre métodos aproximados en optimización multiobjetivo que cubre 52 artículos publicados durante los años 1979 y 2020. Sin entrar a enumerar la gran cantidad de trabajos existentes en este campo, dado que no es el objetivo de esta tesis, a continuación describiremos brevemente tres conocidos ejemplos de algoritmos evolutivos que sí serán utilizados y comparados con las propuestas algorítmicas de los próximos capítulos.

- NSGA-II
- SPEA2
- MOEA/D

El algoritmo NSGA-II (*elitist Non-dominated Sorting Genetic Algorithm*) fue publicado por Deb et al. [42] en el año 2002. Es uno de los algoritmos genéticos más conocidos. Se puede aplicar tanto a optimización continua como combinatoria. Esto implica una readaptación en los operadores de mutación y recombinación según el tipo de problema que se esté considerando. Partiendo de una población inicial, P_0, se repite un bucle generacional hasta que se satisfaga una condición de parada. En cada iteración, la población pasa a ser llamada P_0, P_1, P_2, etc. El bucle generacional tiene dos partes. En una primera parte se obtiene una población de descendencia (*offspring*), Q_t, a partir de dos padres de la población P_t, la recombinación

UNIVERSIDAD DE MÁLAGA

de los mismos y una mutación. En la segunda parte, a la población $P_t \cup Q_t$ se le aplica una ordenación no dominada (*ranking procedure*) que consiste en separar los individuos de la población por niveles, de manera que en el primer nivel están los individuos no dominados, en el segundo nivel están los no dominados (excluyendo los del primer nivel), y así sucesivamente. Además, los individuos dentro de un mismo nivel son ordenados según una distancia entre ellos (*crowding distance*), permitiendo así la dispersión de las soluciones obtenidas. Finalmente se establece un criterio de selección que determina la población de la siguiente generación, P_{t+1}, que tendrá el mismo número de elementos que P_t (el resto de individuos es descartado).

El algoritmo SPEA2 fue propuesto por Zitler et al. [192] y produjo una mejora respecto a su algoritmo predecesor, SPEA (*Strength Pareto Evolutionary Algorithm*), propuesto por Zitzler y Thiele [193]. SPEA usa una población regular y un archivo externo. Comienza con la población inicial y un archivo externo vacío. Todos los individuos no dominados de la población se copian al archivo. Los que sean dominados (o duplicados) se eliminan del archivo durante la operación de actualización. Si se excede del límite predefinido del archivo, los sobrantes son eliminados mediante una técnica de agrupamiento o *clustering*, manteniendo la no dominancia en los individuos del frente aproximado. A continuación se calcula el valor de *fitness* para cada individuo, tanto de la población como del archivo externo. A partir de ahí, se seleccionan los supervivientes mediante torneo binario[4] (*binary tournament*). Tras la aplicación de cruce y mutación se pasa a la siguiente iteración, donde se vuelven a copiar las soluciones no dominadas al archivo externo. Las diferencias de SPEA2 respecto de SPEA son principalmente tres: SPEA2 utiliza una estrategia de grano fino para asignar el *fitness*, donde se tienen en cuenta el número de individuos que cada solución domina y la cantidad de individuos que domina a dicha solución; utiliza una una técnica de estimación de la densidad que tendrá el vecindario con el fin de guiar la búsqueda de una manera más eficiente; y por último, posee un esquema de truncamiento a la hora de eliminar individuos del archivo externo, de forma que se conservan las soluciones de la frontera.

El algoritmo MOEA/D fue desarrollado por Zhang y Li [186]. Utiliza una descomposición (de ahí la D al final del nombre, *Decomposition*) del problema multiobjetivo en un número finito de problemas escalares y los optimiza simultáneamente. Cada subproblema es optimizado usando solo la información de sus subproblemas vecinos. La descomposición del problema multiobjetivo puede ser realizada de varias formas. En el caso de problemas combinatorios, las más usuales son: mediante suma ponderada o mediante la métrica de Tchebycheff ponderada. Partimos de un conjunto de vectores de pesos y de un punto de referencia. Calculamos la distancia euclídea entre cada pareja de vectores de pesos y consideramos el vecindario de un vector de pesos como aquel grupo de vectores más cercanos a él. La población está formada por la mejor solución encontrada en cada subproblema. Para cada individuo, solo sus vecinos son utilizados en busca de una mejora. Se mantiene una población externa, usada para guardar soluciones no dominadas durante la búsqueda. La recombinación y la búsqueda local se describen de manera genérica (pueden ser adaptadas por el programador).

Finalizada la pequeña descripción de estos tres algoritmos evolutivos, hacemos referencia al paquete *jMetal* [55, 139], que es un software de desarrollo programado en lenguaje Java y diseñado para resolver problemas de programación multiobjetivo usando metaheurísticas. Los algoritmos NSGA-II, SPEA2 y MOEA/D se encuentran implementados en esta herramienta (aparte de muchas otras metaheurísticas) y pueden encontrarse en `https://github.com/jMetal/jMetal`.

3.3. Metaheurísticas híbridas

A lo largo de los años, después de diversas publicaciones de algoritmos tanto exactos como metaheurísticos, surge de manera natural la idea de utilizar técnicas que combinen una metaheurística con otras técnicas de optimización. Estas otras técnicas podrían ser otras metaheurísticas, programación dinámica [9], o técnicas exactas ILP.

Conforme a lo descrito en el trabajo de Blum et al. [13], no hay una definición estándar de lo que es una metaheurística híbrida, si bien lo comentado en el párrafo anterior podría considerarse como una definición válida. La inexistencia de consenso en esta definición ha

[4] Véase Eiben y Smith [65].

UNIVERSIDAD DE MÁLAGA

hecho que el término esté sujeto a numerosas críticas, aunque ha permitido dejar más libertad a la hora de investigar en este campo hasta que finalmente se ha consolidado como una nueva línea de investigación. Sin ir más lejos, la propia definición de metaheurística podría entrar en relativa contradicción con la definición de metaheurística híbrida para el caso en que combinemos metaheurísticas con otras metaheurísticas. No obstante, aún con esta imprecisión definitoria, la idea es sencilla de entender: *una metaheurística híbrida es aquella que combina una metaheurística con otra técnica de optimización*. Por extensión lógica, los términos *hibridación* o *algoritmo híbrido* parecen también encajar como equivalentes al de metaheurística híbrida.

A la hora de clasificar las distintas metaheurísticas híbridas, comentamos brevemente algunas de las clasificaciones más significativas encontradas durante la revisión de la literatura:

- Blum [14] ya destacó la importancia de la hibridación de la metaheurísticas en el año 2003. Nombrando el trabajo de Talbi [167] sobre una taxonomía de dichos métodos híbridos, avanza que será un área de investigación prometedora. Inicialmente distingue tres formas de hibridación: las que incluyen una metaheurística en otra; las cooperativas, en las que varios algoritmos intercambian información entre sí; y las que integran metaheurísticas con métodos sistemáticos, como por ejemplo la programación con restricciones (*Constraint Programming*, CP). Este último tipo tiene a su vez tres enfoques distintos.

- Puchinger y Raidl [152] clasifican la combinación entre algoritmos exactos y metaheurísticas en dos tipos: las combinaciones colaborativas y las integrativas. En las colaborativas, el algoritmo exacto y la técnica metaheurística trabajan por separado en un nivel superior (ninguno está contenido en otro). Pueden ser de ejecución secuencial o paralela. En las de ejecución secuencial, o el método exacto actúa como preproceso del metaheurístico, o viceversa. En las paralelas, ambos algoritmos se ejecutan en paralelo y luego intercambian información. En los métodos integrativos, uno de los algoritmos (exacto o metaheurístico) es un subcomponente de la otra técnica. Dicho de otra forma, o bien el algoritmo exacto se incorpora dentro de la metaheurística, o bien la metaheurística se incorpora en el algoritmo exacto.

- Ehrgott y Gandibleux [61] publicaron en 2008 un trabajo que clasificaba los métodos híbridos multiobjetivo en tres categorías: los que combinan metaheurísticas con metaheurísticas, los que combinan metaheurísticas con otras técnicas y los que combinan metaheurísticas con preferencias dadas por el decisor. Como podemos observar, se produce una generalización natural de este concepto para el caso multiobjetivo.

Un caso que requiere especial mención es aquel algoritmo que combina una metaheurística con programación matemática. Recibe el nombre de *matheurística* (ver Boschetti et al. [15]) y también es frecuente encontrar en la literatura varios trabajos que emplean esta nomenclatura. En la Sección 5.1 introduciremos como propuesta algorítmica una matheurística capaz de aproximar el frente de Pareto en problemas binarios con restricciones. En el Capítulo 7 comentaremos una formulación genérica de método híbrido publicada por Blum et al. [12] y la aplicaremos al problema de selección de requisitos monoobjetivo.

3.4. Evaluación de las soluciones obtenidas

Evaluar la calidad de un frente de Pareto aproximado tras la ejecución de un algoritmo MOCO es el asunto que trataremos en esta sección. Comenzaremos definiendo el concepto de indicador de calidad, como medida descriptora de dicho frente. Este indicador será tanto mejor cuanto más se acerque su valor a un cierto número real (o infinito). En el segundo apartado describimos una formulación teórica basándonos en el concepto de método de comparación. Esta formulación permite, por así decirlo, inferir si la métrica utilizada nos confirma si un algoritmo es mejor que otro en base a las salidas de dichos algoritmos. Sin embargo, solo es válida para el caso de una única ejecución para cada algoritmo. Dado el carácter estocástico de algunos de ellos, se hace necesario la ejecución múltiple de cada una de las instancias para completarlo con un estudio estadístico, como se verá en la última parte de esta sección. La tercera parte define el concepto de algoritmo *anytime*, que es aquel que ofrece un 'buen'

UNIVERSIDAD DE MÁLAGA

conjunto de soluciones aunque paremos su ejecución en un momento cualquiera. Cerraremos la sección con una breve descripción de algunos test de hipótesis no paramétricos que evaluarán estadísticamente si un algoritmo es mejor que otro u otros, en base a varias ejecuciones independientes de los mismos y utilizando las métricas obtenidas por ellos.

3.4.1. Indicadores de calidad

Supongamos que después de ejecutar un cierto algoritmo disponemos de un frente de Pareto aproximado. ¿Cómo podemos evaluar el frente obtenido? ¿Hay alguna forma de medir la calidad de las soluciones? De una manera formal, en base al trabajo de Zitler et al. [194], definiremos el concepto de métrica o indicador de calidad. Para ello, partiremos de la premisa de que cualquier conjunto aproximado no contiene ninguna pareja de puntos dominados entre sí.

Definición 3.4. Sea $N \subset \mathbb{R}^p$ un subconjunto del espacio objetivo en un problema MOCO. Decimos que N es un *frente de Pareto aproximado* o un *conjunto aproximado* si cualquier elemento de N no está dominado por ningún otro elemento de N. Denotaremos por Ω al conjunto de todos los conjuntos aproximados asociados a ese problema.

Definición 3.5. Sea $m > 0$ un número natural. Un *indicador de calidad m-ario* es una función $I : \Omega^m \longrightarrow \mathbb{R}$. Cuando $m = 1$ también recibe el nombre de *indicador de calidad* (*unario*).

Obsérvese que para poder decir que un conjunto es aproximado, debemos garantizar que no tiene vectores débilmente dominados, por lo que pudiera ser necesario realizar un filtrado de soluciones después de la ejecución del algoritmo. Por otro lado, un indicador de calidad no es más que una función que asocia un valor real a un conjunto aproximado. La idea es encontrar buenos indicadores de calidad que permitan evaluar el frente aproximado. Definimos a continuación algunas de las métricas o indicadores de calidad más usadas en la literatura.

Definición 3.6. ***Overall Non-dominated Vector Generation*, ONVG**
Dado un conjunto aproximado N, definimos *ONVG* como el número de elementos del conjunto,

$$\text{ONVG}(N) = |N|. \tag{3.9}$$

La métrica ONVG, la más intuitiva y sencilla de entender, es claramente un indicador de calidad. Si bien no deja de ser un simple valor de referencia, podría considerarse un buen evaluador del rendimiento de un cierto algoritmo (a más soluciones, mejor calidad).

Definición 3.7. Hipervolumen (*HyperVolume*), HV
Dado un conjunto aproximado N, con d elementos, $N = \{z^1, z^2, \ldots, z^d\}$, el *hipervolumen* es la medida de Lebesgue de la región que es simultáneamente dominada por N y acotada por un punto de referencia $r \in \mathbb{R}^p$:

$$\text{HV}(N, r) = volumen\left(\bigcup_{j=1}^{d} [z^j, r]\right). \tag{3.10}$$

Notemos que se ha definido el hipervolumen como el volumen formado por la unión de una serie de cajas, acotadas superiormente todas ellas por el punto de referencia. Una definición equivalente puede encontrarse en Zitler et al. [191]: $\text{HV}(N) = \int_{z_l}^{z_u} \alpha_A(z)dz$, donde z_l y z_u son cotas inferior y superior, respectivamente, de la región del espacio objetivo sobre la que el hipervolumen es calculado, y $\alpha_A(z)$ es la función de logro, que toma el valor 1 si existe un valor en N que domine débilmente a z; y toma el valor 0, en otro caso. En esta definición, el punto de referencia es $r = z_u$.

El hipervolumen es sin duda una de la métricas con mayor potencial (como justificaremos posteriormente) de entre los indicadores unarios [120, 194]. Existen numerosos paquetes software que calculan el hipervolumen, dado un conjunto aproximado y un punto de referencia. Puede verse el reciente trabajo de Guerreiro et al. [83] que realiza un estudio de los algoritmos más significativos que calculan el hipervolumen y muestra hipervínculos a 25 paquetes

UNIVERSIDAD DE MÁLAGA

software distintos. De entre ellos destacamos el de While y Bradstreet [177], uno de los que tiene mayor eficiencia, el de Fonseca et al. [70], o el de Lacour et al. [109], que calcula el hipervolumen en base a una partición del espacio objetivo en cajas.

Respecto al punto de referencia a considerar a la hora de calcular el hipervolumen, tiene sentido pensar que el mismo debiera ser una cota superior del conjunto aproximado, es decir, $r \geq z^i, \forall i = 1, \ldots, d$. Una posible opción es tomar $r_i = \max\limits_{j=1,\ldots,d} z_i^j,\ \forall i = 1, \ldots, p$. Sin embargo, si resultase que alguna coordenada del punto nadir coincidiese con la de alguno de los puntos del conjunto aproximado, dicho punto no aportaría ningún valor adicional a esta métrica. Por ello, es usual asignar como punto referencia uno que sea dominado estrictamente por el punto nadir. Tal y como se establece en el trabajo de Li y Yao [115], no hay aún consenso sobre qué punto de referencia apropiado elegir a la hora de calcular HV. En algunos casos, se suele tomar 1.1 veces el punto nadir.

Definición 3.8. Indicador epsilon aditivo (*additive epsilon indicator*), ε_+
Dado un frente de referencia PF y un conjunto aproximado N, el indicador *epsilon aditivo* proporciona el mínimo valor para el que el conjunto aproximado tendría que trasladarse en el espacio objetivo para dominar débilmente al conjunto de referencia:

$$\varepsilon_+(N, PF) = \max_{x \in PF} \min_{y \in N} \max_{i=1,\ldots,p} (y_i - x_i)\,. \tag{3.11}$$

Este indicador de calidad es un caso particular del indicador *epsilon aditivo binario* (ver Zitzler et al. [194]), donde los dos conjuntos son N y PF, actuando este último como un conjunto fijo de referencia. Intuitivamente se puede observar que a menor valor de ε_+ mejor aproximación del frente tendríamos.

Definición 3.9. *General spread*, Δ^*
Dado un conjunto aproximado N, con $|N| > 1$, y $\{e_i\}_{i=1}^m$ sus soluciones extremas, que serán las imágenes de sus óptimos lexicográficos para PF, definimos la métrica *general spread* como

$$\Delta^*(N) = \frac{\sum_{i=1}^m d(e_i, N) + \sum_{x \in N} |d(x, N) - \overline{d}|}{\sum_{i=1}^m d(e_i, N) + |N|\overline{d}}, \tag{3.12}$$

donde $\overline{d} = \frac{1}{|N|} \sum_{x \in N} d(x, N)$, $d(x, N) = \min\limits_{y \in N; y \neq x} d^*(x, y)$ y d^* denota la distancia euclídea.

Puede verse la definición de la métrica Δ^* en los trabajos de Jiang et al. [94] y de Zhou et al. [188]. Es una generalización de la original métrica *spread* definida en el trabajo de Deb et al. [42], solo aplicable al caso biobjetivo. Podemos observar que todos los sumandos de la Ecuación (3.12) son no negativos, por lo que se cumple siempre que $\Delta^* \geq 0$. A menor valor, mejor 'distribución' del frente aproximado. Hemos añadido la condición $|N| > 1$ a la definición para evitar que la métrica quede indefinida. Ninguna de las definiciones dadas en [94, 188] deja claro el concepto de punto extremo, aunque parece deducirse que dichos puntos se refieren a las imágenes de los óptimos lexicográficos. Nuevamente esta métrica, si bien es binaria, puede considerarse unaria si dejamos como elemento fijo el frente de Pareto o el de referencia. En realidad PF solo es necesario para calcular los puntos extremos. Si se desconocen, pueden considerarse como extremos los extremos del conjunto aproximado. Obsérvese también que cada imagen de un óptimo lexicográfico no obtenida en la aproximación produce un valor positivo en $\sum_{i=1}^m d(e_i, N)$ y por lo tanto un mayor valor de Δ^*.

Definición 3.10. Distancia generacional (*Generational Distance*), GD
Dado un conjunto aproximado N y su frente de Pareto, PF, definimos la *distancia generacional* como

$$\mathrm{GD}(N, PF) = \frac{\left(\sum_{i=1}^{|N|} d_i^2\right)^{1/2}}{|N|}, \tag{3.13}$$

siendo d_i la distancia euclídea entre el vector i-ésimo en N y el vector más cercano en PF.

Nuevamente consideramos como conjunto de referencia el frente de Pareto completo, si bien la definición es aplicable a cualquier otro conjunto de referencia. La definición de esta métrica puede encontrarse en Liefooghe y Derbel [116]. Mide, de alguna forma, cómo de lejos se encuentra el conjunto aproximado respecto del frente de Pareto, por lo que cuanto más próximo a cero sea este valor, mejor calidad tendrán las soluciones.

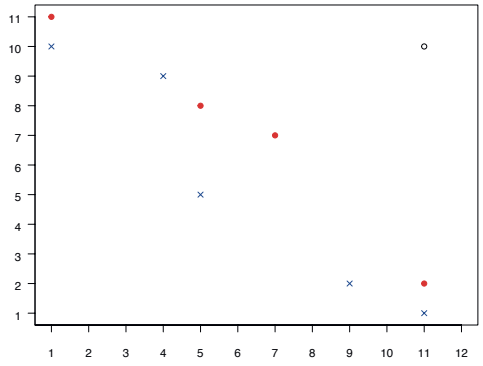
UNIVERSIDAD DE MÁLAGA

Definición 3.11. Distancia generacional inversa (*Inverted Generational Distance*), IGD
Dado un conjunto aproximado N, y su frente de Pareto, PF, definimos la *distancia generacional inversa* como la menor distancia entre cualquier elemento de PF y su solución más cercana en N,

$$\text{IGD}(N, PF) = \text{GD}(PF, N). \tag{3.14}$$

La distancia generacional inversa [116] procede de manera inversa a la distancia generacional. En este caso medimos cómo de cerca está el frente de Pareto respecto al conjunto aproximado. En general, los valores de GD e IGD son distintos.

Apliquemos las definiciones anteriores en un ejemplo práctico para el caso bidimensional. Disponemos de un frente aproximado y su correspondiente frente de Pareto dados por la Figura 3.8. El punto de referencia considerado es el punto nadir. Como tenemos 4 vectores en la aproximación, es claro que ONVG $= 4$. Para hallar el hipervolumen, consideramos la suma de los volúmenes de las cajas $[(1,11),(11,10)]$,$[(5,8),(11,10)]$ y $[(7,7),(11,8)]$. Obsérvese que la primera caja es vacía (el punto $(1,11)$ no domina al punto de referencia). Así, HV $= 0 + 12 + 4 = 16$. Además, el punto $(11,2)$ no aporta ningún valor al hipervolumen, ya que una de sus coordenadas coincide con su correspondiente coordenada del punto nadir. Generalmente se suele tomar como punto de referencia una cota estrictamente superior de z^N, aunque no lo hemos hecho así en este caso simplemente por su efecto didáctico. Calculemos ahora la métrica Δ^*. Los puntos extremos son los óptimos lexicográficos, $e_1 = (1,10)$ y $e_2 = (11,1)$. El término $\sum_{i=1}^{2} d(e_i, N)$ se obtiene sumando las mínimas distancias de cada punto extremo al conjunto N. En este caso, $\sum_{i=1}^{2} d(e_i, N) = 1+1 = 2$. Por otro lado, tenemos que $\overline{d} = \frac{1}{|N|}\sum_{x\in N} d(x, N)$ $= \frac{1}{4}\left(5 + \sqrt{5} + \sqrt{5} + \sqrt{41}\right)$, Repitiendo el mismo proceso para obtener $\sum_{x\in N}|d(x, N) - \overline{d}|$ y operando, obtenemos que $\Delta^* = \frac{7-2\sqrt{5}+\sqrt{41}}{7+2\sqrt{5}+\sqrt{41}} \approx 0.4996$. La métrica GD utiliza las distancias entre cada punto de N y el más cercano a PF. Estas distancias son $1, \sqrt{2}, \sqrt{8}, 1$ (siguiendo el orden de aparición de los elementos de N en la Figura 3.8). Tomando la raíz cuadrada de la suma de sus cuadrados y dividiéndolo por el cardinal de N, concluimos que GD $= \sqrt{3}/2$. Para el cálculo de IGD tomamos ahora como elementos a comparar los de PF y el más cercano a él en N. En este caso, IGD $= \frac{1}{5}\left(\sqrt{1^2 + (\sqrt{2})^2 + (\sqrt{8})^2 + 2^2 + 1^2}\right) = 4/5$. Finalizamos calculando el indicador ε_+. Las operaciones para el cálculo de $\max_{i=1,2}(y_i - x_i)$ en cada combinación de vectores de N y PF pueden verse en la Tabla 3.3. Por ejemplo, si comparamos los vectores $(1,11)$ y $(1,10)$, tenemos que $\max\{1-1, 11-10\} = 1$, y este es el valor que aparece en la casilla correspondiente. Repetimos el mismo proceso con las restantes celdas de la tabla. El siguiente paso es obtener $\min_{y\in N}\max_{i=1,2}(y_i - x_i)$ para cada $x \in PF$ fijado, que es equivalente a hallar el mínimo de cada columna de la tabla. Los valores mínimos de cada columna (por orden) son 1, 1, 2, 2 y 1. El valor final de la métrica será el máximo de todos esos valores, por lo que $\varepsilon_+ = \max\{1,1,2,2,1\} = 2$.



$N = \{(1,11),(5,8),(7,7),(11,2)\}$
$PF = \{(1,10),(4,9),(5,5),(9,2),(11,1)\}$
Punto de referencia: $r = z^N = (11,10)$

ONVG $= 4$	GD $= \frac{\sqrt{3}}{2}$
HV $= 16$	IGD $= \frac{4}{5}$
$\Delta^* \approx 0.4996$	$\varepsilon_+ = 2$

Figura 3.8: Cálculo de métricas asociadas a un frente aproximado para $p = 2$. En rojo marcamos los puntos del conjunto aproximado y en azul el frente de Pareto.

UNIVERSIDAD DE MÁLAGA

TABLA 3.3: Cálculo de $\max_{i=1,2}(y_i - x_i)$ para cada combinación de vectores en N y PF según el ejemplo de la Figura 3.8.

	(1,10)	(4,9)	(5,5)	(9,2)	(11,1)
(1,11)	1	2	6	9	10
(5,8)	4	1	3	6	7
(7,7)	6	3	2	5	2
(11,2)	10	7	6	2	1

Es fácil deducir el rango de cada uno de los indicadores que hemos definido. Así, vemos que ONVG $\in \mathbb{N}$; HV, GD, IGD y Δ^* son números reales no negativos, y ε_+ puede tomar en principio cualquier valor real. Aunque pudiese parecer extraño, es posible que la métrica ε_+ tome un valor negativo, y esto ocurre cuando todos los puntos del conjunto de referencia son dominados estrictamente por algún punto del conjunto aproximado. Vamos a demostrar que esto se cumple en un espacio objetivo de cualquier dimensión.

Proposición 3.6. Sean N y R dos conjuntos aproximados. Entonces $\varepsilon_+(N, R) < 0$ si y solo si todo punto del conjunto R es dominado estrictamente por algún punto de N.

Demostración. Basándonos en la definición, tenemos que

$$\begin{aligned}
\varepsilon_+(N,R) = \max_{x\in R}\min_{y\in N}\max_{i=1,\dots,p}(y_i - x_i) < 0 \iff \\
\min_{y\in N}\max_{i=1,\dots,p}(y_i - x_i) < 0, \quad \forall x \in R \iff \\
\forall x \in R,\, \exists\, y^* \in N \mid \max_{i=1,\dots,p}(y_i^* - x_i) < 0 \iff \\
\forall x \in R,\, \exists\, y^* \in N \mid y_i^* - x_i < 0, \quad \forall i = 1,\dots,p \iff \\
\forall x \in R,\, \exists\, y^* \in N \mid y_i^* < x_i \quad \forall i = 1,\dots,p \iff \\
\forall x \in R,\, \exists\, y^* \in N \mid y^* \text{ domina estrictamente a } x.
\end{aligned}$$

□

Corolario 3.1. Si un conjunto de referencia R es un subconjunto del frente de Pareto, siempre se cumplirá que $\varepsilon_+(N, R) \geq 0$.

Demostración. Se demuestra como consecuencia inmediata de que los puntos del frente de Pareto no son dominados estrictamente por ningún punto de N. □

Las seis métricas definidas en este apartado son bastante usadas en el campo de la optimización multiobjetivo. Riquelme et al. [156] elaboraron un estudio sobre 54 métricas, discutiendo su uso y tendencias, así como las ventajas y desventajas de algunas de ellas. Tomando como referencia desde 2005 hasta 2013 las conferencias EMO (*International Conference Series on Evolutionary Multi-Criterion Optimization*), que se celebran bianualmente, concluyeron que la medida más usada en todos los trabajos publicados era el hipervolumen, seguida de la métrica GD. Δ^* junto con IGD ocupaban el cuarto lugar. ONVG compartía el quinto lugar junto con otras medidas. En tercer lugar aparece la métrica binaria ε_+, que nosotros hemos adaptado a su versión unaria. Por todo lo anterior podemos conjeturar que estos indicadores podrían considerarse apropiados a la hora de medir la calidad de un frente aproximado. De hecho, éstas métricas serán utilizadas en los experimentos computacionales de los próximos capítulos.

3.4.2. Métodos de comparación

¿Cómo podríamos medir la calidad de un indicador? Ya hemos definido algunos de ellos, que además son utilizadas ampliamente en el ámbito de la optimización multiobjetivo, pero no conocemos las ventajas y desventajas que tienen, ni si producen buenas o malas estimaciones. Si ejecutamos dos algoritmos diferentes para un mismo problema nos gustaría saber, en la medida de lo posible, si alguno de ellos rinde mejor que el otro. Basándonos en el trabajo teórico de Zitler et al. [194], formalizaremos de manera resumida el concepto de método

UNIVERSIDAD DE MÁLAGA

de comparación y como éstos nos ayudarán a entender por qué ninguna métrica unaria (ni ninguna combinación finita de ellas) será jamás lo suficientemente buena para determinar de manera precisa si un conjunto aproximado es mejor que otro.

Empezaremos definiendo relaciones entre conjuntos aproximados. Supongamos que N_1 y N_2 son dos conjuntos aproximados. Se definen las relaciones $\gg$, $\succ$, $\rhd$, $\succeq$ y $\|$ en la Tabla 3.4. De manera análoga se pueden definir las relaciones $\ll$, $\prec$, $\lhd$ y $\preceq$.

Tabla 3.4: Definición de relaciones de comparación entre conjuntos aproximados.

Relación	Denominación	Definición
$N_1 \gg N_2$	(domina estrictamente)	Para todo $z^2 \in N_2$, existe $z^1 \in N_1$ tal que $z^1 < z^2$
$N_1 \succ N_2$	(domina)	Cada $z^2 \in N_2$ es dominado por al menos un $z^1 \in N_1$
$N_1 \rhd N_2$	(mejor)	Cada $z^2 \in N_2$ es débilmente dominado por al menos un $z^1 \in N_1$ y $N_1 \neq N_2$
$N_1 \succeq N_2$	(domina débilmente)	Cada $z^2 \in N_2$ es débilmente dominado por al menos un $z^1 \in N_1$
$N_1 \| N_2$	(incomparable)	Ni N_1 domina débilmente a N_2, ni N_2 domina débilmente a N_1

Vemos que se han definido distintas formas de decir que una aproximación es mejor que otra (salvo en la última fila, donde se define la incomparabilidad o indiferencia), según distintos grados de precisión, siendo preferible la de cada fila respecto a la de la fila siguiente. De hecho, existe un orden natural entre las relaciones: $N_1 \gg N_2 \Rightarrow N_1 \succ N_2 \Rightarrow N_1 \rhd N_2 \Rightarrow N_1 \succeq N_2$.

Los indicadores de calidad necesitan ser relacionados de alguna forma con los conjuntos aproximados de manera que proporcionen información útil que nos permita concluir si alguna aproximación es mejor que otra.

Definición 3.12. Sean N_1 y N_2 dos conjuntos aproximados, $\boldsymbol{I} = (I_1, I_2, \ldots, I_k)$ una combinación de indicadores de calidad, y $E : \mathbb{R}^k \times \mathbb{R}^k \longrightarrow \mathbb{B}$ una función que mapea dos vectores en un booleano (llamada *interpretation function* o función de interpretación). Definimos el *método de comparación* $C_{\boldsymbol{I},E}(N_1, N_2)$ como el booleano devuelto por la función E, aplicado a la combinación de funciones indicadoras, es decir,

$$C_{\boldsymbol{I},E}(N_1, N_2) = E\left(\boldsymbol{I}(N_1), \boldsymbol{I}(N_2)\right),$$

donde $\boldsymbol{I}(N) = (I_1(N), I_2(N), \ldots, I_k(N))$, para cualquier $N \in \Omega$.

Con objeto de simplificar la notación, se utilizará $E :=$ («expresión») para indicar la condición que debe cumplirse para que $E\left(\boldsymbol{I}(N_1), \boldsymbol{I}(N_2)\right) = \mathit{true}$. Veamos un ejemplo de método de comparación asociado a una métrica.

Ejemplo 3.2. Supongamos un problema combinatorio multiobjetivo de minimización y definamos dos indicadores de calidad asociados a un conjunto aproximado N:

$$I_1^{HC}(N) = \sup_{a \in \mathbb{R}} \{\{(a, a, \ldots, a)\} \rhd N\},$$
$$I_2^{HC}(N) = \inf_{b \in \mathbb{R}} \{\{(b, b, \ldots, b)\} \lhd N\}.$$

La combinación $HC = \left(I_1^{HC}, I_2^{HC}\right)$ se conoce como *enclosing hypercube indicator*. Por ejemplo, dado el conjunto aproximado $N = \{(1,4),(3,2),(5,1)\}$, tendríamos que $I_1^{HC} = 1$ e $I_2^{HC} = 5$. Si definimos la siguiente función de interpretación

$$E := \left(I_1^{HC}(N) = I_2^{HC}(N)\right),$$

el método de comparación $C_{HC,E}$ devolverá *false* para este conjunto aproximado. De hecho, dicho método solo valdrá *true* cuando dicho conjunto tenga un único elemento.

La definición de método de comparación parece demasiado formal e inconclusa. Sigue sin especificar, por ejemplo, ¿qué significa que $C_{\mathbf{I},E}$ sea verdadero? Pues bien, si $C_{\mathbf{I},E}$ es

UNIVERSIDAD DE MÁLAGA

una condición suficiente para que $N_1 \triangleright N_2$ (o para cualquier otra relación entre conjuntos aproximados), entonces podríamos decir que el método de comparación es capaz de indicar que N_1 es mejor que N_2. Por otro lado, si además $C_{\mathbf{I},E}$ es una condición necesaria para $N_1 \triangleright N_2$, entonces el método de comparación caracterizaría completamente la comparación de conjuntos aproximados. Esto nos da lugar a las siguientes definiciones:

Definición 3.13. Compatibilidad y completitud

Sea $\blacktriangleright$ una relación tal que $\blacktriangleright \in \{\gg, \succ, \triangleright, \succeq\}$. Sea E una función de interpretación e $\boldsymbol{I}$ un vector de indicadores de calidad.

Decimos que $C_{\boldsymbol{I},E}$ es $\blacktriangleright$-compatible si

$$C_{\boldsymbol{I},E}(N_1, N_2) \Rightarrow N_1 \blacktriangleright N_2, \quad \text{o bien} \quad C_{\boldsymbol{I},E}(N_1, N_2) \Rightarrow N_2 \blacktriangleright N_1, \quad \forall N_1, N_2 \in \Omega.$$

Decimos que $C_{\boldsymbol{I},E}$ es $\blacktriangleright$-completo si

$$N_1 \blacktriangleright N_2 \Rightarrow C_{\boldsymbol{I},E}(N_1, N_2), \quad \text{o bien} \quad N_2 \blacktriangleright N_1 \Rightarrow C_{\boldsymbol{I},E}(N_1, N_2), \quad \forall N_1, N_2 \in \Omega.$$

Ejemplo 3.3. Basándonos en la definición de HC dada en el Ejemplo 3.2, definimos la función de interpretación

$$E := \left(I_2^{HC}(N_1) < I_1^{HC}(N_2)\right).$$

Podemos observar que $C_{HC,E}$ es $\gg$-compatible para cualesquiera $N_1, N_2 \in \Omega$. En efecto, si la función E es verdadera, la 'cota superior' del conjunto N_1 dominará a la 'cota inferior' del conjunto N_2, por lo que se cumplirá que $N_1 \gg N_2$. Sin embargo, el método de comparación $C_{HC,E}$ no es $\gg$-completo (de hecho no es $\blacktriangleright$-completo en ningún caso), ya que si tomamos $N_1 = \{(1,2),(3,1)\}$ y $N_2 = \{(2,3),(4,2)\}$, concluimos que $N_1 \gg N_2$, pero $I_2^{HC}(N_1) = 3$ e $I_1^{HC}(N_2) = 2$, por lo que la función de comparación devuelve *false*.

Es evidente que lo ideal sería encontrar un método de comparación que fuese compatible y completo respecto a cualquiera de las relaciones de preferencia definidas en la Tabla 3.4. Sin embargo esto no sé podrá conseguir nunca para los indicadores de calidad unarios.

Teorema 3.1. Supongamos un problema de optimización con $p \geq 2$ objetivos. No existe método de comparación $C_{\boldsymbol{I},E}$ basado en una combinación finita $\boldsymbol{I}$ de indicadores de calidad unarios tales que se verifique $C_{\boldsymbol{I},E}(N_1, N_2) \iff N_1 \triangleright N_2$ para cualesquiera $N_1, N_2 \in \Omega$.

Demostración. Ver Teorema 1 de [194]. □

Visto lo anterior, queda demostrado que ninguna métrica unaria (ni combinación finita de estas) podrá cuantificar exactamente la comparación de dos frentes aproximados e indicar si uno es mejor que otro, en sentido teórico estricto. Lo mejor que se podrá conseguir es $\gg$-compatibilidad sin ninguna completitud, o bien $\not\triangleright$-compatibilidad[5] con $\triangleright$-completitud.

En un sentido menos teórico que lo comentado anteriormente, algunos trabajos [106, 190, 47] definen otra característica que sería deseable que los indicadores de calidad tuviesen. Nos referimos a la característica de ser *Pareto compliant*, que viene a significar que si un conjunto aproximado es mejor que otro, el indicador de calidad del primero no puede ser peor que el del segundo. En otras palabras, no se debe contradecir el orden inducido por la relación de Pareto.

Definición 3.14. Un indicador de calidad I es *Pareto compliant* si, para cualesquiera $A, B \in \Omega$ con $A \succeq B$ y $B \not\succeq A$, tenemos que $I(A) \geq I(B)$. El caso estricto se produce cuando $I(A) > I(B)$.

Considerando el conjunto de los indicadores unarios, HV es el único que se conoce hasta el momento que sea *Pareto compliant* estricto.

En la Tabla 3.5 mostramos las características principales que poseen cada una de las métricas definidas al comienzo de esta sección.

El hecho de que un indicador no sea completo ni compatible respecto a algún orden y además no sea *Pareto compliant* tampoco significa que no sea un buen indicador. Éstos

[5] Una relación es $\not\triangleright$-compatible cuando no es $\triangleright$-compatible.

UNIVERSIDAD DE MÁLAGA

Tabla 3.5: Características de las métricas usadas en esta tesis, indicado su función de interpretación.

Métrica	*¿Pareto compliant?*	Compatibilidad	Completitud	Función de interpretación
ONVG	Sí	-	-	$E := (\text{ONVG}(N_1) > \text{ONVG}(N_2))$
HV	Sí	$\not\triangleright$	$\gg$	$E := (\text{HV}(N_1) > \text{HV}(N_2))$
ε_+	Sí	$\not\triangleright$	$\gg$	$E := (\varepsilon_+(N_1) < \varepsilon_+(N_2))$
GD	No	-	-	$E := (\text{GD}(N_1) < \text{GD}(N_2))$
IGD	No	-	-	$E := (\text{IGD}(N_1) < \text{IGD}(N_2))$
Δ^*	No	-	-	$E := (\Delta^*(N_1) < \Delta^*(N_2))$

podrían medir otras características también deseables en la aproximación, como comentaremos en el Apartado 3.4.3.

Para ampliar información sobre indicadores de calidad puede verse el reciente trabajo de Li y Yao [115], donde realizan un estudio sobre 100 indicadores de calidad, detallan algunos de ellos, y discuten algunas propiedades que dichos indicadores tienen o deberían tener. En esta publicación se vuelve a concluir que no existe ningún indicador unario que se ajuste a todas las situaciones deseadas. Otro trabajo a referenciar es el de Ishibuchi et al. [92], donde adaptan la métrica IGD y definen el indicador IGD+, que sí es *Pareto compliant*.

3.4.3. Algoritmos *anytime*

Los indicadores de calidad comentados en el apartado anterior nos cuantifican alguna/s característica/s del frente aproximado. Esta característica puede ser muy variada entre una métrica y otra. Por ejemplo, podría evaluarse cuánto de cerca está un conjunto aproximado respecto del conjunto PF, o cómo de bien distribuidas están las soluciones por el espacio objetivo. Vamos a clasificar estos indicadores conforme al trabajo realizado por Jiang et al. [94], que dividieron las métricas MOO en cuatro grandes grupos.

- Métricas de capacidad (*capacity metrics*). Cuantifican el número o ratio de soluciones no dominadas en una aproximación conforme a unos requerimientos predefinidos. En esta categoría encontramos la métrica ONVG, o también el indicador ONVGR (*Overall Non-dominated Vector Generation Ratio*), definido por

$$\text{ONVGR}(N) = \frac{|N|}{|PF|}, \tag{3.15}$$

  y que mide el porcentaje total de soluciones encontradas en el frente. Esta métrica tiene más sentido utilizarla tras la ejecución de un algoritmo exacto, ya que en este caso $N \subseteq PF$.

- Métricas de convergencia (*convergence metrics*). En este caso medimos el grado de proximidad del conjunto de soluciones respecto del frente de Pareto. Las métricas GD y ε_+ se encuentran en este grupo.

- Métricas de diversidad (*diversity metrics*). Miden la distribución y el *spread* de las soluciones. Decimos que las soluciones tienen una buena distribución cuando mantienen una distancia 'similar' entre ellas, y que tienen un buen *spread* cuando están dispersas por el espacio objetivo. Así, podrían existir aproximaciones bien distribuidas, pero con un *spread* pobre, o viceversa. La métrica Δ^* es una métrica de diversidad de las que mide la calidad del *spread*.

- Metricas de convergencia-diversidad (*convergence-diversity metrics*). Miden la calidad de la solución en términos de convergencia y diversidad en una escala real. Las métricas HV e IGD se encuentran en este grupo. En ocasiones también es útil considerar el porcentaje total de hipervolumen alcanzado,

$$\text{HVR}(N, r) = \frac{\text{HV}(N, r)}{\text{HV}(PF, r)}. \tag{3.16}$$

UNIVERSIDAD DE MÁLAGA

Esta métrica solo es posible usarla si conocemos el frente completo, y proporciona numéricamente una información más útil que la de HV, que generalmente suele ser un valor elevado.

Como podemos observar, las seis métricas que definimos en el apartado anterior ocupan los cuatro tipos de métricas considerados en esta clasificación. Cuando diseñamos un algoritmo precisamente lo que buscaremos es que las soluciones posean buena convergencia y buena dispersión. Más aún, si esto puede obtenerse 'en cualquier momento', es decir, si cuando se detiene la ejecución del algoritmo siempre se consigue obtener una buena dispersión de soluciones por el espacio objetivo, y además éstas son próximas al frente de Pareto, diremos entonces que es un buen algoritmo *anytime*. El concepto de *anytime* fue introducido por Dean y Boddy [40].

Definición 3.15. (*Anytime* según Dean and Boddy) Un algoritmo es *anytime* si cumple las siguientes propiedades:
(i) Pueden ser detenidos en cualquier momento y aún así devolverán una salida.
(ii) La salida devuelta mejora, en cierta manera, en función del tiempo.

Esta definición, muy general y poco precisa, ha ido matizándose según la interpretación que se ha ido dando por los investigadores hasta derivar en una definición algo más concreta [120].

Definición 3.16. Un algoritmo que muestra un buen balance entre la calidad de las soluciones y el tiempo de ejecución se dice que tiene un buen comportamiento *anytime*.

Para una información más detallada sobre el concepto *anytime* puede verse el trabajo de Shlomo Zilberstein [189].

3.4.4. Contrastes de hipótesis no paramétricos

Para comparar el rendimiento de los algoritmos, aun habiendo sido evaluados con distintas métricas, es necesario el uso de test estadísticos que permitan confirmar si un algoritmo es mejor que otro. De hecho, desde comienzos del siglo XXI ya se vienen realizando cada vez más test estadísticos en los trabajos que comparan el rendimiento de distintos algoritmos (ver Derrac et al. [44]). Los métodos estadísticos pueden ser paramétricos o no paramétricos. Los paramétricos se basan en suposiciones como la independencia, normalidad y homocedasticidad de los datos, que generalmente no se cumplen para los algoritmos MOO. Por ello, en este tipo de algoritmos la evaluación debe ser realizada mediante test no paramétricos.

Es bien conocido que un contraste de hipótesis es un procedimiento que infiere si una propiedad que se supone en una población estadística es compatible con lo observado en una muestra de dicha población. Tras definir la hipótesis nula H_0 y la hipótesis alternativa H_1, se calcula un estadístico que depende de los valores muestrales, y se decide aceptar H_0 o rechazarla (en favor de H_1) en base a un nivel de significación $\alpha \in (0, 1)$. Para ser más precisos, en la decisión adoptada al realizar un contraste de hipótesis, decimos que *rechazamos la hipótesis nula*, o bien que *no hay evidencia suficiente para rechazar la hipótesis nula*. En lugar de trabajar con los niveles de significación, es más usual usar el *p-valor*, que es un número en el intervalo $(0, 1)$ que, cuanto menor valor tenga, mayor probabilidad de rechazar H_0 inferimos. Es por ello que generalmente, cuando el p-valor es menor que 0.05 o 0.01, decimos que el test estadístico es significativo, es decir, disponemos de evidencia estadística para rechazar la hipótesis nula.

Los tests no paramétricos aplicados a la evaluación de algoritmos MOO se pueden clasificar en aquellos que realizan comparaciones pareadas (*pairwise comparisons*) y los que no lo hacen. Las comparaciones pareadas comparan muestras en las que existe una determinada relación entre los sujetos muestreados. Por ejemplo, supongamos que los algoritmos A y B se aplican a las funciones $f_1, f_2, \ldots, f_{20}$. El resultado de A en la función f_1 se compara con el resultado de B en la función f_1, y así sucesivamente. El test tiene en cuenta estos 40 resultados para dar un veredicto sobre si A es diferente de B. Pero hay veces en las que los valores no tienen relación. Por ejemplo, si los algoritmos A y B fuesen estocásticos y se aplicasen 20 veces a una función objetivo. El resultado de la primera ejecución de A no tendría sentido emparejarla con el resultado de la primera ejecución de B. Aquí todas las ejecuciones son 'iguales' desde un punto de vista estadístico.

UNIVERSIDAD DE MÁLAGA

Dados varios algoritmos a comparar, la hipótesis nula muestra la igualdad de rendimiento entre todos ellos, y la alternativa determina que existe alguna diferencia. Si el número de algoritmos es mayor que dos, suele ser necesario realizar un análisis *post-hoc* para determinar dónde se encuentran dichas diferencias entre los algoritmos. Decimos entonces que el test es de comparaciones múltiples (*multiple comparisons*).

A continuación comentamos brevemente el funcionamiento de algunos test no paramétricos, concretamente aquellos que utilizaremos en los resultados computacionales de las propuestas algorítmicas de esta tesis.

Test de Wilcoxon de los rangos signados

El test de Wilcoxon de los rangos signados [180] es un test pareado que se usa para responder a la pregunta de si dos muestras representan dos poblaciones diferentes. Aplicado a nuestro caso la pregunta sería si dos algoritmos se comportan igual en cuanto a 'rendimiento'. Disponemos de n valores pareados, representados (x_i, y_i). Se supone que los valores $d_i = y_i - x_i$ son independientes entre sí, y que son variables aleatorias continuas. La formulación viene dada por

$$H_0 : \theta = 0,$$
$$H_1 : \theta \neq 0,$$

siendo θ la mediana de la distribución de las diferencias. Las diferencias d_i son ordenadas de acuerdo a su valor absoluto (previamente son normalizadas si los rangos de las poblaciones fuesen distintos) mediante un rango (*ranking*). En caso de empate se considera el valor medio de los rangos. Por ejemplo, para un empate entre una muestra de rango 1 y otra de rango 2, se considera rango 1.5 para ambas. Definimos R^+ como la suma de rangos en los que el primer algoritmo es mejor que el segundo, y R^- como la suma de los rangos en el caso en que el segundo algoritmo sea mejor. Los empates son tenidos también en cuenta, de manera que

$$R^+ = \sum_{d_i>0} rango(d_i) + \frac{1}{2}\sum_{d_i=0} rango(d_i),$$
$$R^- = \sum_{d_i<0} rango(d_i) + \frac{1}{2}\sum_{d_i=0} rango(d_i).$$

Finalmente, se considera el estadístico $W = \text{mín}\{R^+, R^-\}$. Este valor se compara con el valor de la distribución de Wilcoxon para n grados de libertad, y si es menor o igual, se rechaza la hipótesis nula. En caso contrario, no hay evidencia suficiente para rechazarla.

Cuando tratamos de extraer una conclusión sobre la comparación de más de dos algoritmos, al comparar los test dos a dos, perdemos significancia. Para ello existen test de comparaciones múltiples que establecen correcciones, como la de Bonferroni o la de Tukey-Kramer.

Test de Friedman

El test de Friedman [72] utiliza k poblaciones, cada una de ellas con mediana θ_i, $i = 1, \ldots, k$. En este caso estamos interesados en saber si existe igualdad entre las medianas de todas las poblaciones, o hay diferencias entre al menos dos de ellas. La formulación del test es

$$H_0 : \theta_1 = \theta_2 = \ldots = \theta_k,$$
$$H_1 : \text{otro caso.}$$

Los datos se ordenan por filas o bloques (que serán las instancias) y por columnas o tratamientos (que serán los algoritmos). Para n observaciones disponemos de una matriz de orden $n \times k$. Para cada fila i, ordenamos los datos según rangos, siendo 1 el mejor y k el peor (mismo tratamiento para los empates que en el caso del test de Wilcoxon). Esta ordenación da lugar a una nueva matriz cuyos elementos son r_i^j, siendo $i \in \{1, \ldots, n\}$ el número de observación, y $j \in \{1, \ldots, k\}$ el tratamiento correspondiente. A continuación, para cada algoritmo j se hace la media de los rangos obtenidos en todas las muestras, obteniendo un rango final,

UNIVERSIDAD DE MÁLAGA

$R_j = \frac{1}{n}\sum_i r_i^j$, $\forall j = 1, \ldots, k$. El estadístico del contraste viene dado por

$$F = \frac{12n}{k(k+1)}\left(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right),$$

que se compara con el valor correspondiente a la distribución χ^2 con $k-1$ grados de libertad para el caso $n > 10$ y $k > 5$. Para otros valores menores, la distribución aproximada se contrasta con otras tablas, que pueden encontrarse en [159].

Si un test de comparación múltiple es significativo, podemos concluir que existen diferencias entre los algoritmos. Sin embargo, seguimos sin saber dónde están esas diferencias, o si puede considerarse si un algoritmo es mejor que otro. En estos casos se recurren a los test *post-hoc*.

Test *post-hoc* de Nemenyi

Este test, formulado por Peter B. Nemenyi [140], realiza un análisis *post-hoc* una vez obtenido significancia en el test de Friedman. Utiliza un método similar al test de Tukey para el ANOVA. Si tenemos k poblaciones, R_j es la suma de los rangos en el test de Friedman para el algoritmo j y n_j es su tamaño, entonces diremos que la diferencia entre dos grupos (algoritmos) i y j es diferente para cierto nivel α cuando se satisfaga la desigualdad

$$\left|\frac{R_i}{n_i} - \frac{R_j}{n_j}\right| > \frac{q_{k,\alpha}}{\sqrt{2}}\sqrt{\frac{k(k+1)}{6n}},$$

siendo $q_{k,\alpha}$ el valor crítico para la distribución t-Student con k grados de libertad a nivel α. Este cálculo se repite para cada pareja de algoritmos y se obtiene una matriz (simétrica) que indica los p-valores asociados a cada pareja (véase Janez Demšar [43]).

Los distintos test *post-hoc* difieren entre ellos en la forma en que ajustan el valor de α para compensar la comparación múltiple. Para más información sobre este asunto puede verse Derrac et al. [44], que incluye una descripción de algunos de los métodos *post-hoc*.

Para finalizar, comunicamos al lector que existen diversos paquetes informáticos que calculan las métricas descritas en este capítulo, así como para realizar los diferentes test estadísticos. Por ejemplo, el lenguaje de programación **R**[6] proporciona el paquete *eaf*, que permite calcular ONVG, HV, GD, IGD y ε_+ una vez introducido el conjunto aproximado (ver Fonseca et al. [71]). El paquete *PMCMRplus* realiza, entre otros, los contrastes de hipótesis anteriormente descritos (ver Pohlert y Pohlert [147]).

UNIVERSIDAD DE MÁLAGA

[6]Proyecto R para computación estadística, `https://www.r-project.org/`.

UNIVERSIDAD
DE MÁLAGA

Parte II

Propuestas algorítmicas

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Capítulo 4

Algoritmo *TPA*

El presente capítulo recoge la primera de las propuestas algorítmicas presentadas en esta tesis. Presentaremos un algoritmo exacto capaz de resolver problemas combinatorios multiobjetivo, proporcionando en cualquier instante de tiempo un frente de Pareto parcial bien distribuido por el espacio objetivo. Para una mejor comprensión de este trabajo, dividiremos el capítulo en cinco secciones. La primera sección analizará un *framework* basado en un algoritmo genérico que resuelve este tipo de problemas. Además de servir como base de nuestra propuesta, se determinarán y concretarán las características más importantes que se deberían tener en cuenta para poder conseguir el objetivo deseado: que el algoritmo sea *anytime*. En la segunda sección presentaremos una detallada revisión de la literatura en este campo. Se mencionan algunos de los algoritmos exactos relacionados y se determinan cuales de ellos pueden ser considerados como *anytime* para incluirlos en la comparativa. Se eligen dos algoritmos como los más apropiados para añadir a la comparativa, ya que sus soluciones muestran una buena dispersión y, además, estos métodos no han sido aún superados por otros algoritmos. La tercera sección muestra la descripción del nuevo algoritmo, llamado *TPA*. Se han tratado de formalizar todos los conceptos dentro de un marco riguroso a la vez que comprensible, incluyendo las correspondientes definiciones y proposiciones, más un teorema final que demuestra el funcionamiento correcto del algoritmo. En la cuarta sección mostramos un detallado estudio computacional y analizamos los resultados y su comparativa con los otros dos algoritmos, incluyendo la correspondiente validación estadística. Finalizaremos el capítulo con una sección dedicada a las conclusiones y trabajo futuro.

4.1. Algoritmo genérico para problemas MOCO

Comenzaremos analizando el algoritmo que calcula el frente de Pareto completo en un problema MOCO (ver Algoritmo 1 del Capítulo 3). Intentaremos adaptarlo para que sea *anytime*, en el sentido de que las soluciones estén bien distribuidas por el espacio objetivo en cualquier momento en que deseemos detener la ejecución del mismo. Analizando el pseudocódigo de dicho algoritmo deducimos que hay cuatro aspectos importantes que se deben concretar:

- ¿Qué programa matemático debe resolver el resolutor cuando se analice una caja?
- ¿Qué criterio usar a la hora de seleccionar la siguiente caja a explorar?
- Cuando encontramos una solución, ¿cómo partir las cajas en el espacio objetivo de manera que nunca más se encuentre esa misma solución y además no queden excluidas zonas que pudiesen contener a las restantes soluciones?
- ¿Cómo eliminar zonas de búsqueda redundantes para mejorar la eficiencia?

En lo sucesivo iremos respondiendo a todas estas cuestiones, y tendremos de esta forma definido un algoritmo concreto (no genérico), que aplicaremos mediante un caso práctico.

En relación al programa matemático, una posible opción es usar el método híbrido de la Ecuación (3.3), ya que, como vimos, cumple con los requisitos de identificar fácilmente regiones de búsqueda o cajas en función de los parámetros asociados a las restricciones del modelo.

Respecto al criterio a utilizar a la hora de seleccionar la siguiente caja a explorar, una posible forma sería la de seleccionar dicha caja de manera aleatoria. Otra opción sería la de explorar siempre la caja con mayor volumen de entre todas las cajas existentes en el conjunto $\mathcal{U}$.

UNIVERSIDAD DE MÁLAGA

Con este criterio, explorando antes regiones más amplias, previsiblemente será más probable encontrar una nueva solución que podría no existir en cajas con volumen menor. Recordemos que en este caso las cajas tienen siempre la misma cota inferior, definida por el punto ideal, por lo que se puede calcular el volumen de la caja como $v([z^I, u]) = \prod_{i=1}^{p}(u_i - z_i^I)$. Se entiende con esta definición que si alguna de las coordenadas coincide con su correspondiente coordenada del punto ideal, su volumen será 0. Obviamente, se puede redefinir este concepto de volumen, pero mantendremos esta propuesta inicial.

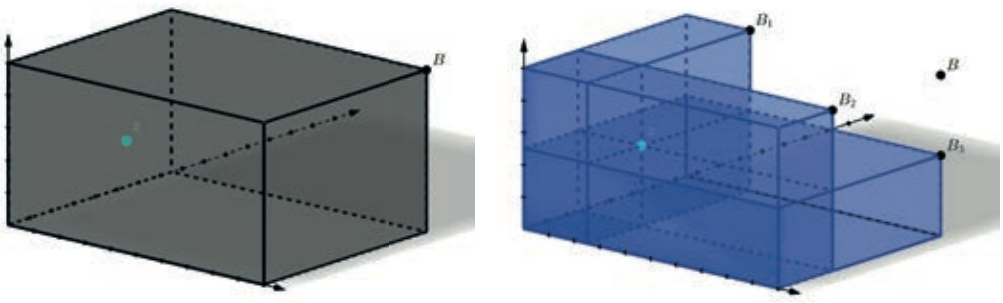
Una vez explorada una caja B y encontrada una solución, z, la forma más usual de partir el espacio objetivo se cononce como *full p-split* y puede verse en el trabajo de Dächert y Klamroth [36]. Consiste en generar p nuevas cajas a partir del punto z, de manera que todos los puntos dominados por z (incluido él mismo) queden excluidos.

Definición 4.1. Dada una caja $B = [z^I, u]$ y un punto $z < u$, decimos que aplicamos *full p-split* cuando la caja B se parte en el conjunto de cajas definido por $\{B_i\}_{i=1}^{p}$, con cota superior $\{u^i\}_{i=1}^{p}$, siendo

$$u^i = (u_1, \ldots, u_{i-1}, z_i, u_{i+1}, \ldots, u_p) \quad \forall i = 1, \ldots, p,$$

y con cota inferior z^I. Cada caja B_i con cota superior u^i se dice que tiene *dirección i*.

Obsérvese que en la definición anterior hemos exigido que $z < u$, lo cual siempre ocurrirá porque al explorar una caja la solución nunca quedará fuera de ella (teniendo en cuenta que la cota inferior de la caja es el punto ideal). La figura 4.1 muestra un ejemplo gráfico de aplicación de *full p-split* para la caja $B = [20, 15, 10]$, suponiendo que la solución encontrada es $z = (5, 5, 5)$. Vemos que el espacio de búsqueda resultante ocupa un volumen menor, y además quedan excluidos todos los puntos dominados por z. Más aún, de acuerdo con la definición, solo quedan excluidos los puntos dominados por z, por lo que los potenciales puntos no dominados del problema quedarán incluidos en alguna/s de las cajas resultantes.



(a) Caja $\mathbf{B} = [(0, 0, 0), (20, 15, 10)]$ con punto no dominado $z = (5, 5, 5)$.

(b) Tras aplicar *full p-split*: $\mathbf{B_1} = [(5, 15, 10)]$, $\mathbf{B_2} = [(20, 5, 10)]$, y $\mathbf{B_3} = [(20, 15, 5)]$. Todas las cotas inferiores son $z^I = (0, 0, 0)$.

Figura 4.1: Partición de una caja usando *full p-split*.

Adviértase además que las cajas resultantes no son disjuntas, por lo que al explorar distintas cajas podríamos encontrar la misma solución repetidas veces. Esto se puede evitar si a la hora de actualizar el conjunto $\mathcal{U}$ aplicamos *full p-split* a todas las cajas que contengan a dicha solución z.

Por último, veamos algún criterio para eliminar cajas redundantes. Decimos que una caja es redundante si está contenida completamente en otra. Esto implica que sería suficiente con explorar la caja continente, ya que también incluiría todos los posibles puntos no dominados de la caja contenida. Programar esto es intuitivamente sencillo. Bastaría con comparar cada una de las cajas del conjunto $\mathcal{U}$ con las restantes, y eliminar aquellas cuya cota superior domina a la cota superior de otra caja (recordemos nuevamente que la cota inferior es siempre la misma). Klamroth et al. [103] diseñaron un algoritmo eficiente en el que solo era necesario comprobar la dominancia entre cajas que tuviesen la misma dirección. Este algoritmo se conoce como RE (*Redundancy Elimination*) y mejora considerablemente la eficiencia respecto de la propuesta inicial. Mostramos su pseudocódigo en el Algoritmo 2.

UNIVERSIDAD DE MÁLAGA

Algoritmo 2 RE (*Redundancy Elimination*)

Entrada: $\mathcal{U}, z$ // Conjunto de cajas y solución encontrada
Salida: $\mathcal{U}$ // Nuevo conjunto de cajas
1: $\mathfrak{R} \leftarrow \{u \in \mathcal{U} : z < u\}$ // Cajas que contienen a z
2: **para** $j \in \{1, \ldots, p\}$ **hacer**
3: $\mathfrak{D}_j \leftarrow \{u \in \mathcal{U} : z_j = u_j \text{ y } z_k < u_k, \forall k \neq j\}$
4: $\mathfrak{P}_j \leftarrow \emptyset$
5: **fin para**
6: **para** $u \in \mathfrak{R}$ **hacer**
7: **para** $j \in \{1, \ldots, p\}$ **hacer**
8: $\mathfrak{P}_j \leftarrow \mathfrak{P}_j \bigcup \{(u_1, \ldots, u_{j-1}, z_j, u_{j+1}, \ldots, u_p)\}$
9: **fin para**
10: **fin para**
11: **para** $j \in \{1, \ldots, p\}$ **hacer**
12: $\mathfrak{P}_j \leftarrow \{z' \in \mathfrak{P}_j : z' \not\leq u', \forall u' \in \mathfrak{P}_j \bigcup \mathfrak{D}_j - \{z'\}\}$ // Filtrar cajas redundantes
13: **fin para**
14: $\mathcal{U} \leftarrow (\mathcal{U} - \mathfrak{R}) \bigcup \left(\bigcup_{j=1}^{p} \mathfrak{P}_j\right)$

El Algoritmo 2 comienza obteniendo el conjunto de cajas que contiene a la nueva solución z (Línea 1). Éstas serán todas las cajas que habrá que partir usando *full p-split* (bucle de Líneas 6–10). Los autores añaden un bucle más en las Líneas 2–5, donde definen conjuntos en los que el punto z se encuentra en la frontera superior de alguna de las cajas. Nótese que las cajas $\mathfrak{D}_j$ no son nunca cajas que están en $\mathfrak{R}$, pero influirán a la hora de filtrar las cajas redundantes, que se muestra en el bucle de las Líneas 11–13. Obsérvese que el filtrado se realiza solo comparando cajas que tienen la misma dirección, de ahí que durante el algoritmo se definan distintos conjuntos $\mathfrak{P}_j$ y $\mathfrak{D}_j$, eliminando así comparaciones innecesarias entre cajas que sabemos que no van a ser redundantes entre ellas y mejorando así la eficiencia. El nuevo conjunto de cajas (Línea 14) será aquel que, tras eliminar las cajas que contenían a z, se le añade el conjunto de cajas obtenido tras las sucesivas aplicaciones de *full p-split* y después de eliminar las cajas redundantes. Abusando de la notación en la Línea 14, podemos considerar que la operación $\mathcal{U} - \mathfrak{R}$ realiza primero la conversión de la caja $[u]$ al vector u, y que el conjunto de vectores final se vuelve a transformar en un conjunto de cajas cuyas cotas superiores son dichos vectores. Veamos un pequeño ejemplo ilustrativo de aplicación del algoritmo RE.

Ejemplo 4.1. Imaginemos que en un momento dado de la ejecución del Algoritmo 1, tenemos que $\mathcal{U} = \{B_1, B_2, B_3, B_4\}$, siendo $B_1 = [(6, 9, 5)]$, $B_2 = [(3, 8, 7)]$, $B_3 = [(3, 9, 6)]$, y $B_4 = [(8, 5, 8)]\}$, y que después de analizar la caja B_2 obtenemos el punto no dominado $z = (2, 2, 6)$. Cuando llamamos al método RE (Algoritmo 2), primero calculamos las cajas que contienen a z, por lo que $\mathfrak{R} = \{B_2, B_4\}$. El primer bucle inicializa los conjuntos $\mathfrak{P}_1 = \mathfrak{P}_2 = \mathfrak{P}_3 = \emptyset$ y calcula los conjuntos $\mathfrak{B}_j$. En este caso $\mathfrak{D}_1 = \mathfrak{D}_2 = \emptyset$ y $\mathfrak{D}_3 = \{B_3\}$. El segundo bucle aplica *full p-split* a cada una de las cajas de $\mathfrak{R}$ y va actualizando los conjuntos $\mathfrak{P}_j$. Así, tras considerar B_2, tenemos que $\mathfrak{P}_1 = \{(2, 8, 7)\}$, $\mathfrak{P}_2 = \{(3, 2, 7)\}$ y $\mathfrak{P}_3 = \{(3, 8, 6)\}$. Al aplicar *full p-split* a la caja B_4 tenemos que $\mathfrak{P}_1 = \{(2, 8, 7), (2, 5, 8)\}$, $\mathfrak{P}_2 = \{(3, 2, 7), (8, 2, 8)\}$ y $\mathfrak{P}_3 = \{(3, 8, 6), (8, 5, 6)\}$. En el último bucle de RE buscamos redundancia entre cajas, pero solo entre aquellas que posean la misma dirección. Así, comparamos las cajas de $\mathfrak{P}_1$ con las de $\mathfrak{P}_1 \bigcup \mathfrak{D}_1$, obteniendo que no hay ninguna relación de dominancia. Analizando la segunda dirección vemos que $(3, 2, 7) \leq (8, 2, 8)$ por lo que eliminamos la caja redundante y $\mathfrak{P}_2$ pasa a ser $\mathfrak{P}_2 = \{(8, 2, 8)\}$. Para la última dirección no hay redundancia entre las cajas de $\mathfrak{P}_3$, pero sí la hay si lo comparamos con $\mathfrak{D}_3$, ya que $(3, 8, 6) \leq (3, 9, 6)$. Así que $\mathfrak{P}_3$ se actualiza a $\mathfrak{P}_3 = \{(8, 5, 6)\}$ y el nuevo conjunto de cajas a analizar será $\mathcal{U} = \{B_1, B_3, [(2, 8, 7)], [(2, 5, 8)], [(8, 2, 8)], [(8, 5, 6)]\}$.

El crecimiento del número de cajas a explorar durante la ejecución del Algoritmo 1 podría ser exponencial, incluso en el caso de usar el algoritmo RE para eliminar redundancias. De hecho, el trabajo de Laumanns et al. [111] estableció un método con una cota máxima de $(|PF| + 1)^{p-1}$ llamadas al resolutor para generar el frente completo. Aunque los trabajos de Ralphs et al. [154] para $p = 2$ y de Dächert y Klamroth [36] para $p = 3$ muestran unas

UNIVERSIDAD DE MÁLAGA

cotas de $|PF|+1$ y $2|PF|-1$ llamadas al resolutor, respectivamente, es todavía un problema abierto determinar si el número de llamadas al resolutor puede ser acotado linealmente en función del número de puntos no dominados del frente de Pareto para cualquier dimensión.

En el trabajo de Klamroth et al. [103] se establece también otro método distinto a RE para detectar cajas redundantes, pero en lugar de eliminarlas, evita que se generen. Esta técnica se conoce como RA (*Redundancy Avoidance*). Más recientemente, Dächert et al. [37] desarrollaron un estudio teórico sobre las propiedades estructurales de las regiones de búsqueda en el espacio objetivo y consiguieron mejorar los métodos RE y RA para dimensiones $p \geq 6$. Para más información al respecto el lector puede remitirse a las referencias indicadas.

Una vez han sido comentadas las distintas características fundamentales a tener en cuenta en el algoritmo genérico que resuelve problemas MOCO, mostramos un resumen de estas en la Figura 4.2, y su pseudocódigo en el Algoritmo 3.

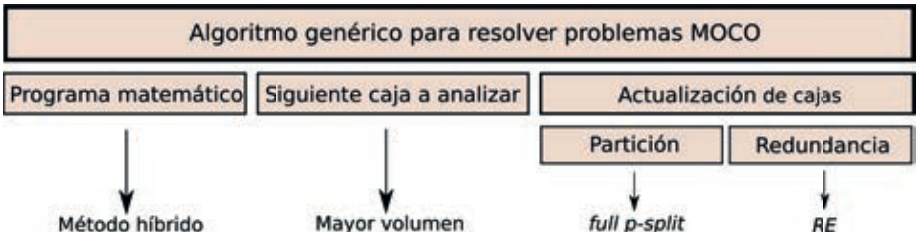


FIGURA 4.2: Definición de características asociadas al algoritmo genérico que resuelve problemas MOCO.

El algoritmo de Holzmann y Smith [91] utiliza una propuesta similar a la de la Figura 4.2. En su caso, el programa matemático usado es sustituido por un modelo de Thebycheff aumentado y ponderando también el término asociado a la norma $\|\cdot\|_1$. Para la técnica de filtración de regiones de búsqueda redundantes no queda muy claro si se realiza utilizando el algoritmo RE o la propuesta de Dächert et al. [37], ya que no ha sido posible obtener el código fuente desde el material suplementario proporcionado por los autores de este trabajo. Comentaremos esta propuesta algorítmica en la Sección 4.3.

Algoritmo 3 Resolutor de problemas MOCO usando *full p-split*

Entrada: $\lambda_i, \quad \forall i = 1, \ldots, p$
Salida: PF
1: $PF = \emptyset$
2: $\mathcal{U} \leftarrow [\overline{z}^N]$
3: $P(u) \equiv \min\limits_{x \in \mathcal{X} \wedge f(x) \leq u} \left(\sum\limits_{i=1}^{p} \lambda_i f_i(x) \right)$
4: **mientras** $(\mathcal{U} \neq \emptyset)$ **hacer**
5: $\quad B \leftarrow \text{argmax}_{B \in \mathcal{U}} (v(B))$ $\quad$ //$v(B)$ denota el volumen de la caja B
6: $\quad$ **si** *(P(B.u) tiene solución)* **entonces**
7: $\quad\quad x^* \leftarrow$ Solución óptima de $P(B.u)$
8: $\quad\quad PF = PF \cup \{f(x^*)\}$
9: $\quad\quad \mathcal{U} \leftarrow \text{RE}(\mathcal{U}, f(x^*))$
10: $\quad$ **si no**
11: $\quad\quad \mathcal{U} \leftarrow \mathcal{U} - \{B\}$
12: $\quad$ **fin si**
13: **fin mientras**

En el Algoritmo 3 declaramos como parámetros de entrada los pesos a considerar en la parametrización de objetivos del método híbrido. Estos valores, que se han determinado como fijos durante la ejecución, se podrían haber considerado también como interactivos o variables. Por otro lado, aunque la salida final es el conjunto PF, se podría adaptar dicho pseudocódigo fácilmente para obtener también una solución eficiente asociada a cada punto no dominado.

Finalicemos este apartado analizando el Algoritmo 3 con un ejemplo práctico de un problema de asignación multiobjetivo para $p = 3$.

UNIVERSIDAD DE MÁLAGA

Ejemplo 4.2. Basándonos en la definición dada por la Ecuación (2.18), consideremos el siguiente problema de asignación triobjetivo para 5 agentes y 5 tareas.

$$\begin{aligned} \text{mín} \quad & f(x) = (C_1x,\, C_2x,\, C_3x)\,, \\ s.a. \quad & \sum_{j=1}^{5} x_{ij} = 1 \quad \forall i = 1,\ldots,5, \\ & \sum_{i=1}^{5} x_{ij} = 1 \quad \forall j = 1,\ldots,5, \\ & x_{ij} \in \{0,1\} \quad \forall i,j = 1,\ldots,5. \end{aligned}$$

Los costes asociados a cada agente y tarea vienen definidos mediante tres matrices, cada una de ellas referida a una función objetivo. De esta forma, fijado un objetivo concreto, el valor c_{ij} indica el coste asociado a que el agente i realice la tarea j:

$$C_1 = \begin{pmatrix} 8 & 14 & 11 & 6 & 20 \\ 16 & 8 & 8 & 1 & 5 \\ 3 & 8 & 20 & 15 & 7 \\ 7 & 14 & 9 & 17 & 18 \\ 11 & 20 & 20 & 20 & 8 \end{pmatrix}, \quad C_2 = \begin{pmatrix} 2 & 1 & 7 & 16 & 12 \\ 13 & 3 & 6 & 3 & 9 \\ 5 & 19 & 16 & 13 & 20 \\ 1 & 15 & 8 & 1 & 10 \\ 14 & 8 & 3 & 2 & 4 \end{pmatrix}, \quad C_3 = \begin{pmatrix} 1 & 12 & 4 & 20 & 12 \\ 11 & 2 & 13 & 18 & 18 \\ 5 & 10 & 20 & 10 & 13 \\ 9 & 15 & 12 & 4 & 8 \\ 11 & 5 & 2 & 18 & 5 \end{pmatrix}.$$

A la hora de codificar una solución en el espacio de decisiones, en lugar de utilizar una matriz cuadrada de orden 5 llena de ceros y unos (o equivalentemente un vector binario de longitud 25), resulta más sencillo utilizar el vector $\varphi = (\varphi_1, \ldots, \varphi_5)$, donde φ_i indica la tarea asignada al agente i. Esta forma simplificada de representación es posible ya que existe una biyección entre los agentes y las tareas, por tanto las distintas soluciones factibles representarán las distintas permutaciones del conjunto $\{1,2,3,4,5\}$. Por ejemplo, la solución $\varphi = (1,3,4,5,2)$ indica que el agente 1 tiene asignada la tarea 1, el agente 2 tiene asignada la tarea 3, y así sucesivamente hasta el agente 5, que tiene asignada la tarea 2. Para determinar el coste asignado al primer objetivo de esta solución, solo hay que mirar los elementos de la matriz C_1 teniendo en cuenta la permutación. Así, si x fuese el vector de decisiones, tendríamos que $f_1(x) = 8 + 8 + 15 + 18 + 20 = 69$. Repitiendo el proceso para los dos objetivos restantes, obtendríamos que $f(x) = (f_1(x), f_2(x), f_3(x)) = (69, 39, 37)$. Como veremos, esta solución no es eficiente, pues se puede encontrar otra que la domina.

La traza obtenida al aplicar el Algoritmo 3 puede verse en la Tabla 4.1. Para facilitar la comprensión de dicha traza se incluye en la Figura 4.3 el grafo generado durante la ejecución, donde cada nodo representa una caja (que podría ser vacía), y cada arista une dos cajas que se relacionan de tal forma que una de ellas se ha obtenido tras aplicar *full p-split* a la otra. Finalmente, el frente de Pareto viene dado por $PF = \{(47,20,20),\ (34,36,44),\ (35,21,52),\ (50,17,39),\ (35,34,46)\}$.

Podemos observar que la ejecución del Ejemplo 4.2 requiere un total de 12 llamadas al resolutor, sin contar las tres llamadas necesarias para obtener el punto ideal, que en este caso es $z^I = (34,17,20)$, y otras tres llamadas para obtener la cota estrictamente superior del punto nadir, $\overline{z}^N = (94,72,85)$. Este punto nos sirve como referencia de la caja inicial B_0. La caja seleccionada en cada iteración es siempre la de mayor volumen. Es por ello que los valores de los volúmenes aparecen en orden no creciente, como puede apreciarse en la columna correspondiente. Los puntos del frente de Pareto obtenidos son los que realmente proporciona el resolutor CPLEX 12.6.2. durante la ejecución. Las nuevas cajas que se obtienen al aplicar *full p-split* se muestran en la última columna. Algunas de ellas son vacías ya que tienen una de sus coordenadas igual a la correspondiente coordenada del punto ideal (véase la Definición 3.2), y por ello no se tienen en cuenta estas cajas en la iteración siguiente. Una de las cajas es eliminada durante la ejecución, concretamente en la iteración 6, donde B_{13} se descarta tras aplicar RA, pues es redundante respecto de B_7.

UNIVERSIDAD DE MÁLAGA

Tabla 4.1: Traza del Ejemplo 4.2. Problema de asignación triobjetivo.

	$\mathcal{U}$	Siguiente caja	Solución	Nuevas cajas
1	B_0	B_0, $v(B_0) = 214500$	$z^1 = (47, 20, 20)$	$B_1 = [(47, 72, 85)]$ $B_2 = [(94, 20, 85)]$ $B_3 = [(94, 72, 20)] = \emptyset$
2	B_1, B_2	B_1, $v(B_1) = 46475$	$z^2 = (34, 36, 44)$	$B_4 = [(34, 72, 85)] = \emptyset$ $B_5 = [(47, 36, 85)]$ $B_6 = [(47, 72, 44)]$
3	B_2, B_5, B_6	B_6, $v(B_6) = 17160$	$\emptyset$	-
4	B_2, B_5	B_5, $v(B_5) = 16055$	$z^3 = (35, 21, 52)$	$B_7 = [(35, 36, 85)]$ $B_8 = [(47, 21, 85)]$ $B_9 = [(47, 36, 52)]$
5	B_2, B_7, B_8, B_9	B_2, $v(B_2) = 11700$	$z^4 = (50, 17, 39)$	$B_{10} = [(50, 20, 85)]$ $B_{11} = [(94, 17, 85)] = \emptyset$ $B_{12} = [(94, 20, 39)]$
6	B_7, B_8, B_9, B_{10}, B_{12}	B_9, $v(B_9) = 7904$	$z^5 = (35, 34, 46)$	~~$B_{13} = [(35, 36, 52)]$~~ $B_{14} = [(47, 34, 52)]$ $B_{15} = [(47, 36, 46)]$
7	B_7,B_8,B_{10},B_{12},B_{14},B_{15}	B_{14}, $v(B_{14}) = 7072$	$\emptyset$	-
8	B_7, B_8, B_{10}, B_{12}, B_{15}	B_{15}, $v(B_{15}) = 6422$	$\emptyset$	-
9	B_7, B_8, B_{10}, B_{12}	B_{12}, $v(B_{12}) = 3420$	$\emptyset$	-
10	B_7, B_8, B_{10}	B_8, $v(B_8) = 3380$	$\emptyset$	-
11	B_7, B_{10}	B_{10}, $v(B_{10}) = 3120$	$\emptyset$	-
12	B_7	B_7, $v(B_7) = 1235$	$\emptyset$	-

4.2. Revisión de la literatura

Dado que el objetivo principal de este capítulo es el de presentar un algoritmo exacto *anytime* que resuelva problemas MOCO, realizamos una revisión en la literatura de los algoritmos exactos que son o pudieran servir como algoritmos *anytime*. No pretendemos realizar un estudio exhaustivo, así que se señalarán los trabajos relevantes de mayor impacto. Después de la revisión, tomaremos aquellos algoritmos *anytime* más 'eficientes', que serán los que compararemos con nuestra propuesta de la Sección 4.3. Para una lectura algo más amena, dividiremos esta revisión en varios pequeños subapartados. En cada uno de ellos se citarán los trabajos relacionados en orden cronológico, en la medida de lo posible.

Primeros algoritmos exactos y evolución

En el año 2004 se publicó el trabajo de Sylva y Crema [165], que estaba basado en la idea de Klein y Hannan [105]. Sylva y Crema diseñaron un nuevo método para resolver problemas MOILP. Consiguieron dividir el espacio de búsqueda de dimensión p mediante la adición de p variables binarias y p restricciones, que excluían los puntos no dominados previamente generados. Alrededor de tres años más tarde, Sylva y Crema [166] formularon una nueva variante de su trabajo anterior, encontrando subconjuntos de puntos no dominados en el espacio objetivo que estaban bien distribuidos. Este diseño se realizó para problemas multiobjetivo mixtos. Si las variables eran enteras, el frente de Pareto completo se podía obtener. Probablemente es el primer trabajo que resuelve problemas MOMILP apoyado por resultados computacionales, y que puede considerarse *anytime*.

Masin y Bukchin [128] desarrollaron dos algoritmos para resolver problemas MOMILP, basándose en la maximización de una métrica de diversidad. El primero de ellos, M-DMA (*Diversity Maximization Approach*) garantizaba la obtención del frente completo en el caso de algoritmos combinatorios. El segundo algoritmo, B-DMA, tiene una implementación basada en ramificación y acotación (*Branch & Bound*).

Lokman y Köksalan [119] también propusieron dos algoritmos exactos iterativos capaces de generar el frente completo en MOILP. Uno de ellos iba generando nuevos puntos añadiendo variables y restricciones binarias en cada iteración. Está basado, en cierta manera, en el trabajo de Sylva y Crema [165]. El otro evita generar nuevas variables empleando una técnica de

UNIVERSIDAD DE MÁLAGA

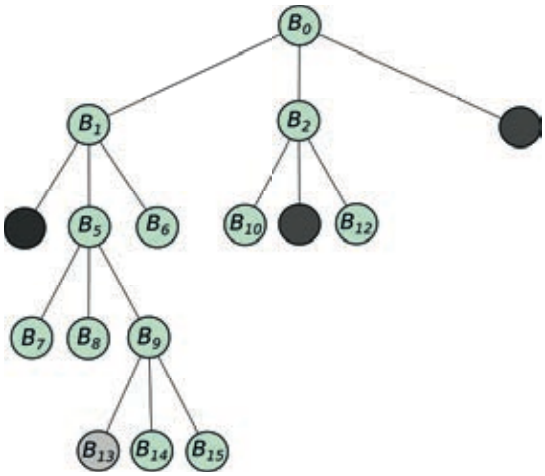


FIGURA 4.3: Grafo asociado a la traza representada en la Tabla 4.1. Los nodos en gris oscuro corresponden a cajas que no han sido creadas, pues están vacías.

búsqueda y resolviendo un número de modelos para encontrar la siguiente solución. En ambos casos se consigue excluir de la búsqueda los puntos no dominados previamente generados.

En el año 2018, Ceyhan et al. [18], basándose en el trabajo de Lokman y Köksalan [119], propusieron varios algoritmos para encontrar un pequeño conjunto representativo y bien disperso de soluciones no dominadas en problemas MOMILP. El primero de los algoritmos, llamado SBA (*Subset-Based Approach*), es capaz de encontrar el frente de Pareto completo si se dispone de suficiente tiempo de ejecución. Los otros dos algoritmos de ese trabajo no están diseñados para calcular el frente de Pareto completo. El algoritmo SBA de Ceyhan et al. consigue mejorar los resultados del algoritmo de Masin y Bukchin. Este último, a su vez, es similar al método utilizado por Sylva y Crema [166], ya que generan subconjuntos de soluciones muy parecidos, como apuntan Ceyhan et al.

Algoritmos basados en recursividad

En el año 2003 Tenfelde-Podehl [169] propuso un algoritmo recursivo para problemas MOCO con p objetivos. Dhaenens et al. [45] propusieron un nuevo método para resolver programación multiobjetivo (MOP, *Multiobjective Programming*), llamado K-PPM (*Parallel Partitioning Method*), que está basado en el trabajo de Lemesre et al. [112], llamado PPM, y que era válido solo para el caso biobjetivo. K-PPM es válido para cualquier dimensión $p \geq 2$, y además puede ser considerado como una extensión de la idea de Tenfelde-Podehl.

Özlen y Azizoğlu [141] desarrollaron un método general para calcular el frente de Pareto en problemas multiobjetivo enteros basándose en el método ε-constraint. Usaron recursividad para reducir el problema a uno que tuviese un número menor de objetivos. Una de las desventajas de su método es que puede generar el mismo punto no dominado muchas veces. Este algoritmo fue mejorado considerablemente en el trabajo de Özlen et al. [142], en el año 2014. Aprovecharon la información obtenida al generar los subproblemas para mejorar considerablemente el tiempo de ejecución empleado en dicho algoritmo recursivo, así como para obtener mejores resultados para dimensiones mayores.

Algoritmos basados en ramificación y acotación

Mavrotas y Diakoulaki [131] fueron los primeros autores en desarrollar un algoritmo basado en ramificación y acotación adaptado para resolver problemas multiobjetivo mixtos donde las variables enteras eran binarias. Nótese que los problemas combinatorios multiobjetivo pueden considerarse como un caso particular. La idea general es enumerar todas las posibles combinaciones de valores para las variables binarias e ir resolviendo distintos subproblemas. Se mantiene una estructura de grafo llamada *árbol combinatorio*. Cada nodo del árbol es una solución parcial (no necesariamente factible), que asigna un valor binario a un conjunto de variables binarias del problema. El nodo raiz es aquel en el que todas las variables binarias son libres (problema completamente relajado). La meta de este tipo de métodos es que el

UNIVERSIDAD DE MÁLAGA

algoritmo diseñado sea capaz de asignar varios valores a vez (en la medida de lo posible), así como de detectar ramas que no proporcionarán una mejora en la solución. Los nodos que son infactibles o que sabemos que no proporcionarán una solución eficiente ya no son explorados más. En estos casos se dice que se ha aplicado un *node fathoming*. Un nodo terminal del árbol combinatorio es una solución parcialmente eficiente (factible). No podemos decir que es eficiente ya que podría ser dominada por otra solución en el futuro. Por ello, es necesario además realizar un filtrado de soluciones dominadas en algún momento de la ejecución. De igual forma que en el trabajo mencionado anteriormente, caben destacar los trabajos de Mavrotas y Diakoulaki [132], que fue corregido y mejorado por Vincent et al. [175], o el trabajo de Stidsen et al. [164]. Mencionamos nuevamente aquí el algoritmo B-DMA de Masin y Bukchin [128], y destacamos también el trabajo de Mavrotas y Florios [133], quienes diseñaron un algoritmo llamado AUGMECON2, que mejoraba al algoritmo AUGMECON [130], desarrollado por Mavrotas. AUGMECON2 ha sido referenciado en numerosas ocasiones. Una aplicación al problema multiobjetivo del viajante de comercio (*traveling salesperson problem*) y al problema de la cobertura (*set covering problem*) puede verse en Florios y Mavrotas [69].

Algoritmos modernos

Laumanns et al. [111, 110] propusieron el primer algoritmo capaz de calcular el frente de Pareto completo para MOP a través de la resolución de un número de problemas monoobjetivo que dependía únicamente del número de elementos del frente de Pareto. Este cambio metodológico aumentó considerablemente la eficiencia de los algoritmos respecto a los desarrollados anteriormente. Uno de sus trabajos se basaba en el método ε-constraint [111]. En el otro, [110], propusieron un algoritmo que calculaba soluciones eficientes mediante el uso de cotas inferiores asociadas al espacio objetivo (en este caso se consideraba la maximización de objetivos). Este último método consideraba un número menor de restricciones que en el primer caso. El algoritmo de Kirlik y Sayın [102] supuso una mejora del trabajo de Laumanns et al. [111]. Ellos cambiaron el orden en que los problemas eran resueltos y gestionaban el uso de rectángulos $(p-1)$-dimensionales. Se empieza a observar como el uso de cajas empieza a aparecer de una forma o de otra en los algoritmos modernos.

Podemos considerar como algoritmos modernos aquellos que han sido desarrollados en la última década. Además del mencionado trabajo de Ceyhan et al. [18], existen otros importantes trabajos como el de Ehrgott y Ruzika [62] o el de Przybylski et al. [150]. El trabajo de Ehrgott y Ruzika, si bien es una técnica de escalarización y no un algoritmo propiamente dicho, desarrolla un método ε-constraint mejorado para programación multiobjetivo, con muy buenos resultados aplicados a un problema de planificación de vuelos biobjetivo (ver [63]). El trabajo de Przybylski et al. [150], si bien podría haber sido incluido en una categoría anterior pues es recursivo, calcula los puntos extremos del frente de Pareto soportado. Junto con la generalización posterior propuesta en [151], en la que obtuvieron el frente de Pareto completo para problemas multiobjetivo enteros, ambos sirvieron de referencia para el desarrollo del algoritmo genérico que aparece en Dächert y Klamroth [36]. El trabajo de Przybylski et al. [151] es una generalización del conocido método de las dos fases de Ulungu y Teghem [172]. En una primera fase se obtiene el frente de Pareto soportado, y en la segunda se exploran las regiones no soportadas del frente. También Özpeynirci y Köksalan [143], entre otros autores, desarrollaron algoritmos que generaban el frente de Pareto soportado para problemas multiobjetivo mixtos. Dächert y Klamroth [36] acuñaron el término *full p-split* que hemos usado en esta tesis a la hora de partir una caja cuando se encuentra una nueva solución dentro de ella. En 2018, Holzmann y Smith [91] diseñaron un método para resolver problemas multiobjetivo discretos usando una suma aumentada y ponderada de la escalarización de Tchebycheff. Basándose en la estructura del algoritmo genérico, consiguieron mejorar los resultados de Kirlik y Sayın [102], y de Özlen et al. [142].

Finalizada la revisión de la literatura, podemos concluir que solo algunos métodos exactos pueden considerarse *anytime*. Así lo manifestaron Ceyhan et al. [18], quienes resumieron en dos el número de algoritmos que podían ser usados como métodos exactos capaces de obtener soluciones bien dispersas por el espacio objetivo [128, 166]. De hecho, su algoritmo SBA ya mejoraba los resultados de [166]. Posteriormente se publicó el trabajo de Holzmann y Smith [91] que también obtiene muy buenos resultados y puede considerarse como *anytime*.

UNIVERSIDAD DE MÁLAGA

Analizamos también qué otros algoritmos existentes en la literatura podríamos incluir en nuestro estudio comparativo. Los algoritmos que usan el método ε-constraint no parecen ser buenos métodos a la hora de dispersar soluciones, porque en la escalarización solo estamos considerando una única función objetivo. Por otro lado, los métodos que usan ramificación y acotación han sido ampliamente mejorados por los algoritmos modernos en cuanto a rendimiento. Los algoritmos recursivos suelen tener una primera fase que calcula el punto nadir, que ya de por sí es un problema de la clase $\mathcal{NP}$-*Hard*, así que también descartamos este tipo de algoritmos en nuestra comparativa. En conclusión, y por todo lo anterior, hemos seleccionado dos algoritmos para comparar los resultados computacionales con nuestra nueva propuesta: SBA, del trabajo de Ceyhan et al. [18], y el algoritmo propuesto por Holzmann y Smith [91].

4.3. Algoritmo *TPA*

En esta sección presentamos un nuevo algoritmo *anytime* que calcula de manera exacta el frente de Pareto de cualquier problema combinatorio multiobjetivo. Su desarrollo se dividirá en tres partes. Inicialmente seleccionaremos el programa matemático que usará el resolutor para el cálculo de cada uno de los puntos eficientes del problema. Después estableceremos las contribuciones o novedades del método. Finalmente mostraremos el pseudocódigo del algoritmo justificando su funcionamiento.

4.3.1. Elección del programa matemático

En la Sección 4.1 ya comentamos un posible programa matemático que sirve como parámetro de entrada en el Algoritmo 1. Hablamos del método híbrido, ilustrado en la Ecuación (3.3). Se ha realizado un estudio preliminar sobre qué otros programas matemáticos se podrían usar y que proporcionasen un mejor rendimiento computacional. Además de considerar el mencionado método híbrido, hemos utilizado una segunda alternativa consistente en usar el algoritmo SBA de Ceyhan et al. [18]. La formulación (adaptada al caso de minimización) es la siguiente:

$$\begin{aligned} &\min_{x\in\mathcal{X}} \left(\varepsilon \sum_{i=1}^{p} w_i f_i(x) - \alpha \right), \\ &s.a. \quad f_i(x) \leq u_i - \alpha, \quad \forall i = 1, \ldots, p, \\ &\qquad \alpha \geq 0. \end{aligned} \tag{4.1}$$

El valor α es una nueva variable que representa la holgura para la cobertura, conocido en inglés como *coverage gap*. El parámetro ε es un valor positivo suficientemente pequeño y $\omega_i > 0$, $\forall i$, son los pesos asociados a cada función objetivo. El vector $u = (u_1, \ldots, u_p)$ representa el espacio de búsqueda, es decir, la cota superior de la caja a explorar.

Proposición 4.1. Si el modelo dado por la Ecuación (4.1) tiene solución óptima x^*, entonces x^* es eficiente.

Demostración. Supongamos que x^* no es eficiente. Entonces existe $\hat{x} \in \mathcal{X}$ tal que $f(\hat{x}) \leq f(x^*)$. Esto significa que $f_i(\hat{x}) \leq f_i(x^*)$, $\forall i = 1, \ldots, p$, y que existe al menos un índice k tal que $f_k(\hat{x}) < f_k(x^*)$. Como $f_i(\hat{x}) \leq f_i(x^*) \leq u_i - \alpha$, $\forall i = 1, \ldots, p$, si consideramos $\hat{\alpha} = \alpha$ tenemos que el par $(\hat{x}, \hat{\alpha})$ es una solución factible para el modelo de la Ecuación (4.1). Sin embargo, $\sum_{i=1}^{p} \omega_i f_i(\hat{x}) < \sum_{i=1}^{p} \omega_i f_i(x^*)$, ya que $\omega_i > 0$, $\forall i$. Como $\varepsilon > 0$, concluimos que $\varepsilon \sum_{i=1}^{p} \omega_i f_i(\hat{x}) - \hat{\alpha} < \varepsilon \sum_{i=1}^{p} \omega_i f_i(x^*) - \alpha$, lo cual es un absurdo, pues hemos encontrado una solución $\hat{x}$ con mejor valor objetivo que el que proporcionaba la solución encontrada, x^*, que se suponía óptima. □

Una tercera alternativa a considerar se basa en una métrica aumentada y ponderada de Tchebycheff, que puede encontrarse en el trabajo de Holzmann y Smith [91]. La formulación viene dada por:

$$\min_{x\in\mathcal{X}} \left(\max_{i=1,\ldots,p} \left(\omega_i \left| f_i(x) - z_i^I \right| \right) + \epsilon \sum_{i=1}^{p} \omega_i \left| f_i(x) - z_i^I \right| \right), \tag{4.2}$$

UNIVERSIDAD DE MÁLAGA

donde el vector $\omega \in \mathbb{R}^p$ verifica $\omega > 0$, el parámetro ϵ es un número positivo, y z^I es el punto ideal del problema. Obsérvese la diferencia entre este modelo y la métrica aumentada de Tchebycheff que presentamos en la Ecuación (3.5). En este caso, el término aumentado correspondiente a la norma $\|\cdot\|_1$ también aparece ponderado por el mismo vector ω que pondera la norma $\|\cdot\|_\infty$.

Proposición 4.2. Si el modelo dado por la Ecuación (4.2) tiene solución óptima x^*, entonces x^* es eficiente.

Demostración. Utilizando un razonamiento similar al de la Proposición 4.1, supongamos por reducción al absurdo que x^* no es eficiente. Entonces existe $\hat{x} \in \mathcal{X}$ tal que $f(\hat{x}) \leq f(x^*)$. Esto significa que $f_i(\hat{x}) \leq f_i(x^*)$, $\forall i = 1, \ldots, p$, y que existe al menos un índice k tal que $f_k(\hat{x}) < f_k(x^*)$. Dado que z^I es el punto ideal, $|f_i(x) - z_i^I| = f_i(x) - z_i^I$, $\forall i$, por lo que podemos prescindir del valor absoluto. Dado que $\omega > 0$ y que $\epsilon > 0$, el término correspondiente a la norma $\|\cdot\|_\infty$ solo nos permite garantizar que $\max_{i=1}^p \left(\omega_i(f_i(\hat{x}) - z_i^I)\right) \leq \max_{i=1}^p \left(\omega_i(f_i(x^*) - z_i^I)\right)$. Sin embargo, en el término asociado a la norma $\|\cdot\|_1$ podemos garantizar la desigualdad estricta. Uniendo ambas expresiones, concluimos que

$$\max_{i=1,\ldots,p} \left(\omega_i |f_i(\hat{x}) - z_i^I| + \epsilon \sum_{i=1}^p \omega_i \left|f_i(\hat{x}) - z_i^I\right|\right) < \max_{i=1,\ldots,p} \left(\omega_i |f_i(x^*) - z_i^I| + \epsilon \sum_{i=1}^p \omega_i \left|f_i(x^*) - z_i^I\right|\right),$$

lo cual contradice que x^* es óptimo, pues el valor objetivo para $\hat{x}$ es menor. □

A continuación mostramos cómo definir apropiadamente los parámetros del modelo de la Ecuación (4.2) para poder asociarlos con regiones de búsqueda en el espacio objetivo.

Proposición 4.3. Consideramos un modelo combinatorio multiobjetivo y el modelo aumentado y ponderado de Tchebycheff dado por la Ecuación (4.2). Definimos el vector ω tal que $\omega_i = 1/\max\{1, u_i - z_i^I\}$, $\forall i$, y el parámetro $\epsilon = 1/(2p(r-1))$, siendo $r = \max_{i=1}^p(\overline{z}_i^N - z_i^I)$, donde u es la cota superior de la región de búsqueda, z^I es el punto ideal y $\overline{z}^N$ es una cota estrictamente superior del punto nadir tal que $\overline{z}_i^N = \max_{x \in \mathcal{X}}(f_i(x)) + 1$, $\forall i$. Se verifica que:
i) ϵ está bien definido y el modelo es siempre factible, salvo en el caso en que $PF = \emptyset$.
ii) Si el valor objetivo es menor que 1 y $z^I < u$, entonces la solución está en la caja $B = [u]$. En caso contrario, la caja B no contiene ningún punto no dominado.

Demostración. En primer lugar observemos que $r \geq 1$, por su propia definición, y que ϵ estará bien definido cuando $r \neq 1$. Pero $r = 1$ si y solo si el espacio objetivo está formado por un único punto. Si todos los valores factibles tienen el mismo valor objetivo no hay ningún problema que resolver. Por otro lado podría ocurrir que no hubiese ninguna solución. Esto se detectaría al calcular z^I o $\overline{z}^N$, y no habría que ejecutar el modelo de Tchebycheff aumentado y ponderado. En la Sección 2 de [91], y dado que este modelo es combinatorio, se puede concluir que el modelo es factible siempre que exista al menos un punto no dominado en el espacio objetivo.

Probemos ahora la segunda parte. Si el valor objetivo es menor que 1 y $z^I < u$, por el Teorema 5 de [91] concluimos que la solución está en la caja $B = [u]$. En caso contrario pueden ocurrir dos cosas: 1) que $z^I \not< u$, pero como $z^I \leq u$, esto significa que existe un índice k tal que $z_k^I = u_k$, por lo que la caja sería vacía; o 2) que $z^I < u$, pero el valor de la función objetivo es mayor o igual que uno, entonces nuevamente el Teorema 5 de [91] garantizaría que la caja está vacía. □

La Proposición 4.3 nos proporciona una forma de adaptar el programa dado por la Ecuación (4.2) al algoritmo genérico. Tras una experimentación preliminar con algunas instancias multiobjetivo, y utilizar este programa, más el método híbrido, más el de la Ecuación (4.1), observamos un mejor rendimiento para el método aumentado y ponderado de Tchebycheff de la Ecuación (4.2). Este será, por tanto, el programa matemático que utilizaremos en nuestra propuesta algorítmica.

4.3.2. Contribuciones

En este apartado mostramos las contribuciones que forman parte de nuestro algoritmo *anytime* que resuelve problemas MOCO. Basándonos en el modelo matemático del trabajo

UNIVERSIDAD DE MÁLAGA

de Holzmann y Smith, así como en el algoritmo genérico propuesto por Dächert y Klamroth, presentamos tres nuevas contribuciones que detallamos en los siguientes apartados.

I. Nueva forma de seleccionar la siguiente caja a explorar

Partiremos del supuesto de que existe una función de *prioridad* y de que cada caja tiene asignado un valor real según esta función, que determina en qué orden serán seleccionadas durante la ejecución del algoritmo (las cajas con mayor prioridad serán seleccionadas primero). Un ejemplo de prioridad de una caja podría ser su volumen, $B.priority = v(B)$.

Proponemos una nueva estrategia de selección de la siguiente caja a explorar en cada iteración en un intento de diversificar la búsqueda por el espacio objetivo. Definimos p listas de cajas, $\mathfrak{L}_i$, $\forall i = 1, \ldots, p$, con la propiedad de que la lista $\mathfrak{L}_i$ solo contiene cajas con dirección i (ver Definición 4.1). Dentro de cada lista las cajas están ordenadas según su prioridad en orden no creciente. Durante la ejecución del algoritmo iremos alternando direcciones, así que seleccionamos la caja de mayor prioridad para un cierto $\mathfrak{L}_i$ y en la siguiente iteración haremos lo mismo para la caja de mayor prioridad en $\mathfrak{L}_{i+1}$. Si alguna lista de cajas es vacía, pasamos a la siguiente. Después de explorar $\mathfrak{L}_p$ (última lista), volvemos a la lista $\mathfrak{L}_1$. El pseudocódigo que realiza esta selección puede verse en el Algoritmo 4.

Algoritmo 4 *Select_ next_ box*

Entrada: $\{\mathfrak{L}_i\}_{i=1}^p$, k // Listas de cajas y contador entero
Salida: (B, k)
1: $B = \emptyset$
2: **si** $\exists\, i : |\mathfrak{L}_i| > 0$ **entonces**
3: $k = (k \mod p) + 1$
4: **mientras** $|\mathfrak{L}_k| = 0$ **hacer**
5: $k = (k \mod p) + 1$
6: **fin mientras**
7: $B \leftarrow \operatorname{argmax}_{B \in \mathfrak{L}_k} (B.priority)$
8: **fin si**

II. Nueva forma de partir las cajas: *p-partition*

La segunda contribución es una nueva forma de partir las cajas distinta de *full p-split*, y que llamamos *p-partition*. La diferencia principal entre *p-partition* y *full p-split* es que en esta nueva propuesta al partir una caja, las nuevas cajas serán disjuntas dos a dos.

Definición 4.2. Sea $B = [l, u]$ una caja y $z \in \mathbb{R}^p$. Para cada $i = 1, \ldots, p$, definimos

$$B_i(z) = \{x \in B \mid x_i < z_i \text{ y } x_j \geq z_j,\ \forall j > i\}.$$

Definimos también $B_0(z)$=$\{x \in B \mid x_j \geq z_j,\ \forall j\}$. Si $B_i(z) \neq \emptyset$, entonces decimos que tiene *dirección* i.

Se comprueba fácilmente que $B_i(z)$ son también cajas, que se expresan de la siguiente forma:

$$B_i(z) = [(l_1, \ldots, l_i, \hat{z}_{i+1}, \ldots, \hat{z}_p), (u_1, \ldots, u_{i-1}, \hat{z}_i, u_{i+1}, \ldots, u_p)], \tag{4.3}$$

donde $\hat{z}_i = \text{máx}\{z_i, l_i\}$, $\forall i = 2, \ldots, p$. Nótese que las cajas también pueden representar el conjunto vacío. Esto ocurrirá cuando $l_i \geq u_i$ para algún i. Un caso extremo se produce cuando $z < l$, en cuyo caso $B_0(z) = B$, y $B_i(z) = \emptyset$, $\forall i = 1, \ldots, p$.

Proposición 4.4. Dada una caja $B = [l, u] \neq \emptyset$ y un vector $z < u$, se verifica que $B_0(z) \neq \emptyset$.

Demostración. Consideremos el vector x tal que $x_i = \frac{\text{máx}\{l_i, z_i\} + u_i}{2}$, $\forall i$. Por un lado tenemos que

$$l_i = \frac{l_i + l_i}{2} < \frac{l_i + u_i}{2} \leq \frac{\text{máx}\{l_i, z_i\} + u_i}{2} = x_i < \frac{u_i + u_i}{2} = u_i,\ \forall i,$$

UNIVERSIDAD DE MÁLAGA

por tanto, $x \in B$. Por otro lado,

$$x_i = \frac{\text{máx}\{l_i, z_i\} + u_i}{2} \geq \frac{z_i + u_i}{2} > \frac{z_i + z_i}{2} = z_i, \, \forall i,$$

así que $x \in B_0(z)$, por lo que $B_0(z) \neq \emptyset$. □

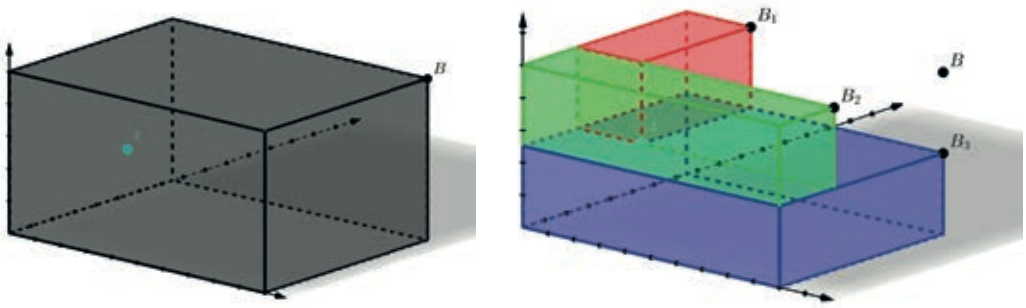
Proposición 4.5. Dada una caja B y $z \in \mathbb{R}^p$, entonces $\{B_i(z)\}_{i=0}^p$ es una partición de B.

Demostración. Tenemos que demostrar que $B_i(z) \cap B_j(z) = \emptyset$, $\forall i \neq j$, y que $\cup_{i=0}^p B_i(z) = B$. Supongamos que $x \in B_i(z) \cap B_j(z)$ para índices i, j con $i < j$. Como $x \in B_i(z)$, entonces $x_j \geq z_j$. Como $x \in B_j(z)$, entonces $x_j < z_j$. Esto es una contradicción, así que los conjuntos son siempre disjuntos y la primera parte queda demostrada.

Supongamos que $x \in \cup_{i=0}^p B_i(z)$. Por definición, $x \in B$, así que $\cup_{i=0}^p B_i(z) \subset B$. Ahora sea $x \in B$. Si $x_i \geq z_i$, $\forall i$, entonces $x \in B_0(z)$. De lo contrario, sea j el mayor índice que verifica $x_j < z_j$. Entonces tenemos que $x \in B_j(z) \subset \cup_{i=0}^p B_i(z)$, así que $\cup_{i=0}^p B_i(z) = B$ y la segunda parte queda demostrada. □

Definición 4.3. Sea $B = [l, u]$ una caja y $z < B.u$. Decimos que aplicamos *p-partition* a la caja B respecto a z cuando consideramos el conjunto $\{B_i(z)\}_{i=1}^p$.

En la Figura 4.4 mostramos un ejemplo de *p-partition* aplicado a una caja B respecto de un punto z. Podemos observar que las cajas son disjuntas y que la caja no incluida en la partición es precisamente $B_0(z)$.



(a) Caja $\mathbf{B} = [(0, 0, 0), (20, 15, 10)]$ con punto no dominado $z = (5, 5, 5)$.

(b) Cajas resultantes tras aplicar *p-partition*: $\mathbf{B_1^*} = [(0, 5, 5), (5, 15, 10)]$, $\mathbf{B_2^*} = [(0, 0, 5), (20, 5, 10)]$, y $\mathbf{B_3^*} = [(0, 0, 0), (20, 15, 5)]$.

Figura 4.4: Partición de una caja usando *p-partition*.

Si cada vez que analizamos una caja y encontramos una solución aplicamos *p-partition* en lugar de *full p-split*, no existe ninguna contradicción en cuanto al uso de la alternancia de direcciones comentada en la contribución anterior, ya que en esta forma de partir las cajas, todas ellas siguen teniendo una dirección (salvo la caja inicial, a la que se le puede asignar una dirección cualquiera). Por otro lado, el uso de este método descarta automáticamente la caja $B_0(z)$ que es precisamente la porción de la caja inicial que contiene a los puntos que son dominados por la nueva solución. Sin embargo hay una desventaja en este proceso. Al analizar una caja, la solución encontrada puede quedar fuera de ella. Por ejemplo, basándonos en la Figura 4.4b, supongamos que analizamos la caja B_2 y que obtenemos como solución el punto $z' = (18, 3, 1)$. Claramente, $z' \notin B_2$. Este hecho no ocurría con *full p-split* ya que la cota inferior de todas las cajas era siempre el punto ideal. Entonces, ¿cuál es el sentido de aplicar esta forma de partir las cajas? En realidad la idea de usar una partición mediante cajas disjuntas no hace más que ajustar mejor el número de potenciales soluciones que podría haber en cada región a explorar. Al ir explorando siempre las de mayor volumen, se intenta de esta forma diversificar más las obtención de soluciones por el espacio objetivo. Si consideramos el volumen de las cajas tras la aplicación de *full p-split*, y dado que estas comparten regiones comunes, ya no se cumple que la suma de los volúmenes de todas las cajas a explorar coincide

UNIVERSIDAD DE MÁLAGA

con el volumen de la región total a explorar. Así, la diversificación por el espacio objetivo al seleccionar las cajas podría ser menor, precisamente por eso, porque las cajas comparten regiones comunes. Volviendo a la aplicación de *p-partition*, necesitamos aún aclarar qué hacer cuando la solución encontrada esté fuera de la caja explorada. Más aún, habrá que justificar si el método RE para detectar redundancias sigue siendo válido en este caso. Esto se abordará en la Sección 4.3.3.

III. Nueva función de prioridad

Como tercera contribución vamos a definir una nueva función de prioridad que mejore, de alguna forma, la selección de cajas a explorar. Para ello, daremos mayor prioridad a aquellas cajas que tengan una región con más potenciales puntos no dominados, que no tiene que coincidir exactamente con su volumen. Observemos en la Figura 4.1b que la caja $[(0,0,0),(5,5,5)]$ es una región común a B_1, B_2 y B_3. Además sabemos que en esa caja no habrá ninguna solución, ya que $z = (5,5,5)$ era un punto no dominado. Por otro lado, los distintos rangos de las funciones objetivo podrían dar lugar a cajas con mayor volumen pero ubicadas en una región muy amplia para una coordenada, pero pequeña para otras. Ya que uno de los propósitos del nuevo algoritmo es diversificar las soluciones por el espacio objetivo, intuitivamente surge la necesidad de normalizar estos rangos. Estas ideas dan lugar a la siguiente definición:

Definición 4.4. Dada una caja no vacía $B = [l,u]$, definimos la función *scaled* como

$$scaled(B) = \prod_{i=1}^{p} \left(\frac{u_i - l_i}{\overline{z}_i^N - z_i^I} \right). \tag{4.4}$$

Si $B = \emptyset$, definimos $scaled(B) = 0$.

Nótese que siempre ocurre que $scaled(B) \geq 0$, ya que podemos asumir, sin pérdida de generalidad, que la cota estrictamente superior del punto nadir, $\overline{z}^N$, siempre verificará $\overline{z}_i^N > z_i^I$, $\forall i$. Aún relajando el valor de $\overline{z}^N$ hacia una cota superior pero no estricta del punto nadir, el valor de *scaled(B)* se puede seguir considerando no negativo y bien definido, ya que, en caso de anularse algún denominador del producto, significaría que el correspondiente objetivo siempre tomaría el mismo valor. Este hecho podría detectarse al comienzo de la ejecución, y dicho objetivo podría suprimirse del problema.

Ahora veremos que en el caso de aplicar *p-partition* (véase nuevamente la Figura 4.4b), podemos detectar en qué caja se encuentra la región que domina al punto z.

Proposición 4.6. Sea $B = [l,u]$ una caja y $z \in B$. Si aplicamos *p-partition* se cumple que $[l,z] \subseteq B_p(z)$.

Demostración. De la definición de caja, $z < u$. Sea $x \in [l,z]$. Entonces $l_i \leq x_i < z_i < u_i$, $\forall i$. En particular, $x_p < z_p$. Esto significa que $x \in B_p(z)$, así que $[l,z] \subseteq B_p(z)$. Si $z = l$, tendríamos que $[l,z] = \emptyset$, y $B_i(z) = \emptyset$, $\forall i \neq 0$, por lo que también se cumple la tesis de la proposición. □

Si la caja $[l,z]$ no es vacía, contiene puntos que dominan a z. Como z es no dominado, la caja $[l,z]$ no puede contener ningún punto no dominado, así que el volumen de dicha caja no aporta información real sobre el número potencial de soluciones que podría contener la caja B_p.

Definición 4.5. Sea $B = [l,u]$ una caja, y $z < B.u$ un punto no dominado. Sea $B_i(z)$, $i = 1,\ldots,p$, la caja de dirección i obtenida tras aplicar *p-partition*. Entonces para cualquier $B_i(z) \neq \emptyset$ definimos el valor *reducido escalado* (en inglés, *reduced scaled value*) como

$$reduced_scaled(B,i,z) = \begin{cases} scaled(B_i) & \text{si} \quad i \neq p \\ scaled(B_i) - scaled([l,z]) & \text{si} \quad i = p. \end{cases} \tag{4.5}$$

Proposición 4.7. La función *reduced_ scaled* siempre toma un valor no negativo.

UNIVERSIDAD DE MÁLAGA

Demostración. Para $i = 1, \ldots, p-1$, es obvio que *reduced_scaled* toma un valor no negativo (ver Ecuación (4.4)). Supongamos ahora $i = p$. Si $[l, z] = \emptyset$, entonces $scaled([l, z]) = 0$ y $reduced_scaled(B_p(z), p, z) = scaled(B_p(z)) \geq 0$. Si $[l, z] \neq \emptyset$, entonces $z \in B$ y la Proposición 4.6 garantiza que $[l, z] \subseteq B_p(z)$, por tanto, $scaled([l, z]) \leq scaled(B_p(z))$, y *reduced_scaled* es no negativo. □

Una vez expuestas las tres contribuciones, las aplicaremos en la propuesta algorítmica del próximo apartado. Es decir, consideraremos alternancia de direcciones en la selección de cajas, aplicaremos *p-partition* en algunas de ellas (en la siguiente sección se explica en cuales), y consideraremos la función *reduced_scaled* como valor de prioridad de cada caja.

4.3.3. Propuesta algorítmica

Nuestra propuesta algorítmica se denomina *TPA*, por sus siglas en inglés, *Tchebycheff-Partition-Alternation*, ya que utiliza una métrica aumentada y ponderada de Tchebycheff, particiona las cajas utilizando *p-partition* y utiliza alternancia de direcciones en el espacio objetivo.

En la sección anterior dejamos una cuestión pendiente de resolver: ¿Cómo adaptar el algoritmo RE para eliminar cajas redundantes cuando se aplica *p-partition*? Detectar cajas redundantes al aplicar *full p-split* es muy sencillo, tan solo hay que comprobar si la cota superior de una caja domina a la cota superior de otra. Sin embargo, en el caso de *p-partition* dicha comprobación no es suficiente, ya que las cotas inferiores de las cajas varían. Para solventar este inconveniente, definimos el concepto de *unión de cajas*.

Definición 4.6. Sean $A = [l^a, u^a]$ y $B = [l^b, u^b]$ dos cajas con $u^a \leq u^b$. Definimos la *unión de cajas* (en inglés *joined box*) como una nueva caja $C = [l^c, u^c]$, donde $l_i^c = \text{mín}\{l_i^a, l_i^b\}$ y $u_i^c = u_i^b$, $\forall i = 1, \ldots, p$.

Cuando procedemos a hacer una unión de dos cajas, no podemos garantizar que la nueva caja sea disjunta del resto de cajas del conjunto $\mathcal{U}$. Sin embargo, sí podemos confirmar que ningún punto no dominado ha quedado excluido de la región de búsqueda, como veremos en la siguiente proposición:

Proposición 4.8. Con las condiciones de la Definición 4.6, se cumple que $A \cup B \subset C$, lo que significa que analizar la caja C sustituye el análisis de las otras dos cajas.

Demostración. Sea $x \in A \cup B$. Si $x \in A$, entonces $l_i^c \leq l_i^a \leq x < u_i^a \leq u_i^b = u_i^c,\ \forall i$, así que $x \in C$. Si $x \in B$, entonces $l_i^c \leq l_i^b \leq x < u_i^b = u_i^c,\ \forall i$, por lo que $x \in C$. □

El valor de prioridad de la nueva caja C podría ser mayor que el valor de las cajas previas A y B. Esto significa que la caja C tendrá mayor preferencia para ser seleccionada en las siguientes iteraciones. A continuación presentamos un nuevo procedimiento de actualización de cajas, que incluye la aplicación de *p-partition* y un filtrado de cajas redundantes que sustituye al método RE. Su pseudocódigo puede verse en el Algoritmo 5.

El algoritmo de actualización de cajas se divide en dos partes. En la primera parte (Líneas 1–9) aplicamos *p-partition* a cualquier caja cuya cota superior esté dominada por la nueva solución encontrada. Obsérvese que al iterar sobre todas las cajas de todas las listas, resolvemos el problema de que la solución podría quedar fuera de la caja que se usó para encontrarla. Algunas de las cajas B_i podrían ser vacías. Las que no lo sean son insertadas en la correspondiente lista de cajas (Línea 6). La segunda parte del método (Líneas 10–17) filtra cajas redundantes. Para ello, une cada pareja de cajas en la que la cota superior de una domine a la cota superior de la otra, y luego elimina las cajas para insertar la unión de ellas. De esta forma, se reduce el número de cajas a explorar.

El pseudocódigo de *TPA* puede encontrarse en el Algoritmo 6.

El método *TPA* utiliza la misma estructura que la del algoritmo genérico (Algoritmo 1), añadiendo las tres nuevas contribuciones y las adaptaciones al programa matemático usado. Las tres primeras líneas inicializan las estructuras de datos. Las Líneas 4–6 generan la caja inicial y determinan el parámetro ϵ que será fijo durante todo el algoritmo. En la Línea 7 se define el programa matemático aumentado y ponderado de Tchebycheff. Para cada caja seleccionada (mediante alternancia en sus direcciones), se fija el correspondiente vector ω

UNIVERSIDAD DE MÁLAGA

Algoritmo 5 *Update*

Entrada: $(\mathfrak{L}, z)$
Salida: $(\mathfrak{L}, z)$
1: **para todo** $B \in \bigcup_{k=1}^{p} \mathfrak{L}_k$ con $z < B.u$ **hacer**
2: **para** $i = 1$ **hasta** p **hacer**
3: $B_i(z) = \{x \in B \mid x_i < z_i \text{ and } x_j \geq z_j, \forall j > i\}$
4: **si** $(B_i(z) \neq \emptyset)$ **entonces**
5: $B_i(z).priority \leftarrow reduced_scaled(B, i, z)$
6: $\mathfrak{L}_i.insert(B_i(z))$
7: **fin si**
8: **fin para**
9: **fin para**
10: **para** $i = 1$ **hasta** p **hacer**
11: **para todo** $A, B \in \mathfrak{L}_i$ con $A.u \leq B.u$ **hacer**
12: $C \leftarrow join(A, B)$
13: $\mathfrak{L}_i.remove(A)$
14: $\mathfrak{L}_i.remove(B)$
15: $\mathfrak{L}_i.insert(C)$
16: **fin para**
17: **fin para**

Algoritmo 6 *TPA*

Entrada: f y $\mathcal{X}$ // Funciones objetivo y restricciones del problema MOCO
Salida: XZ // Un conjunto de soluciones eficientes y su imagen
1: $\mathfrak{L} = (\mathfrak{L}_1, \ldots, \mathfrak{L}_p)$ y $\mathfrak{L}_i = \emptyset,\ \forall i = 1, \ldots, p$ // Listas de cajas según su dirección
2: $\text{XZ} = \emptyset$
3: $k = 0$
4: Calcular z^I y $\overline{z}^N$
5: Fijar $\epsilon = 1/(2p(r-1))$ donde $r = \max_{i=1}^{p} \{\overline{z}^N - z^I\}$
6: $\mathfrak{L}_1.insert\left(\left[z^I, \overline{z}^N\right]\right)$
7: $P(u, \omega) \equiv \min_{x \in \mathcal{X}} \left(\max_{i=1,\ldots,p} \left(\omega_i \left| f_i(x) - z_i^I \right|\right) + \epsilon \cdot \sum_{i=1}^{p} \omega_i \left| f_i(x) - z_i^I \right| \right)$
8: **mientras** $\exists\, i : |\mathfrak{L}_i| > 0$ **y** *intime* **hacer**
9: $(B, k) \leftarrow Select_next_box(\mathfrak{L}, k)$
10: Fijar $\omega = (\omega_1, \ldots, \omega_p)$ donde $\omega_i = 1/\max\{1, B.u_i - z_i^I\}$
11: $(x^*, obj) \leftarrow$ Resolver $P(B.u, \omega)$
12: **si** $obj < 1$ **y** $z^I < B.u$ **entonces**
13: $\text{XZ} \leftarrow \text{XZ} \cup \{(x^*, f(x^*))\}$
14: *Update* $(\mathfrak{L}, f(x^*))$
15: **si no**
16: $\mathfrak{L}_k.remove(B)$
17: **fin si**
18: **fin mientras**

(Línea 10) y se resuelve el programa matemático. Si se cumplen las condiciones dadas por la Proposición 4.3 (Línea 12), significa que se ha obtenido un nuevo punto no dominado, que podrá estar o no en la caja actual. La función *Update* aplica *p-partition* a todas las cajas cuya cota superior esté dominada por la nueva solución, y realiza una unión de cajas cuando proceda. Las cajas vacías (sin solución) se eliminan (Línea 16). El valor lógico *intime* determina si el algoritmo puede continuar su ejecución o debe detenerse por haberse alcanzado el tiempo máximo de ejecución.

Teorema 4.1. El algoritmo *TPA* es correcto y termina encontrando el frente de Pareto completo.

Demostración. Veamos que el espacio de búsqueda se reduce en cada iteración.

UNIVERSIDAD DE MÁLAGA

Si al explorar una caja no hay solución, ésta se elimina. La caja eliminada tiene al menos un punto que no está en ninguna otra caja de $\mathcal{U}$, de lo contrario, estaría contenida en otra caja y habría sido eliminada en la unión de cajas de la iteración anterior. Por tanto, al eliminarla, el espacio de búsqueda se reduce.

Si al explorar una caja $B = [l, u]$ se encuentra una solución z, se garantiza que es eficiente por la Proposición 4.2. Además, por la Proposición 4.3, sabemos que domina a la cota superior de la caja explorada, es decir, $z < u$. Esto garantiza que al actualizar las cajas entramos en el primer bucle del Algoritmo 5, así que al menos a la caja B se le aplicará *p-partition*, por lo que la región $B_0(z)$ se descarta. Dado que $B_0(z) \neq \emptyset$ (por la Proposición 4.4), el espacio de búsqueda se reduce. De hecho, a todas las cajas cuya cota superior sea dominada por z se les aplicará *p-partition*, y en particular, se le aplicará a aquella que contenga a z, por lo que z no será generado más en sucesivas iteraciones y una región contenida en $\left[z, \overline{z}^N\right]$ será descartada de la búsqueda. Por otro lado, en el segundo bucle del Algoritmo 5 se unen algunas cajas. La definición de unión de cajas establece que la cota superior de la nueva caja coincide con la mayor de las cotas superiores en las antiguas, por lo que no estamos aumentando el espacio de búsqueda.

Dado que en un modelo combinatorio el espacio objetivo es finito y en cada iteración se reduce el espacio de búsqueda, el algoritmo *TPA* finaliza si se dispone de suficiente tiempo de ejecución. Además, genera todo el frente de Pareto ya que en ningún momento se eliminan de la búsqueda zonas que pudiesen contener puntos no dominados (solo se eliminan regiones dominadas por la última solución encontrada). □

4.4. Resultados computacionales

En esta sección presentamos los resultados de un análisis experimental. En la primera parte describiremos las instancias que usamos. Estas instancias se agrupan en categorías y estas, a su vez, en clases de problemas. Se analizan tres clases de problemas MOCO. Se ha considerado el uso de estas instancias por la variedad de características que poseen (distinto número de variables y restricciones), además de ser de los tipos más usados en este campo de investigación, pudiendo así evaluar el rendimiento general de manera más objetiva. Indicaremos también las métricas usadas, los parámetros de CPLEX utilizados y los algoritmos empleados en la comparativa. En la segunda parte mostraremos los resultados computacionales desde diversos puntos de vista. Para cuatro tiempos de ejecución fijados, analizamos el número de veces que cada algoritmo es mejor que el resto, y esto lo hacemos también particularizando a cada clase de problemas y a cada métrica. A continuación, establecemos un *ranking* para los algoritmos, clasificando cuales de ellos son mejores en función de la métrica considerada y del tiempo de ejecución, que se calcula en intervalos regulares hasta los 900 segundos. De esta forma podemos estimar el comportamiento *anytime* de los algoritmos. Por último, mostramos gráficamente el comportamiento de las tres instancias más duras (una de cada clase), evaluando cada una de las métricas cada cinco segundos, hasta los 900, para cada algoritmo. Finalizamos la sección con la realización de los contrastes de hipótesis apropiados para contrastar si las diferencias observadas son estadísticamente significativas.

4.4.1. Instancias, métricas y parámetros de CPLEX

Las instancias se dividen en tres clases de problemas multiobjetivo combinatorios bien conocidos. La primera contiene problemas de asignación multiobjetivo, *MOAP* (ver Ejemplo 2.9). La segunda clase comprende problemas multiobjetivo enteros, MOILP (ver Ejemplo 2.12). La tercera clase se corresponde con problemas de la mochila multiobjetivo, *MOKP* (ver Ejemplo 2.10). Todos los detalles sobre la formulación de estas tres clases pueden encontrarse también en [102] y en la dirección `http://home.ku.edu.tr/~moolibrary/`. Los datos quedan resumidos en la Tabla 4.2.

Por simplificar la notación, omitiremos el prefijo 'MO' y denotaremos a las clases de instancias como *AP*, *ILP* y *KP*. Cada clase de problemas tiene distintas categorías, que varían en función del número de variables que tenga el problema. Dentro de cada categoría encontramos 10 instancias distintas. En total, 480 instancias. Para las instancias de la clase *ILP* el número de restricciones es siempre la mitad que el número de variables, de ahí que se denote con la expresión $\{XX/2\}$.

UNIVERSIDAD DE MÁLAGA

Tabla 4.2: Instancias usadas en los experimentos computacionales. Clase, dimensión, número de variables o agentes, número de instancias y nombre.

Clase	p	Var (XX)	Instancia (YY)	Total	Nombre
AP	3	$\{5, 10, \ldots, 50\}$	$\{1, 2, \ldots, 10\}$	100	*AP_p-3_n-XX_ins-YY*
ILP	3	$\{10, 20, \ldots, 100\}$	$\{1, 2, \ldots, 10\}$	100	*ILP_p-3_n-XX_m-{XX/2}_ins-YY*
	4	$\{10, 20, \ldots, 80\}$	$\{1, 2, \ldots, 10\}$	80	*ILP_p-4_n-XX_m-{XX/2}_ins-YY*
	5	$\{10, 20, 30, 40\}$	$\{1, 2, \ldots, 10\}$	40	*ILP_p-5_n-XX_m-{XX/2}_ins-YY*
KP	3	$\{10, 20, \ldots, 100\}$	$\{1, 2, \ldots, 10\}$	100	*KP_p-3_n-XX_ins-YY*
	4	$\{10, 20, 30, 40\}$	$\{1, 2, \ldots, 10\}$	40	*KP_p-4_n-XX_ins-YY*
	5	$\{10, 20\}$	$\{1, 2, \ldots, 10\}$	20	*KP_p-5_n-XX_ins-YY*

Los nombres que utilizaremos para los algoritmos usados en los experimentos computacionales son:

- *ceyhan*[1] , para el algoritmo SBA de Ceyhan et al. [18].
- *holzmann*, para el algoritmo de Holzmann y Smith [91].
- *TPA*[2], para el algoritmo propuesto en la Sección 4.3.

Los algoritmos han sido programados usando lenguaje C/C++. El algoritmo *ceyhan* ha sido adaptado del original programando los contadores de tiempo y configurando los archivos de salida. El algoritmo *holzmann* ha sido programado usando el método RE para el filtrado de soluciones. El algoritmo *TPA* está basado en el de *holzmann*, pero añadiendo las tres nuevas contribuciones que comentamos anteriormente.

Las dos máquinas utilizadas para computar los experimentos son Dell PowerEdge T630 con dos CPUs Intel Xeon (E5-2670 v3) a 2.3 GHz, un total de 48 núcleos para cada una, 160 GB y 192 GB de memoria, respectivamente, y sistema operativo Ubuntu 16.04 LTS. Para cada ejecución se usa un único núcleo y 2 GB de RAM.

Los tres algoritmos emplean CPLEX 12.6.2 como resolutor. A continuación fijamos algunos de sus parámetros (extraídos de la Tabla 3.2). El resto de parámetros tendrán su valor por defecto:

$$\text{CPX_PARAM_EPGAP} = \text{CPX_PARAM_EPAGAP} = \text{CPX_PARAM_EPINT} = 0,$$
$$\text{CPX_PARAM_PARALLELMODE} = \text{CPX_PARAM_THREADS} = 1.$$

El parámetro CPX_PARAM_EPINT indica la tolerancia de integralidad permitida para las variables enteras. Su valor por defecto es 10^{-5}. El parámetro CPX_PARAM_EPGAP establece una tolerancia relativa para la diferencia entre el mejor valor objetivo y el objetivo del mejor nodo de los restantes que hay en el árbol de búsqueda en el procedimiento de *Branch & Cut*. Su valor por defecto es 10^{-4}. Por otro lado, CPX_PARAM_EPAGAP fija la tolerancia absoluta para la diferencia entre el mejor valor objetivo y el objetivo del mejor nodo restante. Su valor por defecto es 10^{-6}. Como vemos, estableciendo estos tres parámetros a 0 indicamos que la precisión debe ser exacta, más alla de los errores de redondeo limitados por la precisión de la máquina. El parámetro CPX_PARAM_PARALLELMODE se fija en modo determinista (valor 1). Esto significa que múltiples ejecuciones con el mismo modelo y la misma configuración de parámetros de entrada, proporcionarán el mismo camino en el árbol y la misma solución. Finalmente, CPX_PARAM_THREADS fija a 1 el número de hebras en paralelo que usará CPLEX. Cabe destacar que aunque la ejecución de los algoritmos es determinista y el tiempo límite será fijado, el número de soluciones encontrada podría diferir en las distintas ejecuciones. Esto es debido a que CPLEX maneja otros parámetros internos, como la memoria restante disponible, que podrían influir en la exploración del árbol que determinará la siguiente solución. Es por ello que, para suavizar estas diferencias, se ha ejecutado cada instancia 30 veces y hemos reportado sus valores medios.

Los indicadores de calidad que se utilizarán para evaluar las soluciones dadas por los algoritmos, así como para medir si son buenos algoritmos *anytime*, son los siguientes:

[1]Código fuente en `https://github.com/gokhanceyhan/MOIP_Solvers` (*Commit* 6b3b7e7bb9 del 17/08/19).
[2]Código fuente en `https://github.com/MiguelAngelDominguezRios/boxMO`.

UNIVERSIDAD DE MÁLAGA

- Porcentaje de soluciones obtenidas, ONVGR, definido en la Ecuación (3.15).
- Hipervolumen y porcentaje de hipervolumen, HV y HVR, definidos en las Ecuaciones (3.10) y (3.16), respectivamente.
- *General spread*, Δ^*, definido en la Ecuación (3.12).
- Una variante del epsilon aditivo, ε_+, que se definirá en la Ecuación (4.6).

La variante que vamos a utilizar del método epsilon aditivo, que se definió inicialmente en la Ecuación (3.11), normaliza los valores al intervalo $[0, 1]$. Así,

$$\varepsilon_+(N) = \max_{x \in PF} \min_{y \in N} \max_{i=1,\ldots,p} \left(\frac{y_i - x_i}{r_i} \right), \tag{4.6}$$

donde r_i es el rango del objetivo i en el conjunto PF. Abusando de la notación, seguiremos usando la misma notación para esta métrica epsilon aditiva normalizada. Esta métrica también es utilizada en los experimentos del trabajo de Ceyhan et al. [18] y es nombrada con el término de *coverage gap*. De acuerdo con la definición, es claro que $\varepsilon_+(N) \leq 1$. Por otro lado, ya que utilizaremos algoritmos exactos en los experimentos, el Corolario 3.1 garantiza que $\varepsilon_+(N) \geq 0$.

Nótese que para calcular las métricas anteriores es necesario conocer el frente de Pareto (en su defecto, necesitaríamos un frente de referencia, y los indicadores serían menos precisos). Inicialmente se han ejecutado todas las instancias sin tiempo límite para obtener los frentes completos, usando el Algoritmo 3 y usando *TPA*. Los tiempos empleados para resolver cada instancia variaron entre menos de un segundo y cerca de dos semanas. La razón de ejecutar cada instancia usando dos algoritmos distintos se justifica con la intención de detectar algún potencial error numérico debido al resolutor. De hecho, tras la ejecución final, se detectaron dos instancias de la clase *AP* que ofrecieron un resultado distinto para cada algoritmo:

- Instancia *AP_p-3_n-35_ins-4*, que presentó puntos dominados en el espacio objetivo.
- Instancia *AP_p-3_n-45_ins-6*, que generó 18374 puntos no dominados en lugar de 18375. Uno de ellos no fue encontrado.

Repitiendo la ejecución de estas dos instancias, pero manteniendo los parámetros CPXPARAMEPGAP , CPXPARAMEPAGAP y CPXPARAMEPINT a sus valores por defecto, obtuvimos la correcta solución en ambos casos. No entramos en analizar los motivos, pues cosideraremos el resolutor CPLEX como una caja negra. No obstante, mantendremos la configuración de los parámetros expuesta anteriormente (fijados a 0) con el fin de obtener mejor precisión en los resultados computacionales.

Aprovechando que conocemos todos los frentes de Pareto (por tanto, conocemos sus puntos nadir), a la hora de calcular el hipervolumen tomaremos como punto de referencia el vector $\overline{z}^N = z^N + (1, \ldots, 1)$. De esta forma permitimos que todos los puntos extremos tengan contribución al cálculo del hipervolumen. While et al. [177] proporcionan un programa para calcular el hipervolumen que puede obtenerse en `https://github.com/lbradstreet/WFG-hypervolume`. Este será el que usemos en nuestros experimentos computacionales.

4.4.2. Análisis de los resultados computacionales

Evaluamos las métricas para cada uno de los algoritmos (*ceyhan*, *holzmann*, *TPA*) y para cada tiempo fijado en el conjunto $\{10, 60, 300, 900\}$, medido en segundos. Estos tiempos de corte permiten, por un lado, conocer el rendimiento inicial, y por otro, el comportamiento en un tiempo mayor (aunque aún no muy elevado) de 15 minutos, además de medirlo en dos tiempos intermedios. Se realizan 30 ejecuciones para cada caso y se toman valores medios.

Análisis inicial

En las Tablas 4.3 y 4.4 presentamos los resultados para cada algoritmo y métrica. Hay dos columnas en cada combinación. La primera columna muestra el número de instancias para las que el el algoritmo tiene el mejor valor medio. Este valor puede ser compartido con otros algoritmos. En la segunda columna aparece el número de instancias para las que el algoritmo

UNIVERSIDAD DE MÁLAGA

correspondiente es el único que obtiene el mejor valor medio. Los resultados se muestran para cada una de las clases de instancias (*AP*, *ILP* y *KP*) así como para el total, que aparece en las cuatro últimas filas de cada tabla.

Tabla 4.3: Número de veces que cada algoritmo toma el mejor valor medio, sin exclusividad (*best*) y con exclusividad (*excl*), para ONVGR y HV.

		ONVGR						HV					
		ceyhan		*holzmann*		*TPA*		*ceyhan*		*holzmann*		*TPA*	
		best	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*
AP	t= 10 s	11	0	14	1	99	86	11	0	27	14	86	73
	t= 60 s	20	0	44	21	79	56	20	0	38	15	85	62
	t=300 s	29	0	66	33	67	34	29	0	33	0	100	67
	t=900 s	33	0	72	27	73	28	33	0	45	0	100	55
ILP	t= 10 s	43	6	109	48	166	105	40	4	113	53	163	103
	t= 60 s	52	3	144	57	160	73	51	3	130	44	173	87
	t=300 s	62	2	142	31	187	76	60	0	126	15	205	94
	t=900 s	76	0	155	16	204	65	76	0	146	7	213	74
KP	t= 10 s	41	0	77	19	141	83	41	0	94	36	124	66
	t= 60 s	56	0	98	5	155	62	56	0	93	1	159	67
	t=300 s	71	0	128	0	160	32	71	0	128	1	158	32
	t=900 s	80	0	149	0	160	11	80	0	148	3	156	12
TOTAL	t= 10 s	95	6	200	68	406	274	92	4	234	103	373	242
	t= 60 s	128	3	286	83	394	191	127	3	261	60	417	216
	t=300 s	162	2	336	64	414	142	160	0	287	16	463	193
	t=900 s	189	0	376	43	437	104	189	0	339	10	469	141

Tabla 4.4: Número de veces que cada algoritmo toma el mejor valor medio, sin exclusividad (*best*) y con exclusividad (*excl*), para Δ^* y ε_+.

		Δ^*						ε_+					
		ceyhan		*holzmann*		*TPA*		*ceyhan*		*holzmann*		*TPA*	
		best	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*
AP	t= 10 s	75	64	30	18	18	6	11	0	19	6	94	81
	t= 60 s	59	39	41	21	40	20	20	0	25	2	98	75
	t=300 s	44	15	42	12	73	43	29	0	33	0	100	67
	t=900 s	55	22	33	0	78	45	33	0	45	0	100	55
ILP	t= 10 s	141	105	72	23	92	43	43	7	122	54	159	91
	t= 60 s	162	114	83	22	84	23	52	4	137	44	172	79
	t=300 s	169	109	86	10	101	25	60	0	143	31	189	77
	t=900 s	179	103	100	6	111	17	76	0	159	18	202	61
KP	t= 10 s	121	80	63	14	66	17	41	0	90	32	128	70
	t= 60 s	122	66	70	3	91	24	56	0	99	7	153	61
	t=300 s	139	69	86	2	89	5	71	0	127	0	159	33
	t=900 s	145	66	93	2	92	1	80	0	148	2	157	12
TOTAL	t= 10 s	337	249	165	55	176	66	95	7	231	92	381	242
	t= 60 s	343	219	194	46	215	67	128	4	261	53	423	215
	t=300 s	352	193	214	24	263	73	160	0	303	31	448	177
	t=900 s	379	191	226	8	281	63	189	0	352	20	459	128

Para ONVGR, los algoritmos que proporcionan un mayor número de soluciones son *holzmann* y *TPA*. De hecho, *TPA* obtiene mejores resultados en todas las clases de instancias. La métrica ONVGR no es siempre la mejor referencia, especialmente cuando buscamos dispersión en las soluciones obtenidas. Por ello hemos seleccionado también el hipervolumen en nuestro análisis. Observando los resultados, vemos que para *TPA* volvemos a obtener nuevamente los mejores valores en todas las clases de instancias, sea cual sea el intervalo de tiempo considerado. Por ejemplo, en la clase *ILP*, a los 10 segundos, *ceyhan* obtiene el mejor hipervolumen medio en 40 ocasiones, pero en solo cuatro de ellas es el único ganador. En las otras 36 instancias, su valor es compartido con el de algún otro algoritmo. Las grandes diferencias que podemos observar para HV parecen justificarse en el hecho de que, en general, *ceyhan* encuentra muchas menos soluciones que el resto de algoritmos, por tanto, su hipervolumen es más bajo. Sin embargo, este algoritmo es capaz de encontrar pocas soluciones muy bien distribuidas rápidamente, de ahí sus buenos resultados para la métrica Δ^*, como podemos ver en la Tabla 4.4. Para la última métrica, ε_+, volvemos a observar que los mejores resultados

UNIVERSIDAD DE MÁLAGA

se obtienen para *TPA* en todos los casos, si bien no sabemos aún si las diferencias respecto a *holzmann* pueden considerarse significativas.

Resumiendo, mientras *ceyhan* está diseñado para encontrar pocas soluciones (pero muy bien distribuidas) rápidamente, esta dispersión no es tan buena cuando el tiempo progresa. Esto es debido a que el algoritmo realiza más de una llamada al resolutor en cada iteración, y utiliza subconjuntos de vectores que dependen del número de soluciones encontrado hasta el momento. Como consecuencia, se produce un crecimiento exponencial en el número de llamadas al resolutor, así que cuanto mayor sea el tiempo de ejecución, menor rendimiento tendrá el algoritmo. Comparando *holzmann* con *TPA*, hemos observado que los resultados de nuestra propuesta algorítmica son mejores salvo para la métrica Δ^*. Al estar estos dos métodos basados en la misma estructura y el mismo modelo matemático, queda patente que el uso de las tres contribuciones (alternancia de direcciones, uso de *p-partition* y función de prioridad basada en *reduced_scaled*) ha producido diferencias notables en el rendimiento, al menos acorde a las métricas usadas.

Ranking de algoritmos

Proporcionamos otro tipo de experimento computacional que mide el rendimiento de los algoritmos. Calculamos el ranking de cada uno de ellos en cada instancia y en diferentes tiempos de parada, desde los 90 hasta los 900 segundos, en intervalos de 90 segundos. Hacemos esto para cada una de las métricas: ONVGR, HVR, Δ^* y ε_+. Por ejemplo, fijado un tiempo límite de 360 segundos, comparamos el valor ONVGR de las 30 ejecuciones de los algoritmos para la primera instancia y asignamos los valores del 1 al 3, siendo 1 el algoritmo con mejor valor promedio para esa métrica. En caso de empate, procedemos a tomar valores medios para el *ranking*, por ejemplo 1.5 si hay un empate entre los dos primeros algoritmos.

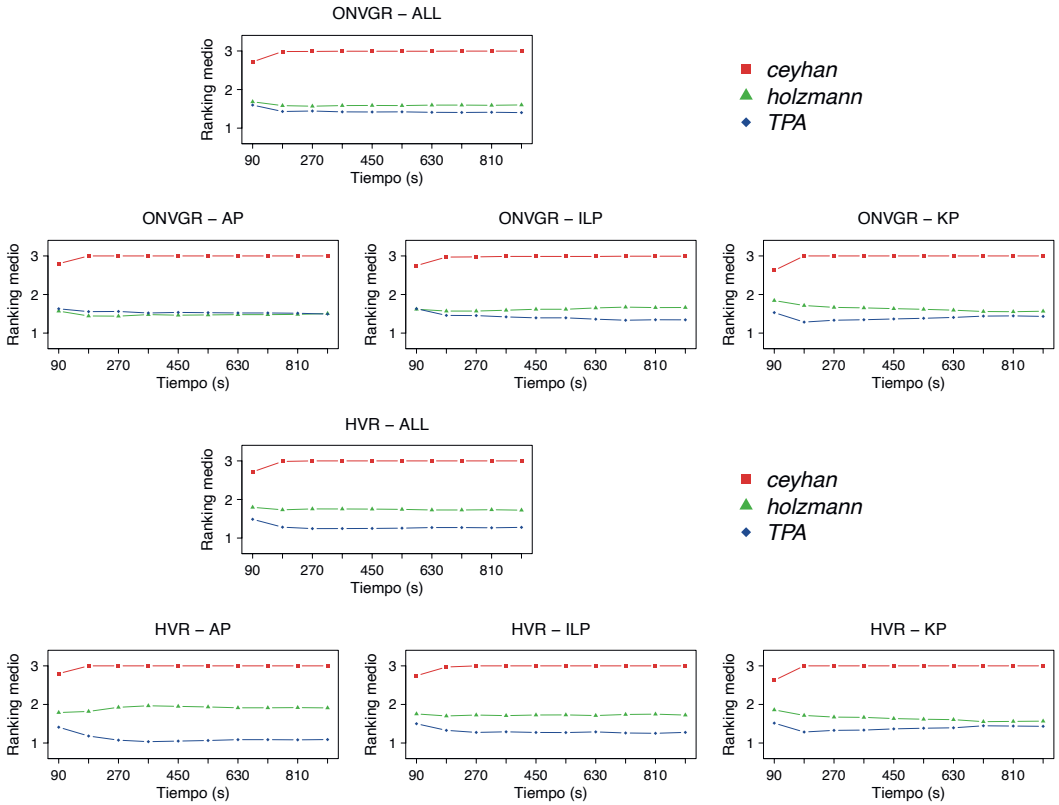


Figura 4.5: Valor medio del rango para los algoritmos en 10 puntos de corte, usando las métricas ONVGR y HVR. Cuanto menor es el valor del rango, mejor es el algoritmo.

Hay algunos casos en los que un algoritmo termina su ejecución en un tiempo concreto y otros no. Dada esa situación, se le asigna el valor 1 a ese algoritmo y el resto de valores del *ranking* se distribuye para los restantes en función de su resultado. Cuando todos los

UNIVERSIDAD DE MÁLAGA

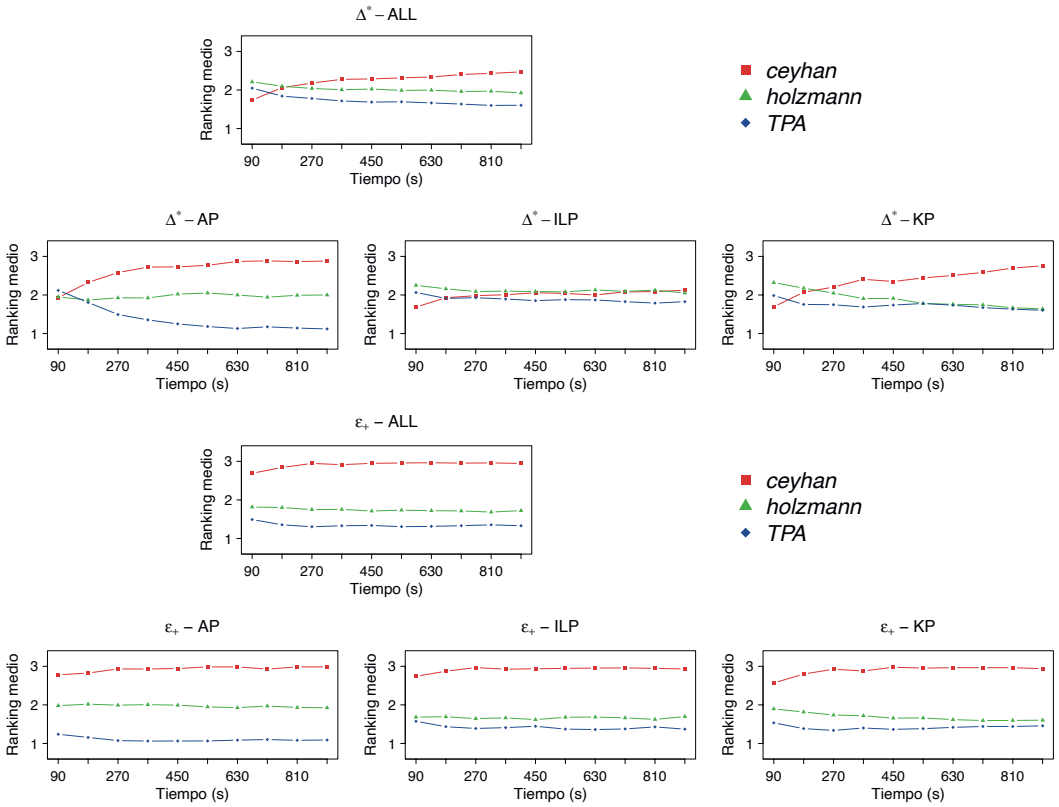


FIGURA 4.6: Valor medio del rango para los algoritmos en 10 puntos de corte, usando las métricas Δ^* y ε_+. Cuanto menor es el valor del rango, mejor es el algoritmo.

algoritmos hayan finalizado su ejecución en un tiempo concreto, eliminamos la correspondiente instancia del estudio. Los resultados se muestran en las Figuras 4.5 y 4.6. Podemos ver como en la mayoría de los casos, el mejor rango medio corresponde al algoritmo *TPA*, incluso con importantes diferencias respecto al resto en algunos casos.

Comportamiento *anytime* en instancias duras

El siguiente experimento computacional consiste en analizar el rendimiento *anytime* para algunas instancias concretas. Evaluamos cada una de las métricas cada 5 segundos de ejecución hasta los 900 segundos. Esto lo hacemos para las instancias más duras (aquellas que necesitan mayor tiempo de ejecución para obtener el frente de Pareto completo) en cada una de las clases consideradas.

La Figura 4.7 muestra los valores para ONVGR y HVR en las instancias *AP_p-3_n-50_ins-4*, *ILP_p-4_n-80_m-40_ins-9* y *KP_p-3_n-100_ins-4*. La Figura 4.8 usa las mismas instancias y muestra los resultados para las métricas Δ^* y ε_+. En ambos casos el eje x representa el tiempo de ejecución en segundos. El eje y representa el valor del correspondiente indicador de calidad. Para ONVGR y HVR vemos que el mejor comportamiento *anytime* podría atribuirse a *TPA* (con escaso margen respecto a *holzmann* cuando consideramos HVR), no solo por ser el que mejores valores obtiene, sino porque los propios valores obtenidos pueden considerarse buenos. Obsérvese que en instancias que tardan varios días en ejecutarse podemos obtener un porcentaje total del hipervolumen cercano al 100 % en solo 15 minutos de ejecución. Esto indica una buena dispersión de las soluciones, confirmado además por el valor de ε_+ para *TPA*, que se encuentra relativamente cerca de 0. El hipervolumen muestra un alto crecimiento inicial y una estabilización de su valor al final, como su propia definición sugiere. En la métrica Δ^* encontramos el único caso donde no gana *TPA*. De hecho es en muchos momentos el peor algoritmo de los tres para los problemas *AP* e *ILP*, aunque posee

UNIVERSIDAD DE MÁLAGA

mejores resultados para *KP*. No obstante, recordemos que este indicador no es Pareto *compliant* (ver Definición 3.14), de ahí también que se observe que las gráficas no son monótonas en este caso.

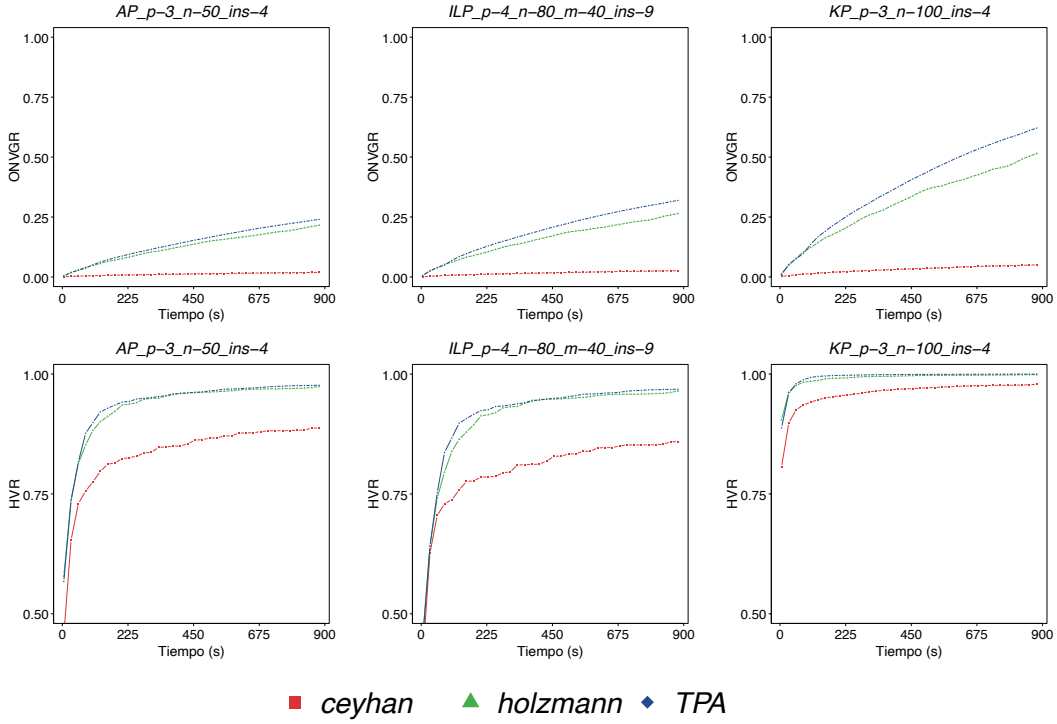


FIGURA 4.7: ONVGR y HVR en función del tiempo para tres instancias seleccionadas. A mayor valor, mejor es el algoritmo.

4.4.3. Validación estadística

Finalizado el análisis computacional, resta validar estadísticamente si las diferencias observadas entre *TPA* y el resto de algoritmos son significativas (véase el apartado 3.4.4). Aplicamos el test no paramétrico de Friedman para comparar si existen diferencias entre los tres algoritmos. Este test se aplica para cada métrica por separado, y considerando puntos de corte desde 90 hasta 900 segundos en saltos de 90 segundos. Cada tiempo de corte es un test distinto. En los casos en los que el p-valor obtenido es menor que 0.01, podemos considerar que hay una fuerte evidencia de que el rendimiento de los tres algoritmos es distinto en cuanto a esa métrica. En caso de ser significativo el contraste, realizamos un test *post hoc*, concretamente el test de comparación múltiple de Nemenyi. Este test compara cada pareja de algoritmos en busca de diferencias significativas usando datos pareados sobre las 480 instancias, tanto para cada clase de instancias como para el conjunto de todas ellas. Seleccionamos aquellos casos en los que un algoritmo es significativamente mejor que los otros dos, y entonces aplicamos el test de Wilcoxon de los rangos signados para confirmar que, efectivamente, dicho algoritmo es mejor. Todos los experimentos se han realizado usando el paquete de R llamado *PMCMRplus*.

Resumimos la realización de todos estos contrastes de hipótesis en la Tabla 4.5. Para una métrica y un tiempo de ejecución fijados, usamos la inicial del algoritmo (C, H o T) para indicar si dicho algoritmo es significativamente mejor que los otros dos para un nivel de significación $\alpha = 0.01$. Si además, el p-valor queda por debajo de 0.001, marcamos la inicial en negrita.

Nótese que, aunque *TPA* mostraba el mejor número de soluciones en los experimentos anteriores, estas diferencias no pueden considerarse significativas si lo comparamos respecto a *holzmann* en el caso *AP*, de ahí que ninguno de los tests haya sido significativo. Sin embargo, para el resto de clases, hay algunos tests significativos, y en el total, sí se puede considerar que *TPA* es estadísticamente mejor a partir de los 630 segundo de ejecución. Para las métricas

UNIVERSIDAD DE MÁLAGA

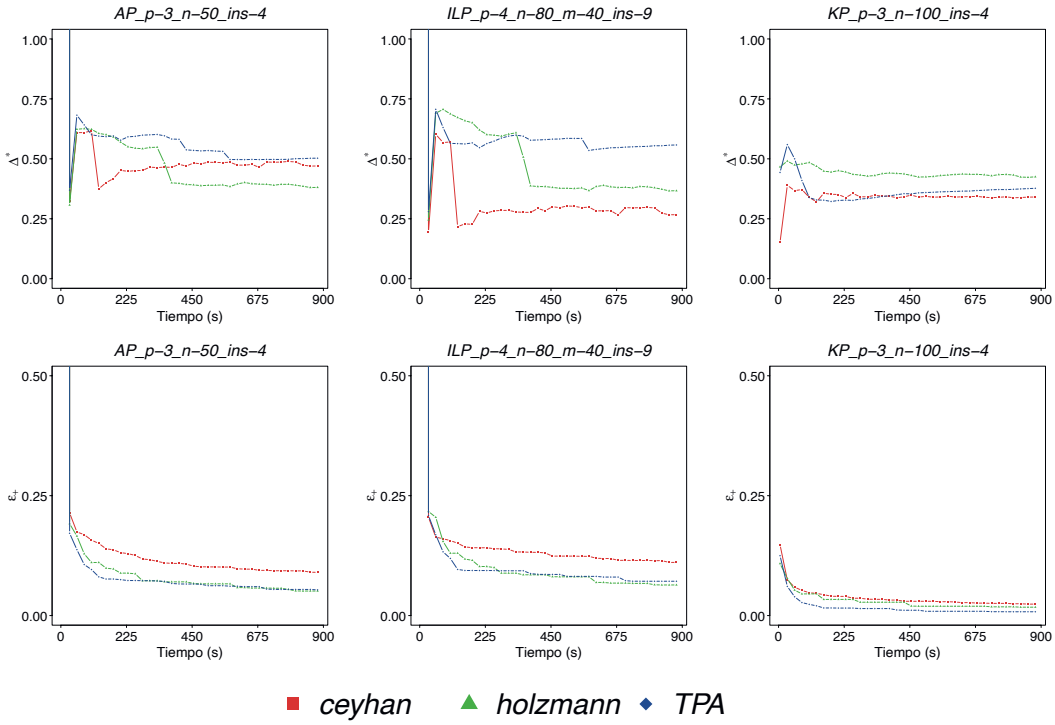


FIGURA 4.8: Δ^* y ε_+ en función del tiempo para tres instancias seleccionadas. A menor valor, mejor es el algoritmo.

TABLA 4.5: Tests estadísticos para cada clase y métrica considerando diferentes tiempos de ejecución. Se muestra la inicial del algoritmo en los casos en que es significativamente mejor que los otros dos con nivel de significación $\alpha = 0.01$. Si además las diferencias son significativas a nivel $\alpha = 0.001$, dicha inicial se marca en negrita.

Test de hipótesis		90s	180s	270s	360s	450s	540s	630s	720s	810s	900s
AP	ONVGR										
	HVR		**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	Δ^*				T	**T**	**T**	**T**	**T**	**T**	**T**
	ε_+	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
ILP	ONVGR							T	**T**	T	**T**
	HVR		**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	Δ^*	**C**	**C**	**C**	**C**	**C**	**C**	**C**	**C**	C	C
	ε_+						T	**T**	T		**T**
KP	ONVGR		**T**	T	T	T	T				
	HVR	T	**T**	**T**	**T**	T	T				
	Δ^*		**C**	C		**C**	**C**	**C**	C		
	ε_+	T	**T**	**T**	**T**	T	T				
TOTAL	ONVGR							T	T	T	T
	HVR	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	Δ^*	**C**									
	ε_+	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**

HVR y ε_+ también en la mayoría de los casos *TPA* es significativamente mejor para cada clase, y lo es siempre si consideramos la unión de todas las instancias. Incluso para Δ^* se observan diferencias significativas a partir de los 360 segundos en la clase *AP*, si bien en el resto de clases es *ceyhan* el ganador mayoritario, aunque si consideramos todas las instancias, solo es significativa la mejora para el caso de 90 segundos de ejecución.

UNIVERSIDAD DE MÁLAGA

4.5. Conclusiones y trabajo futuro

En este capítulo hemos desarrollado con detalle un algoritmo que resuelve de manera exacta problemas combinatorios multiobjetivo. Con suficiente tiempo de ejecución, el algoritmo es capaz de identificar el frente de Pareto completo y un punto eficiente asociado a cada punto no dominado. Más aún, si el algoritmo corta su ejecución en un instante cualquiera, los resultados parciales obtenidos muestran una buena dispersión por el espacio objetivo, acorde a las métricas consideradas: ONVG, HV, Δ^* y ε_+. Ha sido comparado con otros dos algoritmos existentes en la literatura: SBA, del trabajo de Ceyhan et al. [18], y el algoritmo propuesto por Holzmann y Smith [91]. Estos métodos se pueden considerar también buenos algoritmos *anytime* para este tipo de problemas.

Inicialmente se ha realizado un amplio estudio de la literatura en este campo, identificando algunos de los problemas más relevantes y concluyendo cuales podrían ser los más adecuados para utilizar en nuestra comparativa. El algoritmo propuesto está basado en un método genérico propuesto por Dächert y Klamroth [36] y adaptado para ser diseñado como un algoritmo *anytime*. Las contribuciones han sido las siguientes:

- El establecimiento de una nueva estrategia para seleccionar la siguiente caja a explorar en el espacio objetivo, alternando direcciones.
- Una nueva forma de partir el espacio de búsqueda, descartando la región dominada por la última solución encontrada.
- La definición de una función de prioridad asociada a cada espacio de búsqueda que permite medir más objetivamente el número de potenciales puntos no dominados en su interior.

Basándonos en la Figura 4.2 mostramos ahora en la Figura 4.9 un resumen de las características principales usadas en nuestro algoritmo.

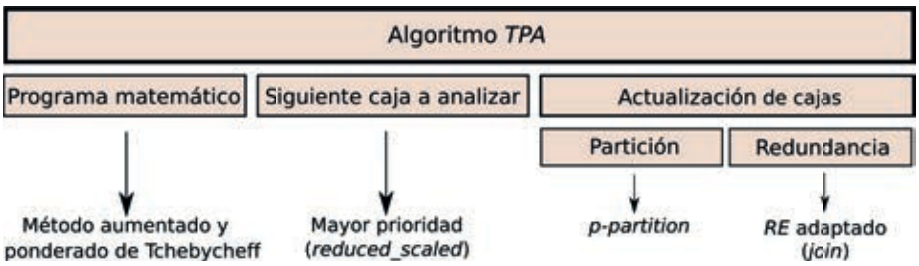


Figura 4.9: Definición de características asociadas al algoritmo *TPA*.

Se ha comparado *TPA* con *ceyhan* y *holzmann* de varias formas. El análisis de los resultados computacionales utiliza 480 instancias existentes en la literatura, ejecutándose 30 veces cada una de ellas y tomando sus valores medios. Los tiempos considerados varían según la tabla o figura considerada, siempre en un tiempo máximo de 900 segundos. Finalmente se han contrastado estadísticamente las diferencias entre los tres métodos, mostrándose que *TPA* resulta tener mejor rendimiento computacional en la mayoría de los casos. En ocasiones los resultados no han sido concluyentes y en algunos casos concretos, *ceyhan* ha resultado tener el mejor rendimiento en cuanto a la métrica Δ^*, que no es Pareto *compliant*.

Como trabajo futuro se considera la extensión de estos experimentos computacionales a otros bancos de instancias, para ver en qué tipo de problemas *TPA* se pudiera comportar mejor. Otra propuesta es la de aplicar este método a problemas multiobjetivo en el campo de la industria, donde el decisor interactúa y se hace necesario tomar decisiones a corto plazo. Algunas de las ideas introducidas en este capítulo pueden ser aplicadas a métodos heurísticos. Un último campo pendiente de explotar es el de usar métodos híbridos que combinen este algoritmo exacto con heurísticas para aprovechar las ventajas de ambos campos. Sobre esto último hablaremos en el Capítulo 5.

UNIVERSIDAD DE MÁLAGA

Capítulo 5

Algoritmos híbridos en optimización combinatoria multiobjetivo

Cuando usamos metaheurísticas para obtener frentes de Pareto aproximados en problemas multiobjetivo, estos pueden obtenerse, en general, en un corto tiempo de ejecución. Uno de los problemas con los que se encuentran las metaheurísticas es el de encontrar una solución factible, especialmente en el caso en que el modelo posea restricciones. Por otro lado, los resolutores ILP son capaces de encontrar soluciones factibles con mayor facilidad que las metaheurísticas, pero podrían requerir un tiempo de cómputo muy elevado si lo que deseamos es obtener el óptimo. De hecho, también podría ocurrir que el resolutor encuentre el óptimo relativamente rápido, pero emplee mucho tiempo en comprobar que dicha solución es, efectivamente, óptima. Estas observaciones nos han permitido desarrollar las dos propuestas algorítmicas híbridas que presentaremos en este capítulo. En primer lugar mostraremos un método capaz de aproximar el frente de Pareto en cualquier problema multiobjetivo binario con restricciones lineales. El segundo algoritmo aprovecha los parámetros internos del resolutor para convertir *TPA* en un algoritmo híbrido aproximado. Este último algoritmo es válido para cualquier problema combinatorio multiobjetivo. Cerramos el capítulo con una sección que describe las conclusiones y el trabajo futuro.

5.1. Algoritmo *MOFeLS*

En esta primera sección presentamos un algoritmo híbrido que mezcla una búsqueda local con programación lineal entera para resolver problemas multiobjetivo binarios con restricciones lineales. El algoritmo proporciona un frente de Pareto aproximado y bien distribuido por el espacio objetivo en cualquier momento, es decir, puede considerarse *anytime*. Tras una pequeña introducción inicial y el desarrollo de la propuesta algorítmica, evaluaremos los resultados computacionales comparándolo con tres metaheurísticas y usando un total de 28 instancias provenientes de dos bancos de instancias usadas en la literatura. Veremos que los resultados muestran un mejor rendimiento para este algoritmo híbrido, que llamamos *MOFeLS*, cuando consideramos el número de soluciones obtenidas, hipervolumen generado, distancia generacional, distancia generacional inversa e indicador epsilon aditivo.

5.1.1. Introducción

Las metaheurísticas, y en particular los algoritmos evolutivos, han sido ampliamente estudiados para resolver problemas de optimización multiobjetivo usando solo la información de la función de *fitness* y de las restricciones (véase Liu et al. [118]). En contextos reales disponemos de más información que la obtenida únicamente por cada solución. La estructura matemática de la función objetivo así como de las restricciones se pueden utilizar para mejorar la eficiencia de los algoritmos de optimización. Por ejemplo, la expresión de la función objetivo puede usarse para implementar eficientemente una búsqueda local usando Δ-evaluación (Chicano et al. [24]) y la información contenida en las restricciones puede aprovecharse para guiar al algoritmo y comprobar la factibilidad de una solución de manera eficiente sin necesidad de reevaluarla completamente (Chicano et al. [25]).

UNIVERSIDAD DE MÁLAGA

Conforme añadimos restricciones a un problema, encontrar soluciones factibles podría resultar cada vez más difícil para un algoritmo evolutivo. La estrategia de búsqueda de estas soluciones factibles suele estar basada en algún tipo de penalización, bien en la función objetivo, o bien mediante el reemplazo o la reparación de la solución en la población [135]. Por otro lado, los resolutores ILP permiten obtener fácilmente una solución factible, mediante el uso de técnicas de programación lineal. Esta será la idea principal de la presente sección. Propondremos una nueva *matheurística* para resolver problemas multiobjetivo con variables binarias y con cualquier número de restricciones lineales (MOBO, *MultiObjective Binary Optimization*). Restringiremos el uso del algoritmo únicamente a variables binarias, dado que utilizaremos una búsqueda local donde la mutación se basará en el cambio de bit (*bit flip*). No obstante, cualquier problema de optimización combinatorio puede reducirse a un problema binario mediante los cambios de variable apropiados.

En una *matheurística*, generalmente el resolutor optimiza algún subproblema menor que el original y dicha solución se obtiene en un tiempo menor. Un ejemplo clásico lo tenemos en el algoritmo CMSA (*Construct, Merge, Solve and Adapt*), desarrollado por Blum et al. [12, 11]. En este caso, nuestra propuesta es diferente. Usaremos el resolutor únicamente para obtener una solución factible (exploración), y la búsqueda local se encargará de la optimización. Tomaremos como referencia base el algoritmo que resuelve problemas MOCO (Algoritmo 3), que permite explorar distintas regiones de búsqueda, y que pueden ser consideradas como cajas. Sin embargo, en este caso, las soluciones encontradas no tienen por qué ser eficientes, por lo que el algoritmo será aproximado, no exacto. Las principales contribuciones del trabajo que presentamos en esta sección son:

- Uso del resolutor ILP para encontrar rápidamente soluciones factibles en el espacio de búsqueda, en problemas binarios multiobjetivo con restricciones lineales.
- Combinación del resolutor ILP con una técnica de exploración usando búsqueda local basada en Δ-evaluación.
- Comparación del algoritmo *MOFeLS* (*MultiObjective search based on integer linear programming for Feasibility and Local Search*) con tres conocidos algoritmos evolutivos: NSGA-II [42], SPEA2 [192] y MOEA/D [186]. Usaremos 28 instancias procedentes de dos bancos diferentes.

El método *MOFeLS* es capaz de aproximar el frente de Pareto para cualquier problema binario multiobjetivo, incluso con restricciones lineales de igualdad, que son un *handicap* para las metaheurísticas basadas en mutación con cambio de bit. De hecho, al basarnos en el Algoritmo 3, pretendemos conseguir buena dispersión de las soluciones si vamos seleccionando cajas a explorar con el mayor volumen posible.

5.1.2. Propuesta algorítmica

Definimos un problema binario multiobjetivo de dimensión p, en su forma matricial, como

$$\min_{x \in \mathcal{X}} Cx, \tag{5.1}$$

donde $C \in \mathbb{R}^{p \times n}$ es la matriz cuya fila i representa el vector de coeficientes de la función objetivo $f_i(x)$. El vector $x \in \mathbb{B}^n$ representa un vector binario de decisión, y $\mathcal{X} = \{x \in \mathbb{B}^n : Ax *_{op} b\}$ es el conjunto factible, que se supone no vacío. Aquí, A es una matriz de orden $m \times n$ donde cada fila contiene el vector de coeficientes de la restricción m-ésima y $b \in \mathbb{R}^m$ es el vector de términos independientes. Todos los elementos de C, A y b son números racionales. El vector de operadores $*_{op}$ tiene longitud m, y su elemento i contiene el sentido de la i-ésima restricción: '$\leq$', '$\geq$' o '$=$'. El objetivo también puede representarse en la forma $Cx = f(x) = (f_1(x), \ldots, f_p(x))$, en concordancia con la definición dada por las Ecuaciones (2.16)–(2.17). Si tuviésemos que maximizar algún objetivo, utilizaríamos la propiedad $\max(f_i(x)) = -\min(-f_i(x))$, para unificar todos los objetivos al tipo de minimización. En el caso en que las funciones objetivo o las restricciones fuesen polinómicas no lineales, se podría linealizar fácilmente el modelo añadiendo nuevas restricciones para los productos. Por ejemplo, el producto de las variables binarias $x_1 x_2$ se puede reemplazar por y, añadiendo como nuevas restricciones $2y \leq x_1 + x_2 \leq y + 1$.

UNIVERSIDAD DE MÁLAGA

Basándonos en el Algoritmo 3, diseñaremos un método en el que seleccionaremos siempre como siguiente caja a explorar aquella que tenga más volumen, partiremos dicha caja usando *full p-split* cada vez que encontremos una solución (lo cual implica que consideraremos siempre la misma cota inferior para todas las cajas), y usaremos el método RE para detectar redundancias. Existen otras opciones a considerar, como, por ejemplo, usar la partición de cajas mediante *p-partition*, que vimos en el Capítulo 4. No obstante, considerando esta opción, se ha observado durante una experimentación preliminar que el rendimiento era inferior. El uso de *p-partition* generaba un número mucho mayor de cajas a explorar. Dado que las soluciones eran aproximadas, estas podían encontrarse en cualquier lugar del espacio objetivo, y el método de filtrado de soluciones apenas entraba en acción. Descartada esa opción y considerando como base el referido Algoritmo 3, mostramos el pseudocódigo del método *MOFeLS* en el Algoritmo 7.

Algoritmo 7 *MOFeLS*

Entrada: *TILIM* // Tiempo límite para una llamada al resolutor ILP
Salida: N // Frente de Pareto aproximado
1: $N = \emptyset$
2: $\mathcal{U} = \emptyset$ // Lista de cajas
3: $\delta = (\delta_1, \ldots, \delta_1)$ // $0 < \delta_1 < 1$
4: Estimar cotas del frente de Pareto usando relajación lineal: L, U
5: $\mathcal{U} \leftarrow \{[L, U]\}$
6: **mientras** ($\mathcal{U} \neq \emptyset$) **y** (**no** condición de parada) **hacer**
7: $B \leftarrow \text{argmax}_{B \in \mathcal{U}}\,(v(B))$
8: **si** (Existe solución factible para $P(B.u - \delta)$ en tiempo *TILIM*) **entonces**
9: $x \leftarrow$ Solución de $P(B.u - \delta)$
10: *Slack* $\leftarrow$ Obtener vector *slack* a partir del resolutor ILP
11: $x \leftarrow Hill_Climbing(x, Slack)$
12: $N \leftarrow Filter(N \cup \{(x, f(x))\})$
13: $Actualizar(\mathcal{U}, f(x))$
14: **si no**
15: $\mathcal{U} \leftarrow \mathcal{U} - \{B\}$
16: **fin si**
17: **fin mientras**

El algoritmo mantiene una estructura N (inicialmente vacía) que contendrá una aproximación del frente de Pareto, junto con su correspondiente vector de decisiones asociado, que servirá como aproximación del conjunto eficiente, $\mathcal{X}_E$. La estructura $\mathcal{U}$ es una lista de cajas o regiones a explorar, también inicialmente vacía. La caja inicial se obtiene calculando una cota inferior y superior del frente de Pareto. Esto se consigue resolviendo los p modelos relajados $L_i = \text{mín}\{f_i(x)\}$ y los p modelos $U_i = \text{máx}\{f_i(x)\}$, sujeto a todas las restricciones, pero considerando las variables binarias como continuas. El vector $L = (L_1, \ldots, L_p)$ servirá de cota inferior para todas las cajas de $\mathcal{U}$.

El parámetro de entrada *TILIM* representa el tiempo máximo total (en segundos) empleado por el resolutor en cada llamada. Este tiempo será fijo durante toda la ejecución. El resolutor ILP solamente se utiliza para obtener una solución factible en la caja a explorar. En principio el valor de *TILIM* no debería ser muy elevado, ya que solo pedimos al resolutor que encuentre una solución factible, no necesariamente óptima. Obviamente, esto dependerá del problema considerado, de ahí que consideremos este valor como un parámetro de entrada. Si al explorar una caja, el resolutor no ha sido capaz de encontrar una solución factible en un tiempo *TILIM*, entonces es posible que la caja no contenga puntos no dominados, por lo que la descartamos y pasamos a explorar la siguiente caja. Este parámetro, bien configurado, podría permitir un mejor rendimiento del algoritmo, sin perder tiempo en explorar zonas del espacio objetivo donde potencialmente no existan puntos no dominados. Una desventaja podría ser que podríamos omitir soluciones. Hay que tener en cuenta que *TILIM* no debe tomar un valor muy pequeño, ya que podría incluso no generar ni siquiera la caja inicial, al no encontrar la solución del problema relajado. En general, con un valor entre 0.1 segundos y 1 segundo, funciona bien el algoritmo, aunque esto dependerá, evidentemente, de la instancia considerada.

UNIVERSIDAD DE MÁLAGA

El programa matemático usado en cada iteración es el modelo híbrido dado por la Ecuación (3.3). Dado que el valor asignado a los pesos de cada función objetivo no influye sobremanera en la obtención de una solución factible del modelo, tomaremos $\lambda_k = 1, \forall k$. Denotamos por $P(u)$ al programa matemático que explora la caja con cota superior u. El vector δ de la Línea 3 es necesario para garantizar que la potencial nueva solución no sea igual a la cota superior de la caja explorada, por eso en la Línea 8 resolvemos el modelo para el vector $B.u - \delta$, garantizando así que la aproximación obtenida se encuentra dentro de la caja explorada.

El bucle de las Líneas 6–17 analiza las distintas cajas y va obteniendo soluciones aproximadas mientras no se cumpla una condición de parada (tiempo máximo de ejecución, o $\mathcal{U} = \emptyset$). La caja seleccionada será aquella que tenga mayor volumen (Línea 7). Una vez obtenida una solución, se pasa al proceso de análisis y actualización del conjunto $\mathcal{U}$. Este proceso comienza con la parte heurística del algoritmo, que consiste en encontrar un óptimo local partiendo de la solución factible dada por el resolutor. La llamada al resolutor en la Línea 8 tiene las siguientes posibles salidas[1]:

a) Se encuentra una solución óptima.

b) Se encuentra una solución factible subóptima.

c) Tiempo *TILIM* excedido, pero se ha encontrado una solución factible.

d) No hay solución (infactible).

e) Tiempo *TILIM* excedido y sin solución factible encontrada.

f) Problema infactible o no acotado.

En los casos en los que la salida sea d), e) o f), eliminaremos la caja (Línea 15), y pasaremos a la siguiente iteración. En otro caso, hemos encontrado una solución factible, y *MOFeLS* llama a la función *Hill_ Climbing* para realizar la búsqueda local. Cada óptimo local es insertado en N y las soluciones repetidas o dominadas se descartan durante la ejecución (procedimiento *Filter* de la Línea 12). El procedimiento *Actualizar* de la Línea 13 actúa exactamente igual que el método RE (Algoritmo 2). El vector *slack* de la Línea 10 se obtiene haciendo $Slack = b - Ax$. Cada coordendada de este vector es accesible en tiempo constante gracias al resolutor. La coordenada i representa el salto o *gap* que existe para la restricción i-ésima, es decir, la diferencia entre el valor del término independiente, y el de la restricción i-ésima evaluada mediante el vector x. Si este *gap* vale 0 significa que la restricción está saturada.

Algoritmo 8 *Hill_ Climbing*

Entrada: x, $Slack$
Salida: x
1: $M = (false, false, \ldots, false)$ //Asignar todas las componentes de x como no visitadas
2: **mientras** $(\exists i \mid M[i] = false\ )$ **hacer**
3: $j \leftarrow$ Seleccionar aleatoriamente un índice de una componente no visitada en x
4: $M[j] = true$ // Visitar x_j
5: $(\delta_c, \Delta) \leftarrow Bit_Flip(x, j, Slack)$
6: **si** $(\delta_c < 0)$ **entonces**
7: $x_j = 1 - x_j$
8: $Slack \leftarrow Slack + \Delta$ // Actualizar vector *Slack*
9: $M[i] = false,\ \forall i$ // Marcar todas las componentes en x como no visitadas
10: **fin si**
11: **fin mientras**

A continuación explicamos en el Algoritmo 8 la estrategia de búsqueda local. El valor de *fitness* de una solución factible x viene dado por $\sum_{i=1}^{p} f_i(x)$. Consideramos el vecindario de una solución x como el conjunto de soluciones que están a distancia Hamming igual a 1 del vector x (solo hay un *bit* diferente entre los vectores).

La función *Hill_ Climbing* tiene dos parámetros de entrada, la solución factible x, y su correspondiente vector *Slack* asociado. La salida es otra solución x, también factible, que

[1] Salidas posibles usando CPLEX 12.6.2.

UNIVERSIDAD DE MÁLAGA

además es óptimo local (podría ser el mismo x de entrada). Inicialmente se establecen todas las componentes del vector solución x como no visitadas, a través del vector M que se inicializa a *false* para cada elemento. Mientras exista alguna componente sin visitar, se selecciona una de ellas aleatoriamente, y se comprueba si una mutación del *bit* correspondiente producirá una mejora en su valor de *fitness*. Esto se comprueba mediante una llamada a la función *Bit_Flip* de la Línea 5, que devuelve el incremento en la función objetivo, δ_c. Si este incremento es negativo significa que habrá habido mejora, por lo que cambiamos dicho *bit*, actualizamos el vector *Slack* convenientemente y volvemos a empezar el proceso desde el principio en busca de una nueva variable que, invertida, siga siendo factible y mejore el valor objetivo. Nótese que los valores en el vector *Slack* para las restricciones de tipo '$\leq$' serán siempre no negativos. En el caso de restricción tipo '$\geq$' siempre será no positivo, y para restricciones de igualdad, dicho valor siempre será igual a 0. Veamos esto con un poco más de detalle en el siguiente ejemplo ilustrativo.

Ejemplo 5.1. Consideremos el siguiente problema biobjetivo binario con restricciones lineales:

$$\begin{aligned} \text{mín} \quad & f(x) = (x_2 + x_3 \ , \ x_3 + x_4 + 2x_5)\,, \\ s.a. \quad & x_1 + 2x_2 - 2x_3 + 7x_4 + x_5 \leq 8, \\ & x_1 + x_2 + x_3 - x_4 + 4x_5 \geq 2, \\ & x_1 + x_5 = 2, \\ & x \in \mathbb{B}^5. \end{aligned}$$

Supongamos que el resolutor nos devuelve la solución factible $x = (1, 1, 0, 0, 1)$. Entonces, $Slack = (4, -4, 0)$, y el valor objetivo de esta solución será $f_1(x) + f_2(x) = 1 + 2 = 3$.

Algoritmo 9 *Bit_Flip*

Entrada: x, j, *Slack*
Salida: δ_c, Δ
1: $\Delta = (0, \ldots, 0)$ // Variación del vector *Slack*
2: $\delta_c \leftarrow$ Obtener coeficiente de la función objetivo de x_j a partir del resolutor
3: **si** $(\delta_c \cdot (-1)^{x_j}) \geq 0)$ **entonces return** $(0, (0, \ldots, 0))$
4: $\delta_c = - \mid \delta_c \mid$
5: **para** $i = 1$ **hasta** m **hacer**
6: $d \leftarrow$ Tipo de restricción (1 para '$\leq$' ; -1 para '$\geq$' ; 0 para '=')
7: **si** $(d \neq 0)$ **entonces**
8: $t \leftarrow a_{ij} \cdot (-1)^{x_j}$
9: **si** $(d \cdot Slack[i] < d \cdot t)$ **entonces return** $(0, (0, \ldots, 0))$
10: $\Delta_i = -t$
11: **si no**
12: **si** $(a_{ij} \neq 0)$ **entonces return** $(0, (0, \ldots, 0))$
13: **fin si**
14: **fin para**

La función *Bit_Flip* usa Δ-evaluación para computar eficientemente el valor de la función objetivo y de las restricciones sin necesidad de recalcular todo de nuevo (Chicano et al. [24]). Los parámetros de entrada son el propio vector de decisión, el índice de la variable a invertir y el vector *Slack*. El pseudocódigo puede verse en el Algoritmo 9. Inicialmente definimos un vector $\Delta = (0, \ldots, 0)$ que controla la variación del vector *Slack*. En la Línea 2 obtenemos el coeficiente de la función objetivo asociado a la variable x_j, es decir, $\delta_c = \sum_{i=1}^{p} c_{ij}$. Si este valor vale 0, entonces independientemente del valor del valor de x_j, sabemos que no habrá ninguna mejora en la función objetivo. Más aún, para el caso en que $\delta_c < 0$, si $x_j = 1$, y realizamos una mutación con cambio de bit, la variable j dejará de formar parte de la solución, así que el valor objetivo crecerá en $|\delta_c|$ unidades, por lo que podemos descartar esta inversión. Por el mismo motivo, si $\delta_c > 0$ y $x_j = 0$, el cambio de bit se descarta. Estas condiciones aparecen condensadas en la Línea 3. Si $\delta_c \cdot (-1)^{x_j} < 0$, entonces hemos superado la primera condición y procedemos a chequear si la inversión respeta también las restricciones. En la

UNIVERSIDAD DE MÁLAGA

Línea 4 establecemos δ_c como $-|\delta_c|$. Este será el incremento final en la función de *fitness* si finalmente la inversión es aceptada. De manera similar se chequean todas las restricciones en el bucle de las Líneas 5–14. Si alguna de las restricciones no se satisface (es violada), finalizamos la comprobación y no aceptamos la inversión, devolviendo $\delta_c = 0$.

Volvamos al Ejemplo 5.1. Partiendo de la solución inicial dada por el resolutor, $x = (1, 1, 0, 0, 1)$, vimos que el valor objetivo era 3. Analizamos si es posible realizar una inversión en la segunda componente. Tenemos que $x_2 = 1$ y $\delta_c = 1$. La condición de la Línea 3 es falsa y δ_c cambia su valor a $\delta_c = -1$ (Línea 4). Pasamos a la segunda parte y analizamos las restricciones. Para la primera de ellas, $d = 1$, $t = 2 \cdot (-1) = -2$ y se cumple que $d \cdot Slack[1] = 1 \cdot 4 \geq -2$. Por tanto, se cumple la restricción (no se cumple la condición de la Línea 9), y Δ_1 cambia a $\Delta_1 = 2$ (Línea 10). Esto significa que la holgura para la restricción aumentará en dos unidades si finalmente aceptamos esta inversión. Para la segunda restricción, $Slack[2] = -4$, $d = -1$, $t = 1 \cdot (-1) = -1$ y se cumple que $(-1) \cdot (-4) \geq (-1) \cdot (-1)$. También se cumple la restricción, por lo que Δ_2 cambia a $\Delta_2 = 1$. Así que, si finalmente aceptamos el cambio, la variable x_2 hará que la segunda restricción aumente su holgura en una unidad, lo que implica que estaremos más cerca de su saturación. Para la última restricción, $d = 0$ (restricción de igualdad) y $a_{ij} = 0$, por lo que también se acepta. Finalmente la función *Bit_Flip* devuelve $(-1, (2, 1, 0))$ aceptando el cambio para x_2. Al volver de la función *Hill_Climbing*, actualizamos el vector *Slack*, y finalmente nos queda $Slack = (4 + 2, -4 + 1, 0 + 0) = (6, -3, 0)$. El nuevo valor de *fitness* será $3 - 1 = 2$.

5.1.3. Resultados computacionales

En este apartado exponemos los resultados computacionales. Lo dividimos en tres partes. Inicialmente presentamos las instancias usadas así como los indicadores de calidad que empleamos. En la segunda parte definimos los parámetros de entrada asociados a los algoritmos usados en la comparativa, que serán, además de *MOFeLS*, las metaheurísticas NSGA-II, SPEA2, y MOEA/D. Tras mostrar una configuración de ciertos parámetros de CPLEX, así como de los parámetros asociados a las metaheurísticas, en la tercera parte exponemos los resultados numéricos de los experimentos.

Instancias e indicadores de calidad

Consideramos 28 instancias de problemas multiobjetivo binarios con restricciones lineales, extraídos de dos fuentes distintas de datos. Veinte de las instancias[2] provienen del trabajo de Kirlik et al. [102], y las otras ocho instancias corresponden a problemas de la mochila multidimensional multiobjetivo[3] (véase Ejemplo 2.25). En resumen, disponemos de las siguientes instancias:

- 10 instancias de un problema de asignación multiobjetivo de dimensión 3, con 50 agentes y tareas. Cada instancia tiene 2500 variables y 100 restricciones. Al grupo de todas ellas las llamaremos AP.
- 10 instancias de un problema de la mochila multiobjetivo con $p = 4$. Cada instancia tiene 40 variables y una restricción. El grupo de todas ellas se llamará KP.
- 4 instancias de un problema de la mochila multiobjetivo y multidimensional con $p = m = 3$. El número de variables es 100, 250, 500 y 750, respectivamente. A estas instancias las llamaremos MKP3_100, MKP3_250, MKP3_500 y MKP3_750.
- 4 instancias de un problema de la mochila multiobjetivo y multidimensional con $p = m = 4$. El número de variables es 100, 250, 500 y 750, respectivamente. A estas instancias las llamaremos desde MKP4_100 hasta MKP4_750.

Hemos usado las instancias para problemas de asignación multiobjetivo ya que poseen restricciones de igualdad. Además, los problemas de la mochila multiobjetivo y multidimensional con diferente número de mochilas y de objetivos son ampliamente usados en la literatura.

[2]Disponibles en `http://home.ku.edu.tr/~moolibrary/`.
[3]Disponibles en `https://sop.tik.ee.ethz.ch/download/supplementary/testProblemSuite/`.

UNIVERSIDAD DE MÁLAGA

Los indicadores de calidad utilizados han sido seleccionados con la idea de que el conjunto de todos ellos pueda dar una mayor información sobre la calidad de las soluciones y la buena dispersión de las mismas. Consideraremos las siguientes métricas:

- Número de soluciones obtenidas, ONVG, definido en la Ecuación (3.9).
- Hipervolumen y porcentaje de hipervolumen, HV y HVR, definidos en las Ecuaciones (3.10) y (3.16), respectivamente.
- Distancia generacional, GD, definido en la Ecuación (3.13).
- Distancia generacional inversa, IGD, definido en la Ecuación (3.14).
- Epsilon aditivo, ε_+, definido por la Ecuación (4.6).

El punto de referencia considerado a la hora de hallar el hipervolumen será el punto nadir cuando conozcamos el frente completo. En otro caso, dicho punto será $r = (r_1, \ldots, r_p)$, siendo $r_i = \text{máx}_{j=1}^{k} z_i^j$, $\forall i$, y $\{z^1, \ldots, z^k\}$ el conjunto de referencia. De manera similar, para las métricas GD, IGD y ε_+, adaptaremos sus definiciones y usaremos un conjunto de referencia R, en lugar de PF, ya que no lo conocemos. Así, la distancia generacional la obtendremos calculando $\text{GD}(N, R)$, la distancia generacional inversa haciendo $\text{IGD}(N, R)$ y el epsilon aditivo mediante $\varepsilon_+(N, R)$. El conjunto de referencia para cada instancia, R, se obtendrá como la unión filtrada de todos los frentes aproximados en los experimentos (para cualquier algoritmo considerado), más la unión de una ejecución independiente durante 40 minutos de esa instancia, más una nueva filtración de soluciones dominadas. Se ha considerado esta ejecución de 40 minutos porque cuanto mayor sea el tiempo de ejecución, mejor será la aproximación al frente de Pareto.

Parámetros de CPLEX y de los algoritmos usados en la comparativa

Los algoritmos que utilizaremos en los experimentos computacionales serán:

- Algoritmo híbrido *MOFeLS*[4].
- Metaheurística NSGA-II[5].
- Metaheurística SPEA2[5].
- Metaheurística MOEA/D[5].

Antes de realizar la ejecución de los algoritmos usamos el paquete IRACE (*iterated racing for automatic algorithm configuration*), desarrollado por López-Ibáñez et al. [121]. Este algoritmo ajusta los distintos parámetros de entrada y proporciona una buena configuración para dichos parámetros. Para el algoritmo *MOFeLS*, el parámetro *TILIM* varía en el conjunto {0.0001, 0.001, 0.01, 0.1, 0.2, 0.4, 0.6, 0.8, 1}, medido en segundos. IRACE seleccionó el valor 0.4. Para las otras tres metaheurísticas ejecutamos también IRACE para poder decidir una buena configuración para los siguientes parámetros: número de cortes en el operador de cruce, variando entre 1 y 5; probabilidad de cruce, con un valor libre en el intervalo $[0, 1]$; probabilidad de mutación, en el rango [0, 0.5]; y tamaño de la población, que varía desde 10 hasta 1000 con saltos de 10 en 10 individuos. Para la metaheurística MOEA/D [6], añadimos dos parámetros más en la configuración. El primero determina la probabilidad de seleccionar una solución en el vecindario frente a la población completa. Este valor está en el rango $[0, 1]$. El segundo configura el tamaño del vecindario, que varía desde 2 hasta 20. La configuración final dada por IRACE para los parámetros de las tres metaheurísticas se muestra en la Tabla 5.1.

El algoritmo *MOFeLS* emplea CPLEX 12.6.2 como resolutor. Fijamos algunos de sus parámetros (extraídos de la Tabla 3.2). El resto de parámetros tendrán su valor por defecto:

[4]Código fuente disponible en `https://github.com/MiguelAngelDominguezRios/MOFeLS`. Programado en C/C++.

[5]Código fuente extraído del paquete *jMetal* (Durillo y Nebro [55]), programado en Java y convenientemente adaptado para usar tiempo límite como condición de parada.

UNIVERSIDAD DE MÁLAGA

Tabla 5.1: Configuración de los parámetros para NSGA-II, SPEA2 y MOEA/D usando IRACE.

Algoritmo	Nº cortes en el cruce	Probabilidad de cruce	Probabilidad de mutación	Tamaño de la población	Probabilidad vecindario	Tamaño vecindario
NSGA-II	5	0.77	0.0009	310	-	-
SPEA2	4	0.70	0.0017	430	-	-
MOEA/D	2	0.86	0.0036	760	0.86	15

- CPX_PARAM_MIPEMPHASIS = 1. Este parámetro controla el balance entre factibilidad u optimalidad para las soluciones. Fijándolo a 1 estamos priorizando la busqueda de soluciones factibles. Su valor por defecto es 0, que indica un balance entre ambas opciones, realizando esfuerzo en encontrar soluciones factibles de buena calidad pero tratando también de encontrar el óptimo rápidamente.
- CPX_PARAM_INTSOLLIM = 1. Este parámetro fija el número de soluciones enteras encontradas por el resolutor antes de parar su ejecución. El valor por defecto es 2100000000.
- CPX_PARAM_TILIM = *TILIM*. Este parámetro mide el tiempo límite de ejecución (en segundos) por parte del resolutor. Su valor viene dado por el parámetro de entrada *TILIM* en el Algoritmo 7. El valor por defecto es 10^{75}.

Resultados numéricos

Para cada una de las 28 instancias consideradas, ejecutamos 30 veces cada algoritmo, usando un cluster con diez máquinas Intel Core 2 Quad (Q9400) CPU a 2.7 GHz, con 4 núcleos cada una, 11 GB de memoria y sistema operativo Ubuntu 16.04 LTS. Para cada ejecución utilizamos solamente 1 núcleo, 2 GB de RAM y 30 segundos de tiempo límite, reportando valores medios para cada uno de los indicadores de calidad considerados. Para los grupos de instancias AP y KP hemos considerado también los valores medios entre las 10 instancias dentro de cada grupo. Al ejecutar las tres metaheurísticas, solo consideramos las soluciones finales que sean factibles, es decir, aquellas que no violen ninguna de las restricciones. Posteriormente, filtramos los puntos dominados en el espacio objetivo. Este proceso se realiza *offline* para las tres metaheurísticas.

Tabla 5.2: Resultados computacionales para las instancias con frente de Pareto conocido. Los mejores resultados se marcan en negrita. Los valores indeterminados se marcan con guión.

		MOFeLS	NSGA-II	SPEA2	MOEA/D
ONVG	AP	**98.85**	0.00	0.00	0.00
	KP	**628.62**	97.40	119.46	122.41
	MKP3_100	**225.70**	40.03	42.27	94.03
HV	AP	**9.90E+07**	0.00E+00	0.00E+00	0.00E+00
	KP	**4.56E+14**	3.70E+14	3.72E+14	4.38E+14
	MKP3_100	**9.41E+11**	6.04E+11	6.52E+11	**9.41E+11**
IGD	AP	**0.54**	-	-	-
	KP	**6.57**	27.83	27.169	15.88
	MKP3_100	13.36	44.60	38.57	**12.18**
GD	AP	**1.94**	-	-	-
	KP	**0.78**	16.17	11.55	5.91
	MKP3_100	**8.55**	51.27	62.65	28.76
ε_+	AP	**0.11**	-	-	-
	KP	**0.11**	0.49	0.49	0.27
	MKP3_100	**0.22**	0.58	0.56	0.31

Dividimos las instancias en dos grupos: aquellas para las que conocemos el frente de Pareto completo tras usar el algoritmo exacto *TPA* que vimos en el Capítulo 4, y aquellas para las que no conocemos su frente completo. En este segundo caso, tomamos un frente de referencia que consiste en la unión de todos los frentes aproximados disponibles para esa instancia más la aproximación obtenida tras una ejecución durante 40 minutos del algoritmo *TPA* para esa

UNIVERSIDAD DE MÁLAGA

instancia. Después, filtramos todas las soluciones que sean dominadas. Las Tablas 5.2 y 5.3 muestran los resultados numéricos para estos dos grupos considerados.

Tabla 5.3: Resultados computacionales para las instancias con frente de Pareto desconocido. Los mejores resultados se marcan en negrita.

		MOFeLS	NSGA-II	SPEA2	MOEA/D
ONVG	MKP3_250	**438.63**	43.60	48.20	160.47
	MKP3_500	**430.33**	59.97	49.00	194.87
	MKP3_750	**283.87**	47.87	40.27	165.67
	MKP4_100	**393.03**	91.10	107.83	168.77
	MKP4_250	**546.03**	93.63	100.33	267.80
	MKP4_500	**384.50**	91.53	95.50	292.93
	MKP4_750	**307.67**	84.17	82.57	240.77
HV	MKP3_250	**5.25E+12**	2.63E+11	4.42E+11	3.79E+12
	MKP3_500	**6.31E+13**	2.02E+12	1.93E+12	1.96E+13
	MKP3_750	**1.89E+14**	4.13E+12	3.98E+12	2.48E+13
	MKP4_100	**8.06E+15**	6.5E+13	2.36E+14	2.55E+15
	MKP4_250	**2.32E+17**	1.77E+15	3.35E+15	4.12E+16
	MKP4_500	**5.57E+18**	2.08E+16	3.06E+16	3.18E+17
	MKP4_750	**1.69E+19**	4.88E+16	6.62E+16	3.56E+17
IGD	MKP3_250	**32.65**	86.43	84.88	43.47
	MKP3_500	**51.91**	132.16	176.40	110.91
	MKP3_750	**75.17**	255.64	354.76	268.28
	MKP4_100	**17.26**	39.69	32.97	19.20
	MKP4_250	**31.00**	69.93	73.86	48.40
	MKP4_500	**53.25**	129.94	175.09	126.95
	MKP4_750	**71.05**	254.02	356.08	266.17
GD	MKP3_250	**8.56**	519.09	615.60	138.02
	MKP3_500	**40.18**	1487.41	2369.15	732.73
	MKP3_750	**106.41**	3592.33	5588.16	1985.89
	MKP4_100	**31.22**	49.77	72.07	47.02
	MKP4_250	**60.16**	337.82	478.08	183.39
	MKP4_500	**280.19**	1409.77	2112.19	897.70
	MKP4_750	**396.11**	3167.34	4561.25	2008.69
ε_+	MKP3_250	0.26	0.41	0.35	**0.20**
	MKP3_500	0.19	0.33	0.27	**0.18**
	MKP3_750	0.23	0.36	0.27	**0.22**
	MKP4_100	**0.15**	0.47	0.42	0.24
	MKP4_250	**0.12**	0.44	0.40	0.30
	MKP4_500	**0.14**	0.42	0.35	0.32
	MKP4_750	**0.10**	0.33	0.24	0.24

En los resultados numéricos de la Tabla 5.2 podemos observar que ninguna de las tres metaheurísticas es capaz de encontrar alguna solución para el grupo AP. Esto es debido a que las metaheurísticas utilizan un operador de mutación de inversión de bits (*bit flip*), y cuando solo un bit cambia, alguna restricción es violada, dado el carácter que tiene este tipo de problema. Esto hace que sea alta la probabilidad de que la solución final obtenida por la metaheurística sea infactible. Aquí podemos ver una de las grandes ventajas del método *MOFeLS*, ya que, gracias al resolutor, encuentra fácilmente una solución factible, independientemente del tipo de restricciones que posea, y en particular, para aquellas que sean de igualdad. La búsqueda local se encarga de realizar los movimientos necesarios hasta alcanzar el óptimo local. Si no pudiese realizar ningún movimiento, la solución factible inicial terminaría siendo el óptimo local. Para las restantes instancias, observamos que *MOFeLS* encuentra los mejores valores en casi todas las métricas y para casi todas las instancias o grupos de instancias. El único caso en el que no se consigue el mejor resultado es para MKP3_100, donde el valor IGD es mejor para la metaheurística MOEA/D. Por otro lado, nótese que un valor bajo en la métrica GD significa que el frente aproximado está cerca del verdadero frente de Pareto.

En el segundo grupo de instancias analizado no disponemos del frente de Pareto completo, por lo que utilizamos el mencionado frente de referencia. Ahora, las métricas HV, GD, IGD y ε_+ serán menos informativas que en el caso anterior. Analizando la Tabla 5.3, comprobamos

UNIVERSIDAD DE MÁLAGA

que *MOFeLS* obtiene el mejor número medio de soluciones con una gran diferencia respecto a los otros tres algoritmos. Además, *MOFeLS* obtiene mejores valores para HV, IGD y GD en todos los casos. Para la métrica ε_+, no existe un claro ganador, ya que MOEA/D obtiene también resultados similares. No obstante hay que tener en cuenta que, dado que el número de soluciones en MOEA/D es menor que en *MOFeLS*, esto facilita a que el indicador ε_+ tome un valor menor.

Para justificar la buena dispersión de las soluciones obtenidas por *MOFeLS*, analizamos el porcentaje real de hipervolumen obtenido para las métricas de las que conocemos el frente de Pareto completo. Numeramos cada una de las instancias de los grupos AP y KP con un sufijo que indica el número de instancia. Los resultados de la Tabla 5.4 indican unos porcentajes bastante buenos de hipervolumen conseguido, teniendo en cuenta que solo se han usado 30 segundos de ejecución para cada instancia. Más aún, para las instancias del grupo KP los valores superan en la mayoría de los casos el 99 % del hipervolumen total. El valor IGD muestra la distancia media entre puntos del frente de Pareto y su punto aproximado más cercano. Estos valores son muy próximos a cero en el caso del grupo AP, lo cual sugiere también una buena dispersión de las soluciones por el espacio objetivo.

Tabla 5.4: Valores medios para HVR y IGD usando *MOFeLS* para cada instancia con frente de Pareto conocido.

	HVR	IGD		HVR	IGD
AP1	0.8511	0.53	KP1	0.9953	9.37
AP2	0.8549	0.56	KP2	0.9785	6.06
AP3	0.8562	0.57	KP3	0.9999	4.56
AP4	0.8494	0.55	KP4	0.9957	8.28
AP5	0.8666	0.53	KP5	0.9867	7.21
AP6	0.8620	0.56	KP6	0.9908	6.98
AP7	0.8635	0.56	KP7	0.9947	6.64
AP8	0.8533	0.47	KP8	0.9994	5.30
AP9	0.8636	0.51	KP9	0.9819	6.77
AP10	0.8770	0.52	KP10	0.9999	4.52
MKP3_100	0.9041	13.36			

Para finalizar este apartado de experimentos computacionales, realizamos los correspondientes contrastes de hipótesis. Para cada métrica, algoritmo e instancia, tomamos los valores medios después de las 30 ejecuciones durante 30 segundos cada una. Aplicamos el test no paramétrico de Wilcoxon de los rangos signados. Comparamos *MOFeLS* con cada una de las tres metaheurísticas a un nivel de significación $\alpha = 0.01$. Todos los p-valores obtenidos están por debajo de 10^{-5}, lo que nos permite concluir que *MOFeLS* es significativamente el mejor de los algoritmos considerando estas métricas y estas instancias.

5.2. Algoritmo *HBOXES*

En esta sección mostramos una nueva propuesta algorítmica, aún no publicada. Basándonos en la idea del algoritmo *MOFeLS*, en el que usamos el resolutor para obtener una solución factible y luego una búsqueda local se encarga de aproximar cada uno de los puntos del frente, en este caso dejaremos al resolutor que encuentre una solución aproximada, usando sus propios parámetros internos. Se trata, por tanto, de una *matheurística*, pero que también podría considerarse como método exacto si se ajustan los parámetros adecuadamente. Ya que esta nueva propuesta se basa en la estructura dada por el Algoritmo 3, en la que se exploran cajas en distintas regiones del espacio objetivo, recibirá el nombre de *HBOXES* (*Heuristic based on exploration of BOXES*). Dividimos esta sección en tres apartados: descripción de la idea y justificación del algoritmo, propuesta algorítmica propiamente dicha y resultados computacionales.

5.2.1. Descripción de la idea y justificación

Este nuevo algoritmo híbrido utiliza el resolutor ILP como algoritmo aproximado a la hora de resolver la parametrización de los distintos subproblemas. Estos subproblemas son

UNIVERSIDAD DE MÁLAGA

generados dinámicamente dirigiendo la búsqueda hacia regiones del espacio objetivo que no han sido exploradas. Mientras el resolutor ILP puede tener que realizar un esfuerzo computacional elevado para resolver algunos subproblemas de manera exacta, si limitamos algunos de sus parámetros internos y lo usamos solo para obtener soluciones aproximadas, podemos conseguir buen resultado con un coste computacional aceptable (tan aceptable como el decisor prefiera, pues uno de los parámetros a establecer es el tiempo máximo de ejecución permitido por el resolutor ILP en cada iteración).

El algoritmo *HBOXES* posee la ventaja de ser aplicable a cualquier problema combinatorio multiobjetivo (no solo a los binarios, como en el caso de *MOFeLS*), además de ser mucho más rápido en su ejecución que los algoritmos ILP exactos, en general. Basándonos en la estructura dada por el Algoritmo 3, usaremos CPLEX como resolutor, pero estableciendo condiciones de parada mediante la modificación de algunos de sus parámetros. De esta forma, la idea es obtener un buen frente de Pareto aproximado en poco tiempo. La motivación de este algoritmo viene dada tras la observación continuada de los experimentos computacionales en el algoritmo *TPA*, donde se comprobó que el resolutor solía encontrar en muy poco tiempo la solución que luego resultaría ser la óptima del subproblema, pero dedicaba mucho tiempo en confirmar la optimalidad de dicha solución.

Los parámetros de CPLEX que tendremos en cuenta como condición de parada (salvo que el resolutor encuentre antes el óptimo, o determine que el problema es infactible) son los siguientes (véase Tabla 3.2):

- CPX_PARAM_EPGAP. Controla la tolerancia relativa para el hueco o *gap* que hay entre el mejor valor objetivo y el objetivo del mejor nodo de los restantes a explorar en el árbol de búsqueda durante el *Branch & Cut*. Llamaremos *EPGAP* a este parámetro de entrada.

- CPX_PARAM_INTSOLLIM. Este parámetro de entrada fija el número de soluciones enteras encontradas por el resolutor antes de parar su ejecución. Lo llamaremos *FSL* (*Feasible Solution Limit*).

- CPX_PARAM_TILIM. Mide el tiempo límite de ejecución (en segundos) por parte del resolutor a la hora de resolver un subproblema. Lo llamaremos *TILIM* (*TIme LIMit*).

Si al resolver un subproblema el resolutor encuentra una solución óptima, procedemos de manera normal, y partimos el espacio de búsqueda en p nuevas cajas, usando *full p-split* y eliminando las cajas redundantes mediante el algoritmo RE. Así mismo, si el problema es infactible, descartamos la caja correspondiente y pasamos a la siguiente iteración. Si se da alguna de las condiciones de parada y disponemos de una solución factible, la consideraremos como solución aproximada y partiremos las cajas como si de una solución no dominada se tratase. Este hecho hace que *HBOXES* sea, en principio, un algoritmo aproximado. Esta forma de proceder no origina ningún problema, ya que al aplicar *full p-split* lo único que estamos haciendo es descartar la región dominada por la solución encontrada. Si resultase que dicha solución es también dominada, ya será eliminada en el futuro (si disponemos de tiempo suficiente de ejecución). Si la condición de parada se produjese porque en una iteración el resolutor ha alcanzado el número de soluciones factibles *FSL*, o el valor prefijado *EPGAP*, es claro que en estos casos ya disponemos de una solución factible. Sin embargo, para el caso de que la condición de parada se produzca tras haber transcurrido el tiempo de ejecución *TILIM*, podría ocurrir que no dispongamos aún de una solución factible.

Dividimos las regiones a explorar en dos niveles. Por un lado, una lista L contiene cajas ordenadas según su volumen (en orden no creciente), y otra lista *NB* guarda aquellas cajas para las que no se ha encontrado ninguna solución tras una primera exploración, para usar en un próximo barrido (*Next Barrier*) de cajas. Cuando la lista L esté vacía, todas las cajas de *NB* se traspasan a L y volvemos a empezar la exploración. Cabe la posibilidad de que el algoritmo no encuentre ninguna solución en una caja explorada. En este caso no podemos partirla, y por ello la guardaremos en la lista *NB*, que será explorada en el siguiente barrido, una vez haya finalizado la exploración de la lista de cajas actuales.

Al traspasar todas las cajas de *NB* hacia L, es necesario modificar el valor del parámetro *TILIM*, ya que hay que tener en cuenta que todas esas cajas no han encontrado ninguna solución factible en un tiempo *TILIM*, por lo que al explorarlas de nuevo considerando el mismo

UNIVERSIDAD DE MÁLAGA

tiempo prefijado, podríamos entrar en un bucle infinito. Por ello, añadiremos un parámetro de entrada en nuestro algoritmo que controlará el factor por el que se multiplicará el parámetro *TILIM* en el siguiente barrido, que deberá ser, obviamente, un número real mayor que uno.

5.2.2. Propuesta algorítmica

El Algoritmo 10 muestra el pseudocódigo de *HBOXES*. Los parámetros de entrada prefijados antes de la ejecución son: *FSL* (número de soluciones factibles a encontrar por el resolutor antes de parar), *EPGAP* (tolerancia relativa de la solución), *TILIM* (tiempo máximo de ejecución para cada subproblema), *FTILIM* (factor multiplicativo para *TILIM* en el siguiente barrido) y *MAXT* (tiempo máximo total de ejecución). La salida será un frente aproximado. Con la configuración adecuada podría considerarse *HBOXES* como método exacto, ya que si $FSL = TILIM = \infty$ (valores suficientemente grandes) y $EPGAP = 0$, el algoritmo es equivalente al Algoritmo 3.

Inicializamos el conjunto aproximado de soluciones N y las listas de cajas, NB y L al conjunto vacío. El programa P de la Línea 3 a considerar será el modelo híbrido de la Ecuación 3.3. La caja inicial U será la obtenida al resolver la relajación lineal del modelo, tal y como hicimos en el método *MOFeLS* (Línea 4 del Algoritmo 7). Los parámetros *FSL* y *EPGAP* son fijos durante todo el algoritmo y proporcionan, junto a *TILIM*, condiciones de parada en la exploración. El bucle externo de las Líneas 5–22 gestiona los barridos. En el barrido inicial, *TILIM* toma el valor dado como parámetro de entrada. Para los siguientes barridos, este valor se actualiza multiplicando por el factor *FTILIM*, a fin de dar más tiempo al resolutor para encontrar soluciones factibles, evitando entrar en un bucle infinito. Después de mover todas las cajas desde *NB* hasta L (Línea 7), entramos en el segundo bucle (Líneas 8–20). En él seleccionamos la siguiente caja a explorar, que será aquella que tenga mayor volumen, y resolvemos el correspondiente programa lineal. Tanto si obtenemos una solución óptima como si llegamos a una condición de parada disponiendo de una solución factible, añadimos la nueva solución (Línea 14), filtramos el correspondiente conjunto N y actualizamos la lista de cajas usando el algoritmo RE. Utilizamos la variable lógica *TILIM-FS* para el caso en que se alcanza el tiempo límite de ejecución y se ha encontrado una solución factible (*Feasible Solution*), y la variable lógica *TILIM-IS* (*Infeasible Solution*) para indicar que el resolutor no ha sido capaz de encontrar ninguna solución factible en *TILIM* segundos. En este último caso, asumimos que la caja podría estar vacía y, para no perder más tiempo explorándola, se traspasa al conjunto *NB* para que sea analizada nuevamente en el siguiente barrido (Línea 18). Por último, si la solución es infactible, la caja simplemente se elimina (Línea 10).

El algoritmo *HBOXES* está diseñado para que la mayoría de las soluciones (a priori) se encuentren durante el primer barrido. Con suficiente tiempo disponible de ejecución, y tras sucesivos barridos, el resolutor será capaz de determinar finalmente si las cajas exploradas contienen una solución factible o no. Obviamente el algoritmo resulta muy ineficiente en la ejecución si tiene que realizar múltiples barridos. Su diseño está pensado para conseguir buenas soluciones a corto plazo, preferiblemente durante el primer barrido.

5.2.3. Resultados computacionales

En este apartado realizamos la comparativa entre *HBOXES*, *MOFeLS* y *TPA*. Para que dicha comparativa sea coherente, el parámetro *TILIM* de *MOFeLS* se fija al mismo valor que el de *HBOXES*. Comenzamos describiendo las instancias utilizadas, continuando con los indicadores de calidad usados, para finalizar con el análisis de los resultados obtenidos.

Instancias

Para analizar *HBOXES* hemos aumentado considerablemente el número de instancias. Se consideran un total de 911, procedentes de distintos tipos de problema multiobjetivo combinatorios, con dimensiones que varían desde 2 hasta 6. Además de las clases de problemas multiobjetivo, como el de la mochila, asignación, mochila multidimensional y problemas MOILP, ya usados en otros experimentos, añadimos instancias del problema del recubrimiento mínimo (*SCP*) con 2 y 3 objetivos. En 629 de las 911 instancias conocemos el frente de Pareto. Para aquellas en las que no conocemos el frente completo, partimos de un frente inicial de referencia

UNIVERSIDAD DE MÁLAGA

Algoritmo 10 *HBOXES*

Entrada: *FSL*, *EPGAP*, *TILIM*, *FTILIM*, *MAXT*
Salida: N

```
1: N = ∅
2: NB = L = ∅
3: Establecer modelo P
4: NB ← U    //Caja inicial
5: mientras (NB ≠ ∅) and (tiempo <MAXT) hacer
6:    Asignar TILIM en P
7:    Mover todas las cajas desde NB hasta L
8:    mientras (L ≠ ∅) y (tiempo <MAXT) hacer
9:       B ← Selecciona la siguiente caja de L
10:      L ← L − {B}
11:      Resolver el modelo P(B)
12:      x ← Solución de P(B) y z = f(x)
13:      si (x es óptimo) o (FSL alcanzado) o (EPGAP alcanzado) o (TILIM-FS) entonces
14:         N ← N ∪ {z}
15:         N ← Filtrar puntos dominados en N
16:         Actualiza L
17:      si no si (TILIM-IS) entonces
18:         NB ← NB ∪ {B}
19:      fin si
20:   fin mientras
21:   TILIM = TILIM · FTILIM
22: fin mientras
```

consistente en ejecutar *TPA* durante 4 horas. En los casos en que la memoria disponible de la máquina lo permitiese, se ha ejecutado también la correspondiente instancia durante 24 horas. En varios casos el frente de referencia inicial ha llegado a tener más de 100 000 elementos, lo que ha conllevado a tener que invertir más tiempo en calcular las métricas correspondientes que en ejecutar los propios experimentos. Las instancias consideradas, obtenidas desde distintas fuentes de datos existentes en la literatura, se resumen en la Tabla 5.5.

Tabla 5.5: Instancias usadas en los resultados computacionales de esta sección. Las columnas indican el tipo de problema multiobjetivo, la dimensión, el número de instancias que contiene, en cuantas de ellas conocemos el frente de Pareto completo, así como los rangos del número de variables y restricciones en cada clase.

Tipo	p	Nº	*PF* conoc.	Variables (n)	Restricciones
MOKP	3	100	100	$\{10, 20, \ldots, 100\}$	1
	4	40	40	$\{10, 20, 30, 40\}$	1
	5	20	20	$\{10, 20\}$	1
MOAP	3	100	100	$\{5^2, 10^2, \ldots, 50^2\}$	$2n$
	4	190	36	$\{10^2, 15^2, \ldots, 100^2\}$	$2n$
	5	80	26	$\{8^2, 10^2, \ldots, 22^2\}$	$2n$
	6	100	35	$\{4^2, 6^2, \ldots, 22^2\}$	$2n$
MOILP	3	100	100	$\{10, 20, \ldots, 100\}$	$n/2$
	4	80	80	$\{10, 20, \ldots, 80\}$	$n/2$
	5	40	40	$\{10, 20, 30, 40\}$	$n/2$
MOMKP	2	3	3	$\{250, 500, 750\}$	2
	3	4	1	$\{100, 250, 500, 750\}$	3
	4	4	0	$\{100, 250, 500, 750\}$	4
MOSCP	2	44	42	$\{100, 200, 400, 600, 800, 1000\}$	$\{10, 40, 60, 100, 200\}$
	3	6	6	$\{100, 600\}$	$\{10, 60\}$
		911	629		

A continuación mostramos los enlaces de donde han sido extraídas todas las instancias:

UNIVERSIDAD DE MÁLAGA

- Problemas *MOKP*, *MOAP* para $p = 3$ y MOILP. Instancias de Gokhan Kirlik (2014). `http://home.ku.edu.tr/~moolibrary/`
- Problemas *MOAP* para $p \in \{4, 5, 6\}$. Instancias MOASLIB de Fritz Bökler (2015). `https://ls11-www.cs.tu-dortmund.de/staff/boekler/moco-instances`
- Problemas *MOMKP*. Instancias de Zitler-Laumanns (2001). `https://sop.tik.ee.ethz.ch/download/supplementary/testProblemSuite/?page=testProblem.php#source`
- Problemas *MOSCP*. Instancias de Thibaut Lust (2017). `https://webia.lip6.fr/~lustt/Research.html#Main`

En los últimos años se está poniendo de manifiesto en la comunidad científica la necesidad de mantener el código fuente de los algoritmos o los bancos de instancias en algún repositorio que perdure indefinidamente en el tiempo, evitando alojar los datos en servidores asociados a organizaciones o universidades, y que podrían no estar operativos cuando el autor no dependa de dicha institución (como bien podría ocurrir si el lector accede a alguno de los enlaces anteriores en un momento posterior al de la publicación de esta tesis).

Se ejecuta cada instancia una única vez para cada algoritmo. Si bien sería más recomendable realizar múltiples ejecuciones de cada una de ellas y tomar valores medios, solo hemos considerado una ejecución por el motivo comentado anteriormente: algunos frentes de referencia poseen muchos puntos y el cálculo del hipervolumen resulta muy costoso.

Indicadores de calidad

Como métricas para medir la calidad de las soluciones de *HBOXES* utilizaremos ONVG, HVR, GD e IGD+. En este caso, en lugar de considerar la métrica IGD consideraremos IGD+ debido a que este indicador es débilmente Pareto *compliant* [92].

Definición 5.1. Dado un conjunto aproximado $N = \{a_1, \ldots, a_{|N|}\}$, formado por vectores de dimensión p, y su frente de referencia, $R = \{z^1, \ldots, z^{|R|}\}$, definimos la métrica IGD+ como

$$\text{IGD+}(N, R) = \frac{\sum_{i=1}^{|R|} \text{mín}_{a \in N} \{d_i^+(a)\}}{|R|}, \tag{5.2}$$

siendo $d_i^+(a) = \sqrt{\sum_{j=1}^{p} \left(\text{máx}\{a_j - z_j^i, 0\}\right)^2}$ una nueva forma de medir la 'distancia' entre un vector de R y uno de N.

El frente de referencia considerado para cada instancia es el frente resultante tras filtrar la unión del frente inicial de referencia, más el frente originado por los algoritmos *TPA*, *MOFeLS* y *HBOXES* durante la ejecución. El punto de referencia para el cálculo del hipervolumen será el punto nadir del frente de referencia.

Experimentos preliminares

Ejecutamos cada instancia para cada algoritmo durante 30 segundos en un cluster compuesto por varias máquinas con procesador de 2.66 GHz, y cuatro núcleos cada una. En cada ejecución se utilizan 2 GB de RAM y un único núcleo. La versión de CPLEX utilizada es la 12.6.2. Mostraremos los resultados computacionales para cada una de las métricas por separado.

En la Tabla 5.6 observamos los resultados de las ejecuciones considerando ONVG. El primer bloque de tres columnas indica la clase del problema, su dimensión y el número de instancias que contiene dicha clase. La columna etiquetada con '+' indica en cuantas de ellas la diferencia de valores entre *HBOXES* y el correspondiente algoritmo en la comparación (*TPA* o *MOFeLS*) es positiva. La columna etiquetada con '−' indica en cuantas de ellas la diferencia es negativa. Obsérvese que cuando la suma de las columnas '+' y '−' no coincide con el total de instancias, esto quiere decir que los valores obtenidos son iguales para las restantes, es decir, la diferencia es cero (probablemente porque ambos algoritmos han finalizado su ejecución antes de tiempo y han obtenido el frente completo). La columna $\overline{\text{Dif}}$ indica la media de la diferencia de los valores de dicha métrica en los dos algoritmos, empezando siempre

UNIVERSIDAD DE MÁLAGA

Tabla 5.6: Análisis de resultados al ejecutar las 911 instancias y evaluar con ONVG. Cuanto mayor es el valor de '+' y mayor es el valor de $\overline{\text{Dif}}$, mejor son los resultados para *HBOXES*.

ONVG			*HBOXES* vs *TPA*				*HBOXES* vs *MOFeLS*			
	p	Inst	+	−	$\overline{\text{Dif}}$	Desv	+	−	$\overline{\text{Dif}}$	Desv
MOKP	3	100	1	61	-80.1	76.9	0	62	-58.4	68
MOKP	4	40	0	19	-47.2	64.4	0	19	-44.1	61
MOKP	5	20	0	4	-5.6	16.3	0	4	-5.8	16.9
MOAP	3	100	23	57	-33.9	57.5	**46**	**35**	**4.5**	**43.1**
MOAP	4	190	**144**	**46**	**9.9**	**90**	**125**	**65**	**1.6**	**83**
MOAP	5	80	0	80	-177.1	75.1	0	80	-219.9	105.1
MOAP	6	100	8	81	-98.9	90.7	1	88	-165.2	121.7
MOILP	3	100	**44**	**24**	**5.7**	**23.5**	**100**	**0**	**59.4**	**38.5**
MOILP	4	80	26	32	-6.2	39.8	**80**	**0**	**90.6**	**74**
MOILP	5	40	1	27	-50.7	58	**40**	**0**	**159.8**	**93.6**
MOMKP	2	3	**3**	**0**	**99**	**10.1**	1	2	-106	115.2
MOMKP	3	4	**4**	**0**	**74**	**24.9**	0	4	-298.5	75.2
MOMKP	4	4	**4**	**0**	**83.2**	**11.4**	0	4	-298.2	132.2
MOSCP	2	44	14	16	-3.2	16.7	**16**	**13**	**9.9**	**33.3**
MOSCP	3	6	1	5	-77	95.6	2	4	-2	55.2

por *HBOXES*. La columna Desv indica la correspondiente desviación típica de la variable diferencia. En negrita se marcan aquellas filas en las que podría decirse que el algoritmo *HBOXES* se comporta mejor que el otro, debido a que el valor de la diferencia de la métrica es mejor un número superior de veces para la correspondiente clase. El mismo tratamiento se aplicará en las sucesivas tablas que contienen los resultados para las otras tres métricas.

Observando la Tabla 5.6 podemos apreciar que en algunos casos existe una diferencia media muy elevada, como en el caso de *MOAP* con $p = 5$, donde dicha diferencia es de aproximadamente 177 soluciones. También encontramos desviaciones típicas muy elevadas, de hasta más de 90 soluciones, como en el caso de *MOSCP* con $p = 6$. Nótese además que, dado que las variables asociadas a la clase MOILP no son binarias, el algoritmo *MOFeLS* no puede ejecutarse, por lo que nunca podrá tener un mejor resultado que *HBOXES* en estos casos. Se puede apreciar que en la mayoría de los casos tanto *TPA* como *MOFeLS* obtienen un mayor número medio de soluciones. Las justificaciones de estas observaciones podrían ser las siguientes:

- El algoritmo *TPA* genera soluciones exactas en cada iteración. Si el problema no es muy duro, generará muchas soluciones en poco tiempo. Sin embargo, *HBOXES*, si bien podría emplear un menor tiempo de ejecución en cada iteración, la solución obtenida no tiene por qué ser exacta. Las cajas se partirán tras ese punto obtenido, pero después de sucesivas iteraciones, muchos de esos puntos podrían ser dominados por otros en el futuro, dando como resultado final un menor número de soluciones no dominadas. La gran ventaja de *HBOXES* podemos verla en el caso de las instancias MOMKP, que son las más duras, y ahí es donde se aprecia el potencial de emplear el resolutor sin tener que esperar a que termine la ejecución. De hecho, en varias de esas instancias, CPLEX es incapaz de encontrar una solución en *TPA* tras varias horas de ejecución.

- Respecto a la justificación de por qué *MOFeLS* genera más soluciones que *HBOXES*, el primero parte de una solución factible y luego realiza una búsqueda local, y esto podría ser siempre menos costoso que ir buscando sucesivas soluciones factibles. Habrá que ver también si la calidad de dichas soluciones mejora o no a las obtenidas mediante *HBOXES*. Lo comprobaremos analizando las restantes métricas.

En la Tabla 5.7 mostramos los resultados para la métrica HVR (véase Ecuación 3.16). Nótese que los resultados son, en cierta manera, parecidos a los obtenidos para ONVG. Al contar con frentes de Pareto que tienen muchos elementos, *MOFeLS* es capaz de encontrar muchas más soluciones, y esto generará ineludiblemente un mayor hipervolumen. Podemos observar en las instancias MOILP para $p = 3$ que no en todos los casos gana *HBOXES* a *MOFeLS*, a pesar de que este algoritmo no se puede ejecutar para este tipo de instancias. Ello

UNIVERSIDAD DE MÁLAGA

Tabla 5.7: Análisis de resultados al ejecutar las 911 instancias y evaluar con HVR. Cuanto mayor es el valor de '+' y mayor es el valor de $\overline{\text{Dif}}$, mejor son los resultados para *HBOXES*.

HVR			*HBOXES* vs *TPA*				*HBOXES* vs *MOFeLS*			
	p	Inst	+	−	$\overline{\text{Dif}}$	Desv	+	−	$\overline{\text{Dif}}$	Desv
MOKP	3	100	0	62	-0.0128	0.017	19	43	-0.0046	0.01
MOKP	4	40	0	19	-0.0019	0.005	4	15	-0.001	0.003
MOKP	5	20	0	4	-4.5e-05	1.8e-04	0	4	-4.3e-05	1.7e-04
MOAP	3	100	0	81	-0.0451	0.031	30	51	-0.006	0.012
MOAP	4	190	33	157	-0.0285	0.12	**104**	**86**	**0.049**	**0.135**
MOAP	5	80	0	80	-0.0325	0.016	3	77	-0.0302	0.02
MOAP	6	100	**51**	**38**	**0.0171**	**0.036**	14	75	-0.017	0.02
MOILP	3	100	**35**	**33**	**0.0559**	**0.174**	**98**	**0**	**0.9345**	**0.145**
MOILP	4	80	25	34	0.0353	0.117	**80**	**0**	**0.9509**	**0.057**
MOILP	5	40	0	28	-0.01	0.016	**40**	**0**	**0.9747**	**0.045**
MOMKP	2	3	0	3	-0.0721	0.051	0	3	-0.1303	0.104
MOMKP	3	4	**4**	**0**	**0.5629**	**0.352**	0	4	-0.1728	0.11
MOMKP	4	4	**4**	**0**	**0.7034**	**0.069**	0	4	-0.2587	0.101
MOSCP	2	44	13	17	0.1428	0.306	**18**	**12**	**0.0246**	**0.112**
MOSCP	3	6	**3**	**3**	**0.0027**	**0.016**	**4**	**2**	**0.0084**	**0.01**

es debido a que ni siquiera *HBOXES* ha sido capaz de encontrar una solución en 30 segundos (tal vez no la haya), por lo que se produce un empate técnico. Las diferencias más altas en HVR se producen en las instancias *MOMKP* con $p > 2$, donde hay un incremento de hasta un 70 % de mejora frente a *TPA*. Esto refuerza la idea de que nuestra propuesta algorítmica será tanto mejor cuando más duro sea el problema. En relación a la comparativa frente a *MOFeLS* la mayor diferencia se encuentra en *MOMKP* para $p = 4$, donde *MOFeLS* mejora en más de un 25 % los valores del hipervolumen. Para el resto de instancias, las diferencias no parecen ser muy significativas. Para la clase *MOSCP* con $p = 3$, *HBOXES* y *TPA* ganan en tres instancias cada uno. Ya que la diferencia de medias es positiva, la marcamos también en negrita, en favor de *HBOXES*.

Tabla 5.8: Análisis de resultados al ejecutar las 911 instancias y evaluar con IGD+. Cuanto mayor es el valor de '−' y menor es el valor de $\overline{\text{Dif}}$, mejor son los resultados para *HBOXES*.

IGD+			*HBOXES* vs *TPA*				*HBOXES* vs *MOFeLS*			
	p	Inst	+	−	$\overline{\text{Dif}}$	Desv	+	−	$\overline{\text{Dif}}$	Desv
MOKP	3	100	60	2	12.6855	19.248	42	20	4.8274	14.394
MOKP	4	40	15	4	2.2654	5.85	13	6	1.252	5.167
MOKP	5	20	4	0	1.0491	3.307	4	0	1.066	3.299
MOAP	3	100	81	0	3.3077	3.283	60	21	0.5678	1.273
MOAP	4	190	101	89	37.3111	105.241	**91**	**99**	**-254.3866**	**607.806**
MOAP	5	80	**18**	**62**	**-2.3836**	**2.129**	61	19	0.8088	1.178
MOAP	6	100	**20**	**69**	**-1.704**	**1.74**	70	19	0.7857	1.13
MOILP	3	100	**18**	**50**	**-20.0375**	**45.089**	**0**	**100**	**0**	**0**
MOILP	4	80	**25**	**34**	**-18.3449**	**35.628**	**0**	**80**	**0**	**0**
MOILP	5	40	21	7	-1.8048	11.86	**0**	**40**	**0**	**0**
MOMKP	2	3	3	0	101.034	116.058	3	0	272.3899	300.404
MOMKP	3	4	**0**	**4**	**-40.9081**	**0**	4	0	473.5882	534.868
MOMKP	4	4	**0**	**4**	**0**	**0**	4	0	568.0338	622.027
MOSCP	2	44	18	12	-56.9594	186.847	**10**	**20**	**-19.4282**	**132.812**
MOSCP	3	6	**3**	**3**	**-112.0872**	**142.135**	**2**	**4**	**-19.9164**	**32.354**

En la Tabla 5.8 realizamos un análisis de resultados para la métrica IGD+. Ya que en este caso deseamos minimizar su valor, estamos interesados en que haya más casos favorables en las columnas '−' de la tabla, así como una diferencia media cuanto más negativa mejor. Cuando algún algoritmo no consigue obtener ninguna solución y el otro sí, se considera como ganador para esa instancia aquel que encuentre alguna solución, pero a falta de poder asignar un valor IGD+ al que no obtiene ninguna, suprimimos ese dato en el cálculo de $\overline{\text{Dif}}$. Esto

UNIVERSIDAD DE MÁLAGA

puede observarse en la fila *MOMKP* para $p = 4$, donde la diferencia media se considera 0, debido a que *TPA* no encuentra ninguna solución en ninguna de las cuatro instancias. También ocurre lo mismo para todas las instancias MOILP frente a *MOFeLS*. Nuevamente vemos una gran diferencia media en favor de *MOFeLS* para las instancias *MOMKP*. En el resto de casos no hay diferencias notables, salvo para *MOAP* con $p = 4$, donde la diferencia es clara en favor de *HBOXES*, aunque es mejor aún para el caso de *TPA*. Habría que analizar por qué es tan grande esta diferencia para los problemas de asignación multiobjetivo cuando $p = 4$ y no así para $p = 5$ o $p = 6$. Podría ser debido a que los frentes de referencia tienen en torno a 100 000 puntos para $p = 4$, mientras que para el resto de dimensiones no ha sido posible obtener tantos puntos no dominados (en muchos casos debido al desbordamiento de memoria al usar *TPA*). Volviendo a lo anterior, nuevamente parece ponerse de manifiesto que la búsqueda local sigue siendo más efectiva que encontrar sucesivas soluciones factibles a partir del resolutor.

Tabla 5.9: Análisis de resultados al ejecutar las 911 instancias y evaluar con GD. Cuanto mayor es el valor de '−' y menor es el valor de $\overline{\text{Dif}}$, mejor para *HBOXES*.

GD			*HBOXES* vs *TPA*				*HBOXES* vs *MOFeLS*			
	p	Inst	+	−	$\overline{\text{Dif}}$	Desv	+	−	$\overline{\text{Dif}}$	Desv
MOKP	3	100	17	3	0.0217	0.069	**0**	**58**	**-5.7365**	**8.043**
MOKP	4	40	1	0	0.0182	0.115	**0**	**9**	**-0.5841**	**1.449**
MOKP	5	20	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
MOAP	3	100	64	3	0.0343	0.078	**0**	**80**	**-2.1918**	**2.294**
MOAP	4	190	100	0	87.3592	324.852	**0**	**189**	**-488.5097**	**676.023**
MOAP	5	80	12	0	0.0103	0.027	**0**	**79**	**-1.902**	**1.407**
MOAP	6	100	8	0	0.0065	0.026	**0**	**79**	**-3.357**	**3.155**
MOILP	3	100	43	4	1.2647	2.189	**0**	**100**	**0**	**0**
MOILP	4	80	26	1	0.9599	1.866	**0**	**80**	**0**	**0**
MOILP	5	40	**0**	**0**	**0**	**0**	**0**	**40**	**0**	**0**
MOMKP	2	3	3	0	0.6089	0.272	**0**	**3**	**-5.7831**	**1.308**
MOMKP	3	4	**1**	**3**	**0.2446**	**0**	**0**	**4**	**-2.8143**	**2.66**
MOMKP	4	4	**0**	**4**	**0**	**0**	**0**	**4**	**-7.3738**	**7.572**
MOSCP	2	44	18	6	14.2695	39.659	**4**	**22**	**-9.2661**	**46.337**
MOSCP	3	6	5	0	5.6524	9.664	**0**	**5**	**-40.6725**	**41.489**

Por último analizaremos el indicador de calidad GD, cuyos resultados se ofrecen en la Tabla 5.9. Esta métrica hay que usarla con cautela a la hora de comparar frentes aproximados y nunca debería ser la única a utilizar, ya que la medida no es Pareto *compliant*. No obstante, GD es un descriptor más de la calidad del frente, y significativo a la hora de estimar la cercanía de sus puntos al frente de Pareto. Al disponer del frente completo en muchas de las instancias, esta métrica resulta más significativa a la hora de interpretar los resultados. Nótese que el algoritmo *TPA* es exacto, por lo que la métrica GD siempre vale 0 y no podrá ser mejorado en ningún caso. En la tabla podemos observar que hay algunos casos aislados donde *HBOXES* gana a *TPA* (valores positivos en la columna '−'). Esto es debido a que se han usado frentes de referencia aproximados que contienen puntos que no son del frente de Pareto. Obsérvese que en los casos de *MOKP* con $p = 5$ y MOILP con $p = 5$, *HBOXES* y *TPA* quedan empatados. Esto nos hace concluir que todas las soluciones obtenidas por *HBOXES* son exactas y constituyen un subconjunto del frente de Pareto (lo mismo ocurre para *MOFeLS* en el caso *MOKP* con $p = 5$). Sin embargo, al comparar con *MOFeLS* en el resto de clases, podemos observar que en todos los casos *HBOXES* tiene un valor mejor para GD, lo que nos hace conjeturar que la calidad de las soluciones es mejor que las proporcionadas por la búsqueda local. El análisis de esta última métrica es la que nos hace ver el verdadero potencial que podría tener el utilizar un resolutor de manera controlada (obteniendo soluciones factibles poco a poco y controlando el tiempo máximo de ejecución), frente al uso de heurísticas basadas en búsqueda local. En definitiva, la calidad de las soluciones de *HBOXES* parecen ser bastante mejores que las que proporciona *MOFeLS*.

Como conclusión del análisis de los resultados en base a cuatro indicadores distintos, manifestamos que aunque *MOFeLS* proporciona mejor número de soluciones e hipervolumen en la mayoría de los casos, esta diferencia se reduce al calcular IGD+, y en ningún caso es

UNIVERSIDAD DE MÁLAGA

mejor para la métrica GD, lo cual nos hace conjeturar que esta propuesta inicial de *HBOXES* es prometedora, al menos si buscamos calidad en las soluciones obtenidas.

5.3. Conclusiones y trabajo futuro

En la primera sección de este capítulo hemos presentado una *matheurística* o algoritmo híbrido que resuelve problemas MOBO, presentando una buena dispersión de las soluciones por el espacio objetivo, y con buenos resultados si lo comparamos con las clásicas metaheurísticas NSGA-II, SPEA2 y MOEA/D. Una de las ventajas principales que posee *MOFeLS* es que es capaz de tratar con cualquier número de restricciones lineales de manera eficiente, incluso con restricciones de igualdad. Las restricciones de igualdad son un handicap para las metaheurísticas que usan la inversión de bits como operador de mutación. El algoritmo está basado en el método propuesto por Dächert y Klamroth [36], particularizando la selección de la siguiente caja a explorar como aquella con mayor volumen, usando *full p-split* para partir las cajas y filtrando las cajas redundantes mediante el algoritmo RE. *MOFeLS* usa solo esta estructura para seleccionar cajas bien dispersas por el espacio objetivo. Las soluciones obtenidas son aproximadas, ya que solo utilizamos el resolutor para obtener una solución factible. A partir de ahí se realiza una búsqueda local que permite un ahorro de tiempo computacional considerable. Tras la obtención de una nueva solución aproximada, se eliminan aquellas que sean dominadas.

En los experimentos computacionales, *MOFeLS* proporciona los mejores conjuntos aproximados en términos de número de soluciones, hipervolumen, distancia generacional y distancia generacional inversa. Para el indicador epsilon aditivo los mejores resultados son compartidos por *MOFeLS* y MOEA/D, aunque estadísticamente se confirma que nuestra propuesta es significativamente mejor para un nivel de significación $\alpha = 0.01$.

Estos resultados preliminares tan sorprendentemente buenos nos animan a seguir investigando. Proponemos como trabajo futuro la comparación de *MOFeLS* con otros algoritmos híbridos existentes en la literatura, usando una variedad mayor de bancos de instancias. También sería interesante extender la investigación para problemas con dimensiones mayores, incluso con restricciones no lineales, usando las ventajas de la optimización de caja gris (*gray-box*) para problemas multiobjetivo con restricciones (Chicano et al. [25]). Otra propuesta de investigación futura sería la de analizar los resultados computacionales utilizando otras herramientas (incluyendo gráficas) que permitan confirmar el buen comportamiento *anytime* del algoritmo *MOFeLS*.

El uso de un resolutor ILP de manera controlada, ajustando sus parámetros, permite un amplio abanico de posibilidades. En la segunda sección de este capítulo hemos diseñado el algoritmo *HBOXES*, que utiliza una estructura similar a la de los algoritmos *TPA* y *MOFeLS*. La forma de partir las cajas en *HBOXES* se realiza usando *full p-split*, ya que el uso de *p-partition* ha mostrado un rendimiento bastante menor en la experimentación computacional previa. Al usar *p-partition*, y dado que las soluciones encontradas son aproximadas, se parten muchas más cajas y existen menos cajas redundantes que eliminar. Esto hace que el rendimiento sea menor cuanto más tiempo de ejecución se emplee, porque el número de cajas a mantener usando *p-partition* es mucho mayor.

HBOXES controla principalmente tres parámetros a la hora de decidir parar la búsqueda (salvo en los casos en que se confirme la obtención del óptimo o que haya infactibilidad). Sin embargo, el resolutor CPLEX dispone de otros parámetros que también podrían configurarse y tener alguna influencia en la rapidez a la hora de encontrar soluciones:

- CPX_PARAM_BTTOL. Controla la frecuencia con la que se realiza el *backtracking* durante el proceso de ramificación en el *Branch & Cut*. Su valor se establece en un número real entre 0 y 1.

- CPX_PARAM_NODESEL. Establece una regla de selección del siguiente nodo a procesar en el *backtracking*. Se puede buscar primero en profundidad, mediante aquel nodo que posea el mejor límite (valor por defecto), en la resolución del problema relajado, mediante estimación óptima, o mediante estimación óptima alternativa.

UNIVERSIDAD DE MÁLAGA

- CPX_PARAM_VARSEL. Selecciona una regla de ramificación del nodo, que puede ser aquel que tenga mínima infactibilidad, o ramificar la variable que tenga máxima infactibilidad, ramificar en base a pseudocostes, u otras opciones, permitiendo también la opción de que CPLEX elija que criterio usar.

- CPX_PARAM_BBINTERVAL. Establece el mejor intervalo de cota en la estrategia *MIP* (*Mixed Integer Programming*). Fijar el valor a 0 significa que nunca seleccionará el nodo con mejor cota. Si el valor es 1 siempre lo seleccionará. A mayores valores del parámetro, el nodo con mejor cota será seleccionado con menor frecuencia. Su valor por defecto es 7.

- CPX_PARAM_BRDIR. Decide la rama a considerar, si la superior o la inferior del árbol, permitiendo a CPLEX elegir según el tipo de problema (valor por defecto).

Para manejar un gran número de parámetros sería interesante utilizar alguna herramienta que los ajuste, evitando la comprobación manual del rendimiento de cada uno de ellos.

Otra opción interesante a considerar es que al explorar una caja en la que no obtengamos la solución óptima, podríamos guardar la solución obtenida hasta el momento, explorar otras cajas, y retomar la anterior en un futuro, continuando con la búsqueda del óptimo a partir de la última solución encontrada en esa caja. Esto puede realizarse fácilmente usando la estructura *MIPSTARTS* que proporciona CPLEX. Sin embargo, experimentos preliminares arrojaron resultados pocos convincentes, ya que guardar la solución obtenida hasta el momento, sin partir la caja en cuestión, hacía ralentizar el rendimiento algorítmico, por lo que habrá que profundizar más en esta cuestión y ver hasta qué punto sería interesante incorporar esta herramienta en el diseño de un algoritmo como *HBOXES*.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Parte III

Aplicaciones

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Capítulo 6

Problema de Partición de Módulos Software

Presentamos en este capítulo un problema perteneciente al campo de la Ingeniería del Software. Se trata del problema de partición de módulos software, con un interés notable y de amplia aplicación y uso en este ámbito. Comenzamos con la descripción del problema, y mostramos dos algoritmos que lo resuelven de manera exacta. El primero de ellos es un algoritmo de ramificación y poda, con un interés más bien teórico, pues el criterio de poda es tan exigente que lo convierte prácticamente en un algoritmo enumerativo. En el segundo algoritmo modelamos el problema mediante un programa no lineal y con función objetivo que es suma de fracciones lineales. El interés principal de este algoritmo es la posibilidad de linealizar completamente el problema, dejando la posibilidad de que un resolutor ILP lo resuelva de manera exacta en una única iteración. En un siguiente apartado se muestran resultados computacionales tras la ejecución de varias instancias reales. A continuación se exponen brevemente algunas ideas surgidas en una investigación ampliada del problema y se finaliza el capítulo exponiendo las conclusiones. Este problema ha sido trabajado también en su versión biobjetivo, como se comentará al final del capítulo. Sin embargo, los resultados computacionales no fueron satisfactorios, y observamos que para este problema concreto la versión monoobjetivo se comporta mejor.

6.1. Descripción del problema

El problema de partición de módulos software (en adelante *SMC*, del inglés *Software Module Clustering*) fue inicialmente abordado por Mancoridis et al. [124]. La idea principal consiste en proporcionar una estructura modular de un sistema software a partir de su código fuente. La modularización del sistema se trata como un problema de optimización, que generalmente se resuelve mediante el uso de algoritmos genéticos y búsqueda local.

Dividir en subsistemas un código fuente mejora la calidad de su desarrollo así como la gestión de su mantenimiento. Sin embargo, para sistemas con cientos de miles de líneas de código empieza a resultar complicado realizar esta división manualmente. Lo que se suele hacer en estos casos es dividir los sistemas en subsistemas o *clusters* de módulos.

Para la resolución del problema *SMC*, Mancoridis et al. representaron el sistema de módulos como un grafo, donde cada nodo representa un módulo, y cada arista representa una relación entre módulos. Dicho grafo se conoce como *grafo de dependencia de módulos*, MDG (en inglés, *Module Dependency Graph*). Puede verse en la Figura 6.1 un ejemplo de un MDG, extraído del propio artículo de Mancoridis et al. [124].

Una vez determinado el MDG, el siguiente paso es diseñar un algoritmo que realice una partición de los nodos de dicho grafo de forma que queden identificados los distintos subsistemas. Dichos subsistemas permitirán un mejor desarrollo del software así como una mejor mantenibilidad del mismo. Para medir la calidad de la partición, Mancoridis et al. definieron dos medidas:

- Intra-conectividad, que mide la cohesión entre módulos pertenecientes a un mismo subsistema. El objetivo es maximizar esta medida.
- Inter-conectividad, que mide el acoplamiento existente entre módulos que pertenecen a distintos subsistemas. El objetivo es minimizar esta medida.

UNIVERSIDAD DE MÁLAGA

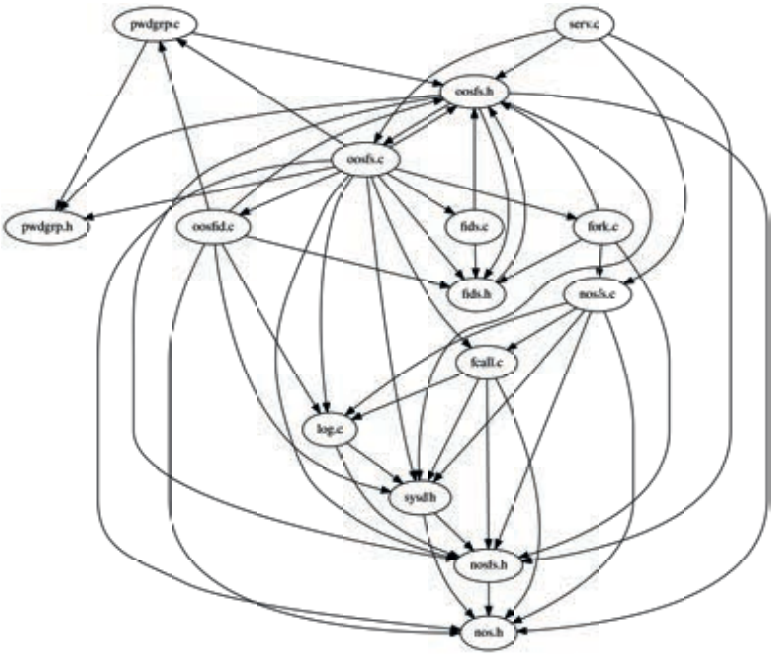


FIGURA 6.1: Ejemplo de grafo de dependencias de módulos (MDG).

Combinando ambas medidas, se define la calidad de modularización del sistema, *MQ* (del inglés, *Modularization Quality*), que debe ser maximizada. El problema *SMC* puede verse como un problema de optimización monoobjetivo con función objetivo *MQ*.

Wiggerts [179] analizó la correspondencia entre el análisis *cluster* y la remodularización de los sistemas software, estableciendo una base teórica que relacionaba ambas áreas. Doval et al. [53] describieron una técnica eficiente para encontrar buenas particiones del grafo MDG. Para ello usaron un algoritmo genético. Mancoridis et al. [123] desarrollaron una herramienta llamada *Bunch* para mostrar el agrupamiento de módulos. Posteriormente comenzaron a desarrollarse algoritmos evolutivos que resolvían el problema *SMC* desde el punto vista multiobjetivo. En el año 2006, Praditwong y Yao [149] diseñaron un algoritmo denominado '*The Two-Archive Algorithm*' que mejoraba a los existentes, sobre todo para el caso de más de cuatro objetivos, dimensión a partir de la cual se empezaban a notar rendimientos bajos en comparación con los antiguos métodos multiobjetivo. Praditwong et al. [148] redefinieron la métrica *MQ* y trataron el problema *SMC* mediante un problema multiobjetivo con $p = 5$. Esta será la función a optimizar que consideraremos en el presente capítulo. Antes de ello, procedemos a definir una serie de conceptos.

Sea $G = (V, A, \omega)$ el grafo no dirigido ponderado que representa el MDG. V representa el conjunto de nodos y A el conjunto de aristas. La función de ponderación $\omega : A \longrightarrow \mathbb{Q}^+$ asigna un peso a cada arista del grafo. Para simplificar la notación escribiremos $\omega_{i,j} = \omega((i,j))$ con $i, j \in V$. Usaremos números naturales para denotar a los nodos. Consideramos que $\omega_{i,j} = 0$ cuando $(i,j) \notin A$. Denotamos por $n = |V|$ al número de nodos (módulos) del grafo y $m = |A|$ al número de aristas (relaciones entre módulos). Supondremos que el grafo G es no dirigido, ya que este hecho no afectará al cálculo de la cohesión y el acoplamiento de cada subsistema. En este caso, se cumple que $\omega_{ij} = \omega_{ji}$, $\forall i, j \in V$. Supondremos también que el grafo G es conexo.

Definición 6.1. Dado un MDG, sea $\{V_1, V_2, \ldots, V_l\}$ una partición del conjunto de nodos V. Definimos el *factor de modularización* del componente $k \in \{1, \ldots, l\}$ como

$$MF_k = \begin{cases} 0 & \text{si} \quad h_k = 0 \\ \frac{h_k}{h_k + \frac{1}{2} p_k} & \text{si} \quad h_k > 0, \end{cases} \tag{6.1}$$

donde h_k es la suma de los pesos de las aristas con los dos extremos en V_k (medida de cohesión) y p_k es la suma de los pesos de las aristas con exactamente un extremo en V_k (medida de acoplamiento).

UNIVERSIDAD DE MÁLAGA

La *calidad de modularización* de la partición se define como

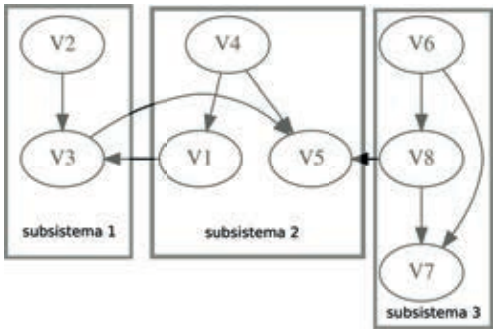
$$MQ = \sum_{k=1}^{l} MF_k. \tag{6.2}$$

En definitiva, estamos interesados en encontrar la partición que haga máximo el valor de MQ. El número de particiones en un conjunto de n elementos es un número de Bell (véase Sloane [161]). Para que nos hagamos una pequeña idea del enorme crecimiento que tiene esta sucesión, llamemos B_n al número de Bell para n elementos. Considerando $B_0 = 1$, es fácil ver que $B_1 = 1$ y $B_2 = 2$. Para $n > 1$, se tiene que $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$. Los primeros términos de esta sucesión son

$$1, 1, 2, 5, 15, 52, 203, 877, 4\,140, 21\,147, 115\,975, \ldots,$$

con lo que podemos apreciar que el crecimiento es más que exponencial. Para una mejor comprensión de cómo se calcula la calidad de modularización mostramos el siguiente ejemplo, extraído de Praditwong et al. [148].

Ejemplo 6.1. Dado el siguiente MDG compuesto por 8 nodos o módulos, consideramos la siguiente partición en tres subsistemas o *clusters*:



En este caso, tenemos que $MF_4 = MF_5 = MF_6 = MF_7 = MF_8 = 0$. Además,

$$MF_1 = \frac{1}{1+\frac{1}{2}\cdot 2} = \frac{1}{2}, \quad MF_2 = \frac{2}{2+\frac{1}{2}\cdot 3} = \frac{4}{7}, \quad MF_3 = \frac{3}{3+\frac{1}{2}\cdot 1} = \frac{6}{7},$$

por lo que $MQ = \frac{1}{2} + \frac{4}{7} + \frac{6}{7} = \frac{27}{14} = 1{,}928571\ldots$

6.2. Resolución del problema *SMC* mediante ramificación y poda

En este apartado diseñamos un algoritmo de ramificación y poda (Algoritmos 11 y 12) para resolver el problema *SMC* (véase Domínguez et al. [48]). El orden de exploración de los nodos y el cálculo de la cota superior para una solución incompleta son dos aspectos clave del algoritmo que influyen en su rendimiento. Para realizar una elección adecuada se ha hecho un estudio matemático del problema, que esbozamos a continuación.

Algoritmo 11 *SMC* por ramificación y poda ($n \geq 2$)

```
1: x = (⊥, ..., ⊥)
2: MQ ← 1
3: para k = 2 hasta n − ⌈MQ⌉ hacer
4:    para d = 0 hasta n − k hacer
5:       explorar (x, MQ, k, d)
6:    fin para
7: fin para
```

UNIVERSIDAD DE MÁLAGA

Algoritmo 12 explorar (x, MQ, k, d)

```
1: u ← calculaCotaSuperior()
2: si MQ < u entonces
3:   si x es solución completa entonces
4:     si f(x) > MQ entonces
5:       MQ = f(x)
6:     fin si
7:   si no
8:     i ← moduloParaAsignar()
9:     para l ∈ subsistemasFactibles() hacer
10:      x_i ← l
11:      explorar (x, MQ, k, d)
12:    fin para
13:    x_i ←⊥
14:  fin si
15: fin si
```

El factor de modularización MF_i siempre toma un valor entre 0 y 1. Su valor será 1 si y solo si el grupo está aislado del resto del grafo y 0 si y solo si el grupo contiene un conjunto independiente de nodos del grafo (no hay aristas entre los nodos del conjunto). En particular, si un grupo contiene solo un nodo, tendrá $MF_i = 0$. De esto se deduce que si consideramos una solución con k subsistemas no vacíos, el valor de MQ no puede ser mayor que k. Tampoco puede ser mayor que k si consideramos $n-k$ subsistemas no vacíos, ya que en este caso habrá como mucho k subsistemas con más de un nodo, y el resto tendrá $MF_i = 0$, por contener solo un nodo. Esto explica que usemos como criterio de parada para la exploración del número de subsistemas a alcanzar el valor $n - \lceil MQ \rceil$ en la Línea 3 del Algoritmo 11.

Ejemplo 6.2. Supongamos que analizando un grafo MDG con $n = 8$ nodos, disponemos de una partición inicial en subsistemas con $MQ = 3{,}4$. Según la Línea 3 del Algoritmo 11, la exploración debe realizarse desde $k = 2$ (el valor $k = 1$ se omite, pues en ese caso, $MQ = 1$) hasta $k = \lceil 3{,}4 \rceil = 4$. No tendría sentido analizar $k = 5$ subsistemas ya que, como mucho habría $8 - 5 = 3$ subsistemas con más de un nodo. Solo estos tres subsistemas podrían aportar valor a MF_i, ya que los restantes estarían formados por un único nodo, y tendríamos $MF_i = 0$. Por tanto, el valor máximo a obtener sería $MQ = 3$, lo cual no es óptimo.

Por otro lado, los resultados empíricos de la literatura (cómo en [149]) muestran que el valor de MQ suele ser por lo general bajo en comparación con el número de módulos, lo que sugiere que el número de subsistemas óptimo es bajo en comparación con n. Por ese motivo, realizamos la búsqueda explorando las soluciones con $k = 2, 3, \ldots$ subsistemas en orden creciente (si $k = 1$, $MQ = 1$). Una vez que k está fijo, las soluciones con valor de MQ más bajo son aquellas en las que hay una gran diferencia en la cardinalidad de los subsistemas (en particular, cuando un subsistema tiene solo un nodo, $MF_i = 0$). En el Algoritmo 11, el valor d denota la diferencia entre la cardinalidad de los grupos más grande y más pequeño. Las soluciones se exploran en orden creciente de d, de forma que subsistemas más equilibrados en cardinalidad son explorados antes. Una vez fijados k subsistemas no vacíos, todos tienen al menos un módulo, por lo que quedan $n-k$ módulos para desbalancear los grupos. Si todos se adjudican en uno de los k grupos, hay una diferencia de $n-k$ módulos entre el subsistema que tiene más nodos y el que tiene menos. Es por ello que el valor máximo para d es $n-k$ (Línea 4).

En relación a la función explorar, en la Línea 8 del Algoritmo 12 se selecciona un nodo que no esté asignado a ningún grupo. El criterio de selección de nodos puede ser muy diverso. En este caso se selecciona el que tiene mayor valor para la suma de los pesos de los ejes incidentes en él, con la idea de que esto proporcione un mayor valor para MQ. Seleccionado dicho nodo, a continuación se le asigna, uno a uno, todos los posibles subsistemas compatibles con la condición de que la solución tenga k subsistemas con diferencia de cardinalidad máxima d (Línea 9).

UNIVERSIDAD DE MÁLAGA

En cuanto a la cota superior (Línea 1), se calcula sumando las cotas superiores de MF_i para cada subsistema. Las cotas para MF_i se obtienen considerando como valor p_i (acoplamiento) la suma de los pesos de los ejes que, en la solución parcial, están asignados a distintos subsistemas, y como valor h_i (cohesión) la suma de los pesos de los ejes que forman parte del subsistema junto con la de todos los pesos de ejes que podrían formar parte del subsistema en alguna solución completa que respete la asignación realizada por la solución parcial. Cuando esta cota se calcula sobre una solución completa, coincide con MQ para dicha solución.

6.3. Resolución del problema *SMC* mediante programación matemática

En este apartado modelamos el problema *SMC* de manera que cualquier resolutor ILP sea capaz de resolver dicho problema de manera exacta. Supondremos que el MDG viene determinado por el grafo $G = (V, A, \omega)$, con n nodos y m aristas, siendo conexo dicho grafo. Sin pérdida de generalidad, podemos considerar que cada arista tiene asociado un coste entero positivo, expresado por la función ω. En primer lugar, definimos n^2 variables binarias de la siguiente manera:

$$x_{ik} = \begin{cases} 1 & \text{si el nodo } i \text{ pertenece al subsistema } k \\ 0 & \text{en otro caso.} \end{cases}$$

Si denotamos por h_k al valor de cohesión del subsistema k y por p_k al valor de acoplamiento de dicho subsistema, teniendo en cuenta que queremos maximizar el factor de modularización dado por la Ecuación (6.2), el modelo matemático a resolver podría formularse de la siguiente manera:

$$\text{máx} \sum_{i=1}^{n} \frac{h_i}{h_i + p_i/2}, \tag{6.3}$$

$$s.a. \sum_{k=1}^{n} x_{ik} = 1, \quad \forall i = 1, \ldots, n, \tag{6.4}$$

$$h_k = \sum_{(i,j)\in A} \omega_{ij} x_{ik} x_{jk}, \quad \forall k = 1, \ldots, n, \tag{6.5}$$

$$p_k = \sum_{(i,j)\in A} \omega_{ij}(x_{ik} + x_{jk}) - 2h_k, \quad \forall k = 1, \ldots, n, \tag{6.6}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i, k = 1, \ldots, n, \tag{6.7}$$

$$h_k, p_k \in \mathbb{Z}^+ \cup \{0\}. \tag{6.8}$$

La función objetivo de la Ecuación (6.3) maximiza la suma de los factores de modularización para cada subsistema, es decir, maximiza el valor MQ. La Ecuación (6.4) establece que cada nodo esté asignado a un único cluster. Las Ecuaciones (6.5) y (6.6) definen la cohesión y el acoplamiento de cada subsistema, respectivamente. Finalmente se define el tipo de cada una de las variables, que en este caso son binarias o enteras.

El modelo dado por las Ecuaciones (6.3)–(6.8) es claramente no lineal. La función objetivo es suma de fracciones lineales (ver definición en [95]). Además, posee restricciones de tipo cuadrático. Los resolutores ILP no son capaces de resolver este problema directamente.

Basándonos en la linealización de fracciones lineales propuesta por Charnes y Cooper [21], vamos a transformar la función objetivo de la Ecuación (6.3) en una función lineal. Definimos las variables $M_{ijk} = x_{ik} \cdot x_{jk}$, $\forall (i,j) \in A$ y $\forall k = 1, \ldots, n$. Estas nuevas mn variables serán también binarias y valdrán 1 si y solo si la arista (i,j) une dos nodos que están en el mismo subsistema, y 0 en otro caso. Para todo $k = 1, \ldots, n$ tal que $h_k + p_k \neq 0$, definimos $t_k = \frac{1}{h_k + p_k/2}$. La condición de que la suma entre la cohesión y el acoplamiento sea distinta de 0 es necesaria para que la variable t_k correspondiente esté bien definida. Estas variables servirán para eliminar los denominadores en la función objetivo. Definimos $\hat{h}_k = h_k t_k$ y $\hat{p}_k = p_k t_k$, $\forall k = 1, \ldots, n$. Estas $2n$ nuevas variables podríamos decir que convierten a continuas las

UNIVERSIDAD DE MÁLAGA

variables de cohesión y acoplamiento en el modelo matemático. Por último, y de manera análoga al caso anterior, definimos $\hat{x}_{ik} = x_{ik}t_k$, $\forall i, k = 1, \ldots, n$, y $\hat{M}_{ijk} = M_{ijk}t_k$, $\forall (i,j) \in A$, $\forall k = 1, \ldots, n$. Con estas nuevas definiciones y un poco de álgebra, el modelo que resuelve el *SMC* es equivalente al que se muestra en las Ecuaciones (6.9)–(6.18).

$$\text{máx} \sum_{i=1}^{n} \hat{h}_k, \tag{6.9}$$

$$s.a. \sum_{k=1}^{n} x_{ik} = 1, \quad \forall i = 1, \ldots, n, \tag{6.10}$$

$$\hat{h}_k = \sum_{(i,j)\in A} \omega_{ij}\hat{M}_{ijk}, \quad \forall k = 1, \ldots, n, \tag{6.11}$$

$$\hat{p}_k = \sum_{(i,j)\in A} \omega_{ij}(\hat{x}_{ik} + \hat{x}_{jk}) - 2\hat{h}_k, \quad \forall k = 1, \ldots, n, \tag{6.12}$$

$$\hat{h}_k + \frac{1}{2}\hat{p}_k = 1, \quad \forall k = 1, \ldots, n, \tag{6.13}$$

$$M_{ijk} = x_{ik}x_{jk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.14}$$

$$\hat{x}_{ik} = x_{ik}t_k, \quad \forall i, k = 1, \ldots, n, \tag{6.15}$$

$$\hat{M}_{ijk} = M_{ijk}t_k, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.16}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i, k = 1, \ldots, n, \tag{6.17}$$

$$\hat{x}_{ik}, M_{ijk}, \hat{M}_{ijk}, \hat{h}_k, \hat{p}_k, t_k \geq 0, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n. \tag{6.18}$$

Ahora la función objetivo es lineal y todas las variables son continuas o binarias. Tan solo restaría linealizar las restricciones cuadráticas de las Ecuaciones (6.14)–(6.16). Para ello, haremos uso de las dos siguientes proposiciones.

Proposición 6.1. La expresión $M = x_1x_2$ linealiza el producto de las variables binarias x_1 y x_2 si añadimos las restricciones

$$\begin{aligned} M &\leq x_1, \\ M &\leq x_2, \\ x_1 + x_2 &\leq 1 + M, \\ M &\geq 0. \end{aligned}$$

Demostración. Se comprueba fácilmente que aunque M no se defina como variable binaria, su valor queda limitado por el producto de las variables binarias, en función de los valores que éstas tomen. □

Proposición 6.2. La expresión $A = X \cdot T$, linealiza el producto de las variables $X \in \{0, 1\}$ y $T \in \mathbb{R}^+$, siendo U una cota superior de T, si añadimos las restricciones

$$\begin{aligned} A &\leq UX, \\ A &\geq T + U(X - 1), \\ A &\leq T - U(X - 1), \\ A &\geq 0. \end{aligned}$$

Demostración. Se demuestra fácilmente que el valor de A queda limitado por el producto de la variable binaria por la variable real no negativa siempre que conozcamos una cota superior de la misma. □

Utilizando las Proposiciones 6.1 y 6.2, podemos linealizar las Ecuaciones (6.14)–(6.16), y obtener el siguiente modelo lineal entero mixto:

UNIVERSIDAD DE MÁLAGA

$$\text{máx} \sum_{i=1}^{n} \hat{h}_k, \tag{6.19}$$

$$s.a. \sum_{k=1}^{n} x_{ik} = 1, \quad \forall i = 1, \ldots, n, \tag{6.20}$$

$$\hat{h}_k = \sum_{(i,j)\in A} \omega_{ij} \hat{M}_{ijk}, \quad \forall k = 1, \ldots, n, \tag{6.21}$$

$$\hat{p}_k = \sum_{(i,j)\in A} \omega_{ij} (\hat{x}_{ik} + \hat{x}_{jk}) - 2\hat{h}_k, \quad \forall k = 1, \ldots, n, \tag{6.22}$$

$$\hat{h}_k + \frac{1}{2}\hat{p}_k = 1, \quad \forall k = 1, \ldots, n, \tag{6.23}$$

$$M_{ijk} \leq x_{ik}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.24}$$

$$M_{ijk} \leq x_{jk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.25}$$

$$x_{ik} + x_{jk} \leq 1 + M_{ijk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.26}$$

$$\hat{x}_{ik} \leq U x_{ik}, \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, n, \tag{6.27}$$

$$\hat{x}_{ik} \geq t_k + U(x_{ik} - 1), \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, n, \tag{6.28}$$

$$\hat{x}_{ik} \leq t_k - U(x_{ik} - 1), \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, n, \tag{6.29}$$

$$\hat{M}_{ijk} \leq U M_{ijk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.30}$$

$$\hat{M}_{ijk} \geq t_k + U(M_{ijk} - 1), \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.31}$$

$$\hat{M}_{ijk} \leq t_k - U(M_{ijk} - 1), \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n, \tag{6.32}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i, k = 1, \ldots, n, \tag{6.33}$$

$$\hat{x}_{ik}, M_{ijk}, \hat{M}_{ijk}, \hat{h}_k, \hat{p}_k, t_k \geq 0, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, n. \tag{6.34}$$

Las Ecuaciones (6.19)–(6.34) linealizan completamente el problema *SMC* utilizando la definición de MQ dada en la Ecuación (6.2). Siguiendo el orden de los grupos de ecuaciones presentados en el modelo tenemos un total de $n+n+n+n+3mn+3n^2+3mn = 3n^2+6mn+4n$ restricciones. Siguiendo el orden que expresa el dominio de las variables en las Ecuaciones (6.33) y (6.34), tenemos n^2 variables binarias y $n^2+mn+mn+n+n+n = n^2+2mn+3n$ variables no negativas, es decir, un total de $2n^2 + 2mn + 3n$ variables.

Respecto al valor de U considerado en las Ecuaciones (6.27)–(6.32), éste debe ser una cota superior para las variables t_k. Si $h_k + p_k \neq 0$, es claro que $h_k + \frac{1}{2}p_k \geq \frac{1}{2}$ (los pesos de las aristas son enteros positivos), por lo que $t_k \leq \frac{1}{1/2} = 2$. Consideramos por tanto el valor $U = 2$. Obsérvese que en la Ecuación (6.23) se fuerza a que en cada subsistema haya al menos un nodo, es decir, estas restricciones harán que el resolutor siempre dé como solución un agrupamiento en n subsistemas, con un valor final de $MQ = 0$. A continuación restringimos el modelo para un número fijado de c subsistemas.

Algoritmo 13 *SMC* fijando el número de subsistemas ($n \geq 3$)

1: $MQ^* \leftarrow 1$
2: **para** $c = 2$ **hasta** $n - 1$ **hacer**
3: Resolver Q_c y obtener MQ
4: **si** $MQ^* < MQ$ **entonces**
5: $MQ^* = MQ$
6: **fin si**
7: **fin para**

UNIVERSIDAD DE MÁLAGA

Definición 6.2. Sea $c \in \{2, \ldots, n\}$. Llamamos Q_c al modelo matemático siguiente:

$$\text{máx} \sum_{k=1}^{c} \hat{h}_k, \tag{6.35}$$

$$s.a. \sum_{k=1}^{c} x_{ik} = 1, \quad \forall i = 1, \ldots, n, \tag{6.36}$$

$$\hat{h}_k = \sum_{(i,j)\in A} \omega_{ij} \hat{M}_{ijk}, \quad \forall k = 1, \ldots, c, \tag{6.37}$$

$$\hat{p}_k = \sum_{(i,j)\in A} \omega_{ij} (\hat{x}_{ik} + \hat{x}_{jk}) - 2\hat{h}_k, \quad \forall k = 1, \ldots, c, \tag{6.38}$$

$$\hat{h}_k + \frac{1}{2}\hat{p}_k = 1, \quad \forall k = 1, \ldots, c, \tag{6.39}$$

$$M_{ijk} \leq x_{ik}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, c, \tag{6.40}$$

$$M_{ijk} \leq x_{jk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, c, \tag{6.41}$$

$$x_{ik} + x_{jk} \leq 1 + M_{ijk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, c, \tag{6.42}$$

$$\hat{x}_{ik} \leq U x_{ik}, \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, c, \tag{6.43}$$

$$t_k + U(x_{ik} - 1) \leq \hat{x}_{ik} \leq t_k - U(x_{ik} - 1), \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, c, \tag{6.44}$$

$$\hat{M}_{ijk} \leq U M_{ijk}, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, c, \tag{6.45}$$

$$t_k + U(M_{ijk} - 1) \leq \hat{M}_{ijk} \leq t_k - U(M_{ijk} - 1), \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, c, \tag{6.46}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, c, \tag{6.47}$$

$$\hat{x}_{ik}, M_{ijk}, \hat{M}_{ijk}, \hat{h}_k, \hat{p}_k, t_k \geq 0, \quad \forall (i,j) \in A, \ \forall k = 1, \ldots, c. \tag{6.48}$$

Obsérvese que el modelo dado por las Ecuaciones (6.35)–(6.48) resuelve el problema *SMC* para un número fijado, c, de subsistemas, y que las Ecuaciones (6.19)–(6.34) representan a Q_n como caso particular. Si resolvemos el *SMC* fijando el número de subsistemas, y este lo variamos desde 2 hasta $n-1$ (obviando los casos triviales $c = 1$ y $c = n$, que proporcionan $MQ = 1$ y $MQ = 0$, respectivamente), disponemos ya de un método que resuelve de manera exacta dicho problema. Puede verse en el Algoritmo 13.

El Algoritmo 13 tiene el inconveniente de tener que realizar $n-2$ llamadas al resolutor, lo cual podría ser muy costoso cuando el número de nodos en el MDG es muy elevado. Presentamos a continuación un nuevo programa matemático que resuelve el problema en una única llamada al resolutor.

Definición 6.3. Llamamos Q_{rel} (modelo relajado) al modelo Q_n sustituyendo el grupo de Ecuaciones (6.39) por

$$\hat{h}_k + \frac{1}{2}\hat{p}_k \leq 1, \quad \forall k = 1, \ldots, n. \tag{6.49}$$

Vamos a demostrar que resolver el problema Q_{rel} de manera óptima es equivalente a resolver el problema *SMC*.

Proposición 6.3. En Q_{rel} una solución factible viene determinada por los valores $\{x_{ik}\}_{i,k=1}^{n}$ y por $\{t_k\}_{k=1}^{n}$, siempre que se verifiquen las restricciones dadas por (6.36) y (6.49), y además se cumpla que $t_k \leq U$, $\forall k$. Esto quiere decir que el resto de variables ya viene determinado a partir de éstas y que además se verifican el resto de restricciones del modelo.

Demostración. En primer lugar observemos que las ecuaciones (6.40)–(6.42) fuerzan a que las variables M_{ijk} sean binarias (véase Proposición 6.1). Si $x_{ik} = 1$, por la Ecuación (6.44), se cumple que $\hat{x}_{ik} = t_k$. Si $x_{ik} = 0$, por la Ecuación (6.43), se cumple que $\hat{x}_{ik} = 0$. Por tanto, las variables $\hat{x}_{ik} \in \{0, t_k\}$ son dicotómicas. Análogamente, si $M_{ijk} = 0$, por la Ecuación (6.45) tenemos que $\hat{M}_{ijk} = 0$ y por la Ecuación (6.46) tenemos que $t_k - U \leq \hat{M}_{ijk} = 0$, luego $t_k \leq U$. Por otro lado, si $M_{ijk} = 1$, por la Ecuación (6.46) tenemos que $\hat{M}_{ijk} = t_k$ y por la Ecuación (6.45) tenemos que $t_k \leq U$. Obtenemos que las variables $\hat{M}_{ijk} \in \{0, t_k\}$ son también dicotómicas, y además el valor t_k está acotado por U, por lo que el resolutor nunca podrá proporcionar un valor tendiendo a infinito para esta variable, aún cumpliéndose que

UNIVERSIDAD DE MÁLAGA

$\hat{h}_k = \hat{p}_k = 0$. Las variables $\hat{h}_k$ y $\hat{p}_k$ vienen determinadas en las Ecuaciones (6.37) y (6.38), y no aparecen en ninguna restricción más salvo en el grupo (6.49). En conclusión, si además se cumplen las restricciones de (6.36) y de (6.49), dispondremos de una solución factible para el modelo. □

Corolario 6.1. Dado $c \in \{2, \ldots, n\}$, tenemos que en Q_c una solución factible viene determinada por los valores $\{x_{ik}\}_{k=1}^{c}$, $\forall i = 1, \ldots, n$, y por $\{t_k\}_{k=1}^{c}$, siempre que se verifiquen las restricciones dadas por (6.36) y (6.39).

Demostración. Obsérvese que en este caso el valor de t_k también viene determinado por el resolutor, debido a que la Ecuación (6.39) contiene restricciones de igualdad. Por ello no se exige en este caso la condición $t_k \leq U$, $\forall k$. □

Teorema 6.1. Cualquier solución factible para el problema Q_c admite una solución equivalente para el problema Q_{rel}, en ambos casos con el mismo valor objetivo.

Demostración. Sea $c \in \{2, \ldots, n\}$. Sea (x, t) una solución factible para Q_c, siendo $x = (x_{11}, \ldots, x_{1c}, \ldots, x_{n1}, \ldots, x_{nc})$ y $t = (t_1, \ldots, t_c)$. Vamos a construir una solución factible (x', t') en Q_{rel}. Definimos

$$x'_{ik} = \begin{cases} x_{ik} & \text{para todo} \quad i = 1, \ldots, n; k = 1, \ldots, c \\ 0 & \text{para todo} \quad i = 1, \ldots, n; k = c+1, \ldots, n, \end{cases}$$

y definimos

$$t'_k = \begin{cases} t_k & \text{para todo} \quad k = 1, \ldots, c \\ 0 & \text{para todo} \quad k = c+1, \ldots, n. \end{cases}$$

Claramente se verifican las ecuaciones $\sum_{k=1}^{n} x'_{ik} = 1$, $\forall i = 1, \ldots, n$, luego se cumple (6.36). Para los primeros c subsistemas es claro que $\hat{h}_k + \hat{p}_k/2 = 1 \leq 1$. Para los restantes subsistemas, se comprueba fácilmente que $\hat{x}_{ik} = 0$ para todo $i = 1, \ldots, n; k = c+1, \ldots, n$; y $\hat{M}_{ijk} = 0$ para todo $(i, j) \in A; k = c+1, \ldots, n$. Esto implica que $\hat{h}_k = \hat{p}_k = 0$, y en consecuencia $\hat{h}_k + \hat{p}_k/2 = 0 \leq 1$. Luego se verifica (6.49). El valor objetivo de la solución (x', t') es claramente el mismo que el que proporciona la solución (x, t), ya que este depende de los valores de $\hat{M}_{ijk}$, que son iguales para los c primeros subsistemas. Para los restantes subsistemas, $\hat{M}_{ijk} = 0$ y no aportan ningún valor añadido a la función objetivo. □

Supongamos que disponemos de una solución factible en Q_{rel}. Realizamos una partición de los índices de los subsistemas mediante cuatro conjuntos que definimos a continuación:

$$\begin{aligned} \mathfrak{I}_1 &= \{k \mid t_k = 0 \text{ y existe } i \text{ tal que } x_{ik} = 1\}, \\ \mathfrak{I}_2 &= \{k \mid t_k = 0 \text{ y } x_{ik} = 0, \ \forall i = 1, \ldots, n\}, \\ \mathfrak{I}_3 &= \{k \mid t_k > 0 \text{ y } \hat{h}_k + \hat{p}_k/2 = 0\}, \\ \mathfrak{I}_4 &= \{k \mid t_k > 0 \text{ y } \hat{h}_k + \hat{p}_k/2 > 0\}. \end{aligned}$$

Es evidente que los cuatro conjuntos definidos son disjuntos dos a dos, por lo que no tienen elementos en común. Además, cualquier índice de un subsistema pertenece a alguno de los cuatro subconjuntos.

En el próximo teorema vamos a determinar que si la solución de Q_{rel} es óptima, se puede definir de manera constructiva una solución en Q_c con el mismo valor objetivo, tomando nodos cuyo subsistema al que pertenece sea un elemento en $\mathfrak{I}_1$, y por otro lado subsistemas de nodos que estén en $\mathfrak{I}_4$, justificando que no será necesario usar los índices de los subsistemas que están en $\mathfrak{I}_2 \cup \mathfrak{I}_3$. Llamamos $\Gamma(i)$ al conjunto de nodos adyacentes al nodo i en el grafo MDG. Sabemos que este conjunto es no vacío, pues el grafo es conexo.

El conjunto $\mathfrak{I}_1$ muestra los índices de subsistemas cuyos nodos estarán aislados en la nueva solución que construiremos. Sea i un nodo que en la solución está en el subsistema k, siendo $k \in \mathfrak{I}_1$. Dado $j \in \Gamma(i)$, las variables M_{ijk} son binarias. Si $M_{ijk} = 0$, por la Ecuación (6.45) obtenemos que $\hat{M}_{ijk} = 0$. Si $M_{ijk} = 1$, por la Ecuación (6.46) obtenemos que $\hat{M}_{ijk} = 0$, ya que $t_k = 0$. Concluimos que para cualquier nodo i que esté en el subsistema k, tenemos

UNIVERSIDAD DE MÁLAGA

que $\hat{M}_{ijk} = 0$. Obviamente, si un nodo i' no está en el subsistema k también verificará $M_{i'jk} = \hat{M}_{i'jk} = 0,\ \forall j \in \Gamma(i)$. De la Ecuación (6.37) se deduce que $\hat{h}_k = 0$, por lo que los nodos del subsistema k no aportan valor a la función objetivo, lo que quiere decir que en la nueva solución podemos considerar a dichos nodos como aislados.

El conjunto $\mathfrak{I}_2$ contiene índices de subsistemas que no contienen nodos, como puede deducirse fácilmente de su propia definición.

El conjunto $\mathfrak{I}_3$ también contiene índices de subsistemas que no contienen nodos, como veremos a continuación. Sea k un subsistema de este conjunto. Como $\hat{h}_k + \hat{p}_k/2 = 0$, tenemos que $\hat{h}_k = \hat{p}_k = 0$, y por las Ecuaciones (6.37) y (6.38), se cumple que $\hat{x}_{ik} + \hat{x}_{jk} = 0,\ \forall (i,j) \in A$. Supongamos por reducción al absurdo que el subsistema k contiene algún nodo, es decir, existe i tal que $x_{ik} = 1$. Entonces por la Ecuación (6.44) obtenemos que $\hat{x}_{ik} = t_k > 0$. Al ser conexo el grafo MDG, sabemos que existe j tal que $(i,j) \in A$. Esto quiere decir que $\hat{x}_{ik} + \hat{x}_{jk} > 0$, lo cual es una contradicción.

El conjunto $\mathfrak{I}_4$ contiene los restantes casos, es decir, los índices de los subsistemas no vacíos.

Teorema 6.2. Cualquier solución óptima en Q_{rel} admite una solución equivalente factible en Q_c para cierto c.

Demostración. Partimos de una solución óptima (x,t) en Q_{rel}. Vamos a construir una solución para un problema Q_c, para cierto c. Dividimos el proceso en dos partes. En primer lugar construimos b subsistemas aislados a partir de nodos que están en un subsistema cuyo índice pertenece a $\mathfrak{I}_1$, y en la segunda parte creamos el resto de subsistemas, numerados desde $b+1$ hasta c, a partir de los subsistemas que estén en $\mathfrak{I}_4$.

Para cada nodo i que verifique $x_{ik} = 1$ para cierto $k \in \mathfrak{I}_1$, formaremos un subsistema aislado en la nueva solución. Denotamos por K el subsistema actual de la nueva solución. Inicialmente $K = 1$. Sea i el primer nodo que verifica $x_{ik} = 1$ para $k \in \mathfrak{I}_1$. Definimos los siguientes valores

$$x'_{iK} = 1; \quad x'_{jK} = 0,\ \forall j = 1,\dots,n, j \neq i,$$
$$t'_K = \frac{2}{\sum_{j\in\Gamma(i)} \omega_{ij}}.$$

Repetimos el proceso anterior para cada nodo i que verifique $x_{ik} = 1$ con $k \in \mathfrak{I}_1$, de manera que K va tomando los valores desde 2 hasta un cierto b. Finalizada esta primera parte tenemos definidos b subsistemas que contienen un único nodo aislado en la nueva solución.

En la segunda parte de la construcción tomamos cada índice $k \in \mathfrak{I}_4$ y vamos creando nuevos subsistemas que comenzarán desde $K = b+1$ hasta un cierto $K = c$. En este caso, los valores de las variables en la nueva solución serán los mismos que en la solución original. Fijado el primer valor de $k \in \mathfrak{I}_4$, definimos:

$$x'_{iK} = x_{ik}, \quad \forall i = 1,\dots,n,$$
$$t'_K = t_k.$$

Repetimos el proceso anterior para cada $k \in \mathfrak{I}_4$ incrementando de uno en uno el valor de K. Al finalizar, dispondremos de una solución con c subsistemas. Veamos que dicha solución es factible en Q_c.

Necesitamos probar, en base al Corolario 6.1, que se cumple $t'_K \leq U,\ \forall K$ y que se verifican las Ecuaciones (6.36) y (6.39).

Para $K \in \{1,\dots,b\}$, tenemos que $t'_K = \frac{2}{\sum_{j\in\Gamma(i)} \omega_{ij}} \leq 2 = U$, ya que el denominador es mayor o igual que uno (se consideran enteros positivos todos los pesos de las aristas). Para $K \in \{b+1,\dots,c\}$, se verifica $t'_K = t_k \leq U$. Por tanto, $t'_K \leq U,\ \forall K$.

Por otro lado, tenemos que $\sum_{i=1}^{n} x'_{iK} = 1,\ \forall K = 1,\dots,b$, ya que, según la definición, solo uno de los sumandos vale uno, y el resto vale cero. Además, $\sum_{i=1}^{n} x'_{iK} = \sum_{i=1}^{n} x_{ik} = 1,\ \forall K = b+1,\dots,c$, por lo que se cumplen las restricciones dadas por (6.36).

Por último, para $K \in \{1,\dots,b\}$ tenemos que $\hat{h}'_K = 0$, por lo que las Ecuaciones (6.38) se transforman en $\hat{p}'_K = \sum_{(i,j)\in A} \omega_{ij}(\hat{x}'_{iK} + \hat{x}'_{jK})$. De todos esos sumandos, y teniendo en cuenta que los primeros b subsistemas están formados por nodos aislados, solo serán positivos

UNIVERSIDAD DE MÁLAGA

los sumandos que contengan el nodo i, por lo que habrá $|\Gamma(i)|$ sumandos positivos, y el resto valdrá cero. Así,

$$\hat{p}'_K = \omega_{ij_1}\hat{x}'_{iK} + \omega_{ij_2}\hat{x}'_{iK} + \ldots + \omega_{ij_{|\Gamma(i)|}}\hat{x}'_{iK} = \sum_{j\in\Gamma(i)} \omega_{ij}\cdot \hat{x}'_{iK} = \sum_{j\in\Gamma(i)} \omega_{ij}\cdot \frac{2}{\sum_{j\in\Gamma(i)}\omega_{ij}} = 2.$$

Esto implica que $\hat{h}'_K + \hat{p}'_K/2 = 0 + 2/2 = 1$.

Faltaría probar que $\hat{h}'_K + \hat{p}'_K/2 = 1$, $\forall K \in \{b+1, \ldots, c\}$. Recordamos que estos subsistemas fueron extraídos del conjunto $\mathfrak{I}_4$, por lo que $0 < \hat{h}'_K + \hat{p}'_K/2 \leq 1$. Vamos a obtener el mínimo valor posible que puede tomar $\hat{h}'_K + \hat{p}'_K/2$, sabiendo que es positivo. Si nos fijamos en la definición de $\hat{h}'_K$ en la Ecuación (6.37) y observando que $\omega_{ij} \in \mathbb{Z}^+$, $\forall (i,j) \in A$, y que $\hat{M}'_{ijK} \in \{0, t'_K\}$, deducimos que el mínimo valor posible que puede tomar $\hat{h}'_K$ es un múltiplo entero de t'_K. De hecho, este mínimo valor positivo será mín$\{\omega_{ij}\}$[1] $\cdot\, t'_K$. Lo mismo podemos razonar en base a la definición de $\hat{p}'_K$ en la Ecuación (6.38). El valor de $\hat{p}'_K$ tiene una serie de sumandos positivos con todos los términos un múltiplo de t'_K, ya que también se verifica que $\hat{x}'_{iK} \in \{0, t'_K\}$, y una serie de sumandos negativos (los que vienen definidos por $-2\hat{h}_K$), que también son múltiplos enteros de t'_K. En cualquier caso, el mínimo incremento positivo que toma $\hat{p}'_K$ ha de ser mín$\{\omega_{ij}\} \cdot t'_K$. Como $\hat{h}'_K + \hat{p}'_K/2 > 0$, y no pueden ser los dos términos iguales a 0, el mínimo valor positivo que tome esta expresión será $1/2\,\text{mín}\{\omega_{ij}\} \cdot t'_K$. Así que tenemos que

$$1/2\,\text{mín}\{\omega_{ij}\} \cdot t'_K \leq \hat{h}'_K + \hat{p}'_K/2 \leq 1. \tag{6.50}$$

Supongamos por reducción al absurdo que $\hat{h}'_K + \hat{p}'_K/2 < 1$. Utilizando la Ecuación (6.50) podemos deducir que

$$\begin{aligned} 1/2\,\text{mín}\{\omega_{ij}\} \cdot t'_K \leq \hat{h}'_K + \hat{p}'_K/2 < 1,\\ \frac{2}{\text{mín}\{\omega_{ij}\} \cdot t'_K} \geq \frac{1}{\hat{h}'_K + \hat{p}'_K/2} > 1,\\ \frac{2}{\text{mín}\{\omega_{ij}\}} \geq \frac{t'_K}{\hat{h}'_K + \hat{p}'_K/2} > t'_K. \end{aligned}$$

Sea $t^*_K = t'_K/(\hat{h}'_K + \hat{p}'_K/2)$. Dado que el denominador es estrictamente menor que uno, se cumple que $t^*_K > t'_K$. Consideremos la solución (x', t^*). Veamos que esta solución es factible:

$$t^*_K = \frac{t'_K}{\hat{h}'_K + \hat{p}'_K/2} \leq \frac{2}{\text{mín}\{\omega_{ij}\}} \leq \frac{2}{1} = U \Longrightarrow t^*_K \leq U. \tag{6.51}$$

Las Ecuaciones (6.36) y (6.49) se cumplen claramente, así que (x', t^*) es factible en Q_{rel}. Sin embargo, el valor objetivo es mayor que el valor objetivo de la solución (x', t'), ya que el objetivo se obtiene como suma de valores no negativos que dependen de $\hat{M}_{ijK}$. Para (x', t') este valor será una suma de ciertos valores en el conjunto $\{0, t'_K\}$. Como $t^*_K > t'_K$, el objetivo estará formado por suma de valores que son mayores. Por tanto, esto contradice que (x', t') es óptimo.

En conclusión, debe cumplirse forzosamente que $\hat{h}'_K + \hat{p}'_K/2 = 1$, $\forall K = b+1, \ldots, c$, y por tanto se verifica la Ecuación (6.39) y (x', t) es solución factible para Q_c.

Cuando decimos que la solución construida en Q_c es equivalente a la solución en Q_{rel} quiere decir que el valor objetivo es el mismo en ambos casos. Esto se puede comprobar fácilmente por propia construcción. Los b primeros subsistemas están compuestos por nodos aislados, por lo que no aportan valor objetivo en Q_c y tampoco lo aportaban en Q_{rel}. Los restantes subsistemas son los que aportan valor a la función objetivo, y dado que $t'_K = t_k$ y que $x'_{iK} = x_{ik}$, entonces tendremos que $M'_{ijK} = M_{ijk}$ y el valor objetivo que proporcionan los subsistemas $K = b+1, \ldots, c$, será el mismo.

□

Teorema 6.3. La solución óptima del modelo Q_{rel} es la solución óptima para el *SMC*.

[1] Abusando de la notación, escribimos mín$\{\omega_{ij}\}$, en lugar de mín$_{(i,j)\in A}\{\omega_{ij}\}$.

UNIVERSIDAD DE MÁLAGA

Demostración. Supongamos que (x, t) es la solución óptima para Q_{rel}. Aplicando el Teorema 6.2, obtenemos una solución equivalente (x', t') en $Q_{c'}$ para cierto c'. Si esta no fuese la solución óptima para el *SMC*, entonces existe (x'', t'') en $Q_{c''}$ (donde podría ocurrir $c'' = c'$) que tiene un valor objetivo mayor. Aplicando el Teorema 6.1, obtendríamos una solución equivalente (x''', t''') en Q_{rel} con valor objetivo mayor que el que proporciona (x, t), lo cual contradice que era óptimo. □

El Teorema 6.3 demuestra que el problema *SMC* puede resolverse con una única llamada al resolutor, sin más que resolver el modelo Q_{rel}. La relajación producida por la Ecuación (6.39), en la que las igualdades se transforman en desigualdades del tipo menor o igual, permiten al resolutor trabajar indistintamente con cualquier número de subsistemas, garantizándose que su óptimo saturará las restricciones en (6.49) para aquellos subsistemas que contienen más de un nodo, y que además dicha solución será óptima.

6.4. Resultados computacionales

Consideramos 9 instancias para el problema *SMC*. Seis de ellas son extraídas de la literatura. Las instancias utilizan el grafo MDG como dirigido. En este capítulo tomamos el MDG como un grafo no dirigido (pues la definición que hemos dado a *MQ* no lo requiere). En algunas de las instancias existen arcos (i, j) y sus correspondientes arcos (j, i). En estos casos realizaremos dos distinciones: en primer lugar consideramos simplemente la arista (i, j) en el grafo no dirigido y omitimos la arista (j, i), ya que es la misma. En segundo lugar, consideramos los dos arcos duplicando el coste de la arista (i, j). Estas dos distinciones de una misma instancia serán consideradas como dos instancias distintas en nuestros experimentos. En el primer caso la instancia recibe el nombre original y en el segundo caso nombraremos a la instancia con el mismo nombre, pero seguido del sufijo '2'. Todas las instancias pueden descargarse desde `https://github.com/MiguelAngelDominguezRios/Software-Module-Clustering/tree/main/Instances` y su resumen puede verse en la Tabla 6.1.

Tabla 6.1: Instancias consideradas para el problema *SMC*. La columna *Links* determina el número de relaciones entre módulos considerando el grafo MDG dirigido.

Nombre instancia	n	m	*Links*
mtunis	20	57	57
ispell	24	97	103
ispell2	24	97	103
rcs	29	155	163
rcs2	29	155	163
bison	37	179	179
grappa	86	252	295
grappa2	86	252	295
incl	174	360	360

Las máquinas utilizadas para computar los experimentos proceden de una *multicore* con cuatro CPUs Intel Xeon (E5-2670 v3) a 3.1 GHz, un total de 48 hilos de ejecución, y sistema operativo Ubuntu 16.04 LTS. Para cada ejecución se usa un único hilo de proceso y 8 GB de RAM.

En relación al valor U, cota superior para t_k considerado en las Ecuaciones (6.27)–(6.32), experimentos preliminares sugieren que existe una influencia importante en el rendimiento en función del valor de esta cota. Si está muy ajustada o muy alejada a los valores que puede tomar t_k, el rendimiento parece ser menor. Este hecho podría tener un componente aleatorio e interno a CPLEX. Consideramos que dejando cierto margen de maniobra al resolutor, estableciendo una cota que no sea ni muy ajustada ni muy alejada de los valores que toma t_k, la exploración del poliedro podría resultar más eficiente. Hemos multiplicado por 2 el valor del coste de cada arista en cada instancia. Así, dada la definición de *MQ*, su valor final será el mismo. Sin embargo, al multiplicar el coste asociado a cada arista por 2, podemos garantizar que si $h_k + p_k \neq 0$, entonces $h_k + \frac{1}{2}p_k \geq 1$, por lo que $t_i \leq \frac{1}{1} = 1$. Por tanto, podemos

UNIVERSIDAD DE MÁLAGA

considerar en este caso el valor $U = 1$. No obstante, hemos mantenido $U = 2$, dejando así un margen de maniobra para el resolutor, como mencionamos anteriormente.

Un parámetro importante a considerar cuando usamos el resolutor CPLEX es el valor del parámetro CPX_PARAM_MIPEMPHASIS. Este marcador controla el balance entre la velocidad, factibilidad, optimalidad y el movimiento en función de las cotas en la resolución del problema entero. Cuando el valor es 0 (CPX_MIPEMPHASIS_BALANCED), CPLEX se centra en una rápida búsqueda de la solución óptima, realizando un esfuerzo superior inicial en encontrar soluciones factibles. Este esfuerzo puede ser modificado. Así, si el valor se asigna a 1 (CPX_MIPEMPHASIS_FEASIBILITY) se generan más soluciones factibles en detrimento de una rápida prueba de la optimalidad. Cuando vale 2 (CPX_MIPEMPHASIS_OPTIMALITY) el esfuerzo en encontrar soluciones factibles es menor, y se centra en la búsqueda directa de la optimalidad. Por defecto, CPLEX usa la opción 0, ya que balancea las dos opciones anteriores. Si se establece el valor del parámetro en 3 (CPX_MIPEMPHASIS_BESTBOUND), se explora el árbol de búsqueda mediante aquellos nodos con mejor valor de cota, en detrimento de la obtención de soluciones factibles. Versiones posteriores a CPLEX 12.6.2., que es la que usamos en estos experimentos, ofrecen nuevas opciones para este parámetro.

Se ejecuta cada instancia una única vez durante los siguientes periodos: $\frac{1}{2}$ hora, 1 hora, 2 horas, 4 horas y 8 horas. Variamos además el valor de CPX_PARAM_MIPEMPHASIS en el conjunto $\{0, 1, 2, 3\}$. Dada la magnitud de las instancias consideradas, no solo ninguna de ellas consigue alcanzar el óptimo dentro de las 8 horas de ejecución, sino que existe un desbordamiento de memoria en algunos casos, y que depende de los valores del parámetro CPX_PARAM_MIPEMPHASIS. Los momentos en los que los 8 GB de memoria no han sido suficientes para abordar el problema se muestran en la Tabla 6.2.

Tabla 6.2: Cota superior del tiempo en el cual el resolutor no ha sido capaz de continuar la ejecución hasta las 8 horas debido a un desbordamiento de memoria al considerar 8 GB de RAM para las distintas instancias, en función de los valores del parámetro CPX_PARAM_MIPEMPHASIS.

Nombre instancia	CPX_PARAM_MIPEMPHASIS			
	0	1	2	3
mtunis	2 h		4 h	
ispell	4 h		4 h	
ispell2				
rcs		4 h		
rcs2	4 h	4 h		
bison		4 h		
grappa		8 h		
grappa2				
incl				

Dado que para el caso CPX_PARAM_MIPEMPHASIS = 3 no se ha producido desbordamiento de memoria en ninguna instancia, ésta será la opción utilizada para los experimentos. Ejecutamos nuevamente las instancias en distintos intervalos de tiempo: 16, 24, 32, 40 y 48 horas. Se consideran distintos tramos para controlar si se produjese algún desbordamiento de memoria intermedio.

Para conseguir un mejor rendimiento incluimos restricciones adicionales al problema, de manera que podamos reducir algunas simetrías. Por ejemplo, podemos considerar que el primer nodo siempre pertenezca al primer subsistema; el segundo nodo podrá estar en el primer subsistema o en el segundo; el tercer nodo podrá estar en el primero, segundo o tercer subsistema, y así sucesivamente. Estas restricciones se pueden condensar añadiendo el siguiente grupo de ecuaciones:

$$\sum_{k=1}^{i} x_{ik} = 1, \quad \forall i = 1, \ldots, n-1. \tag{6.52}$$

Se ha comprobado preliminarmente la considerable mejora en el rendimiento al añadir estas restricciones al modelo.

UNIVERSIDAD DE MÁLAGA

Se ejecutan los experimentos para el algoritmo que usa programación matemática, así como para el que utiliza ramificación y poda (ver Sección 6.2). Ninguna de las instancias ha conseguido ser resuelta de manera exacta en el tiempo límite establecido de 48 horas. Mostramos los resultados parciales en la Tabla 6.3, así como el *gap*[2] existente para el algoritmo que usa programación matemática, teniendo en cuenta que la solución será óptima cuando dicho *gap* valga 0 (a efectos prácticos el resolutor finaliza la ejecución para cuando el límite de tolerancia llega a CPX_PARAM_EPGAP, que por defecto es 10^{-5}). En los casos en los que ha existido un desbordamiento de memoria por parte del resolutor, y no se han podido considerar las 48 horas de ejecución, el mismo tiempo límite se ha usado para el algoritmo de ramificación y poda.

Tabla 6.3: Resultados computacionales para el problema *SMC* tras un máximo de 48 horas de ejecución utilizando el algoritmo que usa programación matemática y el que usa ramificación y poda (*BB*).

Nombre instancia	*gap*	*MQ*	*MQ (BB)*	Tiempo
mtunis	2.49280	2.2466512956	2.3144617531	24 h
ispell	3.09583	2.3632954602	2.2020565493	48 h
ispell2	3.40422	2.1685886009	2.1419319243	48 h
rcs	6.04115	2.2579893990	1.8067892373	48 h
rcs2	6.35946	2.1638967065	1.8041958042	48 h
bison	4.57628	3.0556762826	1.7901692615	48 h
grappa	2.68769	12.3549657563	1.8298731257	48 h
grappa2	3.27449	10.6002318042	1.8551307847	48 h
incl	10.90365	7.1412096519	0.8178644849	48 h

En la Tabla 6.3 puede verse claramente como en la mayoría de los casos el algoritmo que usa programación matemática tiene un valor para *MQ* mucho mayor que en el caso del algoritmo de ramificación y poda. Esta diferencia es tanto mayor cuanto mayor es el número de nodos que tiene el MDG de la instancia. Obsérvese la enorme variación que hay en las instancias grappa y grappa2 si comparamos los dos algoritmos. A continuación, nos centramos en la instancia mtunis, de la que conocemos una solución con *MQ* mejor para el algoritmo de ramificación y poda. Observamos la frecuencia en la que el resolutor va encontrando soluciones factibles en función del tiempo, indicando el valor de *MQ* asociado a cada solución factible, así como su valor de *gap* en función del tiempo. Los resultados pueden verse en la Figura 6.2. Podemos ver que si bien durante los primeros minutos de ejecución el resolutor encuentra varias soluciones factibles, encontrar una nueva puede tomar varias horas de ejecución, aunque no necesariamente cada nueva solución tarda más tiempo en encontrarse que la anterior. Esto dependerá del problema en sí mismo y en como CPLEX va internamente recorriendo los vértices del poliedro. En relación al *gap*, vemos que su decrecimiento es muy lento, por lo que se presume que se necesitaría aún bastante tiempo más de computación para garantizar que hemos encontrado el óptimo.

6.5. Investigación ampliada

Aún siendo interesante modelar el problema *SMC* mediante programación lineal entera mixta, los resultados obtenidos en esta propuesta quedan aún lejos de ser competitivos, ya que existen metaheurísticas que consiguen mejores resultados computacionales (véase [148]). Por otro lado, los valores obtenidos para el *gap* no nos permiten garantizar si estamos aún cerca del óptimo, aunque se podría estimar mediante regresión no lineal el tiempo restante estimado de ejecución. Por ello, el uso de esta propuesta resulta actualmente poco útil a efectos prácticos, más allá del interés matemático que pudiese tener. A fin de intentar explotar las características de este problema, siempre dentro del ámbito de la programación matemática, se ha realizado una investigación profunda del *SMC* cuyo resumen se muestra a continuación, si bien hasta el momento no hemos encontrado una versión del algoritmo que se considere mejor que la expuesta en esta sección.

[2]Tolerancia relativa en entre el mejor valor objetivo y el objetivo del mejor nodo de los restantes a explorar en el árbol de búsqueda en el *Branch & Cut*.

UNIVERSIDAD DE MÁLAGA

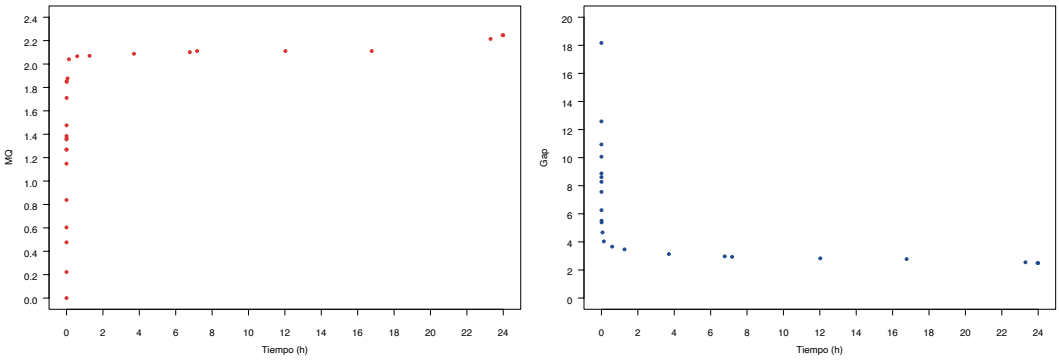


FIGURA 6.2: Valores de *MQ* y del *gap* para la instancia mtunis, en función del tiempo. Cada punto representa la obtención de una solución factible por parte del resolutor CPLEX.

Nuevas restricciones para evitar simetrías

Sustituimos las restricciones de simetría consideradas en la Ecuación (6.52) por otras nuevas, basándonos en la propuesta de la Sección 6.2 en la que fijamos el valor de la diferencia entre los subsistemas con mayor y menor número de nodos. Si añadimos las restricciones

$$\sum_{i=1}^{n} x_{ik} \geq \sum_{i=1}^{n} x_{i(k+1)}, \quad \forall k = 1, \ldots, n-1, \tag{6.53}$$

obligamos a que los subsistemas estén ordenados en orden no creciente en cuanto al número de módulos, lo que permite reducir simetrías. Si además añadimos la restricción

$$\sum_{i=1}^{n} x_{i1} - \sum_{i=1}^{n} x_{in} = \mathit{diff}, \tag{6.54}$$

podemos fijar el valor de la diferencia (*diff*) entre el subsistema con mayor y menor número de módulos y resolver el problema para cada valor fijado de *diff*. Otra opción es usar esta variable diferencia penalizando la función objetivo, y forzando así a que el resolutor busque subsistemas con un número de módulos lo más equilibrado posible. En ambos casos, el rendimiento dado por CPLEX ha resultado ser menos eficiente.

Utilizar ramificación y poda fijando nodos en un mismo subsistema

Se propone una nueva forma de resolver el problema *SMC* resolviendo subproblemas del problema original con un número menor de nodos. Para ello, se parte de la idea inicial de que dos nodos o están en un mismo subsistema, o no lo están. Forzar a que dos nodos estén en un mismo subsistema reduce previsiblemente la complejidad de resolución del mismo. Esto da la idea de diseñar un algoritmo de ramificación y poda en el que en un nivel superior podemos definir un árbol en el que el nodo inicial representa el problema inicial. Cada nodo tendrá dos hijos: uno será aquel en el que los nodos i y j están en el mismo subsistema, y otro aquel en el que dichos nodos están en subsistemas distintos. El criterio de selección de nodos a unir o separar se trata de manera diferenciada. Siguiendo con este procedimiento para cada nodo, se puede resolver también de manera óptima el problema *SMC*. Un aspecto clave a tener en cuenta es la cantidad de tiempo a dedicar en computar cada nodo. Si dejamos que un nodo compute hasta que CPLEX obtenga el óptimo, éste ya no se ramificará más (en particular, computar sólo el nodo raíz es equivalente a resolver el algoritmo propuesto en el apartado anterior). Si fijamos el nodo i en el mismo subsistema que el nodo j, y en una rama posterior del recorrido forzamos a que el nodo j esté en el mismo subsistema que el nodo k, es claro que los nodos i y k también estarán en el mismo subsistema, por lo que sería conveniente controlar que dichos nodos no se seleccionasen para unirse en el futuro. Para realizar esto utilizamos una estructura de datos para conjuntos disjuntos desarrollada por Tarjan [168]. Se modela un

UNIVERSIDAD DE MÁLAGA

nuevo grafo en el que añadimos una arista (i, j) cada vez que unamos los nodos i y j en el árbol usado para resolver el *SMC*. Esta unión de nodos solo será posible si en la estructura de Tarjan unimos dos nodos que estén en distintas componentes conexas, y su tratamiento se realiza eficientemente sin más que definir un vector de tamaño n y dos métodos asociados, *unir()* y *leer()*. Obsérvese que no es posible realizar el mismo razonamiento para el caso de separación de nodos, ya que si i debe estar separado de j, y j debe estar separado de k, no necesariamente tienen que estar separados i y k. Sin embargo, si i está separado de j, también tendría que ocurrir que i esté separado de todos aquellos nodos que ya están unidos a j.

Fijar o separar nodos se puede modelar fácilmente añadiendo nuevas restricciones al modelo. Distinguiendo los casos en que exista o no la arista en el grafo MDG para los nodos que se pretenden unir o separar, las restricciones asociadas se muestran en la Tabla 6.4.

Tabla 6.4: Restricciones a añadir al problema *SMC* cuando unimos nodos a un mismo subsistema, o forzamos a que estén separados.

	Existe arista (i, j)	No existe arista (i, j)
Unir nodos i y j	$\sum_{k=1}^{n} M_{ijk} = 1$	$x_{ik'} + \sum_{k=1, k\neq k'}^{n} x_{jk} \leq 1,\ \forall k^{'} = 1, \ldots, n$
Separar nodos i y j	$\sum_{k=1}^{n} M_{ijk} = 0$	$x_{ik} + x_{jk} \leq 1,\ \forall k = 1, \ldots, n$

Los resultados preliminares obtenidos para la instancia mtunis mejoran respecto a la propuesta inicial, ya que en aproximadamente cinco minutos se obtiene una solución con MQ=2.314... que es igual a la obtenida en el algoritmo de ramificación y poda. Sin embargo, para el caso de instancias con un número muy grande de nodos, como el caso de grappa, son necesarias muchas uniones de nodos (llegar aproximadamente a un nivel de profundidad 50) antes de que CPLEX empiece a encontrar soluciones factibles con $MQ > 1$, por lo que este método no es aún mejor que el obtenido por la metaheurística de Praditwong et al., a nivel general, aunque sí mucho mejor que para el caso de ramificación y poda.

También para esta propuesta se ha considerado añadir una nueva restricción de manera que durante el proceso de ramificación, la poda se produzca si la solución actual no es al menos igual que la anterior. La restricción a añadir sería

$$\sum_{k=1}^{n} \hat{h}_k \geq MQ, \tag{6.55}$$

siendo MQ el mejor valor objetivo obtenido hasta el momento durante la exploración. Dado que hay que limitar el tiempo de ejecución para cada nodo (de lo contrario ya tendríamos el óptimo para ese subproblema), no resulta tampoco efectiva la incorporación de este hiperplano de corte, ya que experimentalmente se ha comprobado que no deja tiempo suficiente al resolutor ni siquiera para encontrar una solución factible al problema.

6.6. Conclusiones

En este capítulo se ha analizado el problema de agrupamiento de módulos software, con el diseño de dos algoritmos, uno de ellos de ramificación y poda, y otro mediante la modelización como un problema de programación lineal entera mixto. Dado que el número de particiones de un conjunto es un número de Bell, resulta prácticamente intratable resolver este problema de manera exacta. El primer algoritmo propuesto utiliza ramificación y poda. En la práctica, su comportamiento es similar al de un algoritmo enumerativo, ya que la cota usada a la hora de podar las ramas se podría decir que no es muy ajustada. Aunque en el segundo algoritmo propuesto el resolutor solo necesita una ejecución para dar con la solución óptima, ésta no se ha conseguido en ningún caso dentro de las primeras 48 horas, para las instancias analizadas, y además se ha constatado que los resultados obtenidos por las metaheurísticas son mejores.

Para el segundo algoritmo, investigaciones posteriores realizan un nuevo proceso de ramificación y poda mediante la unión o separación de parejas de nodos en un mismo o en distintos subsistemas. Se han diseñado varias formas de selección de la siguiente pareja de nodos a unir, produciéndose en todos los casos resultados similares. Se plantea la opción de poder realizar múltiples uniones de nodos. También sería interesante intentar dar una cota para que en la exploración se poden ramas lo antes posible, y se permita que la búsqueda no

UNIVERSIDAD DE MÁLAGA

solo acabe antes, sino que también aumente la probabilidad de encontrar mejores soluciones más rápidamente. Por ejemplo, suponiendo que el grafo MDG es conexo, es claro que $p_i > 0$ para al menos un $i \in \{1, \ldots, n\}$. Queda pendiente la realización de un estudio más profundo que permita sacar provecho de la estructura del grafo MDG e intentar conseguir un mejor rendimiento en la resolución del problema.

Aunque inicialmente se trató el *SMC* como un problema biobjetivo, donde se trataba de maximizar el valor de cohesión y minimizar el valor de acoplamiento en cada subsistema, comprobamos rápidamente que dicha formulación no tenía utilidad, debido a la correlación lineal existente entre los dos objetivos. Sustituyendo uno de los objetivos por el de maximizar el número de subsistemas, se produjo un rendimiento mucho menor que para el caso monoobjetivo. No obstante, como trabajo futuro plantearemos un estudio más profundo de este problema, para tratar de obtener resultados más competitivos utilizando la programación multiobjetivo.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Capítulo 7

Problema de Selección de Requisitos

Este último capítulo de la tesis está dedicado al problema de selección de requisitos, que es bien conocido en el campo de la Ingeniería del Software y ha sido abordado en multitud de trabajos que pueden encontrarse en la literatura. En la mayoría de los casos se utilizan métodos heurísticos para su resolución. Debido a los avances tecnológicos y a la velocidad de procesamiento de las máquinas actuales, resulta factible abordar estos problemas utilizando resolutores ILP. Si bien calcular todas las soluciones del problema podría resultar muy costoso, en ocasiones es suficiente con disponer de un conjunto de ellas que sea representativo, por ejemplo, garantizando que se encuentren bien distribuidas por el espacio objetivo.

En la primera sección describimos el problema y planteamos dos variantes del mismo: monoobjetivo y biobjetivo. En la formulación monoobjetivo presentaremos dos metaheurísticas que lo resuelven, ofreciendo además resultados computacionales. Para el caso biobjetivo se describen varios algoritmos *anytime* que resuelven el problema, aportando también experimentación computacional. Se finalizará el capítulo con las conclusiones obtenidas en ambas variantes, así como describiendo algunas líneas de investigación futura.

7.1. Descripción del problema

El problema de selección de requisitos o de la siguiente versión (en inglés, *Next Release Problem*, *NRP*) ha sido abordado en multitud de ocasiones en la literatura, tanto en su versión monoobjetivo, como biobjetivo. En su formulación usual, consiste en seleccionar un subconjunto de requisitos que se van a desarrollar en la siguiente versión de una aplicación software. Las compañías desarrollan y realizan un mantenimiento de sistemas software que pueden ser muy complejos, y necesitan determinar las características que deben añadirse en sus sistemas (incluyendo posible corrección de errores) como parte de la siguiente versión. El problema de selección de requisitos fue inicialmente propuesto por Bagnall et al. [7], y se demuestra que pertenece a la clase $\mathcal{NP}$-*Hard*, ya que el problema de la mochila se reduce al *NRP*. Tiene como objetivo satisfacer las necesidades de los clientes, minimizando el esfuerzo de desarrollo de la siguiente versión. La idea consiste en encontrar un subconjunto de requisitos o bien un subconjunto de clientes, que maximice una cierta propiedad, como podría ser la satisfacción de los clientes, mientras se mantiene una restricción de cota superior sobre otra propiedad, que podría ser el coste. Hay que tener en cuenta que el problema *NRP* solo tiene en cuenta la siguiente versión software. La planificación sobre varias versiones tiene mayor complejidad y se aborda dentro del ámbito conocido como *planificación de la siguiente versión* [81], (en inglés, *Release Planning Problem*), que puede considerarse como una generalización de este problema. En [158] se pueden ver aproximaciones híbridas del problema de planificación de la siguiente versión.

Cabe destacar que los procesos de desarrollo software, especialmente cuando se pretende la utilización de metodologías ágiles, conllevan un orden de priorización en los requisitos, no solo para identificar aquellos que pudieran ser más importantes, sino también para resolver conflictos que pudieran surgir entre diferentes versiones [97]. Algunos autores establecen diferencias entre varios tipos de decisiones sobre la planificación de la siguiente versión [113, 114, 157]. Algunas se centran en encontrar la mejor combinación de características a implementar, como en los problemas de planificación, y otros se centran en encontrar la mejor combinación

UNIVERSIDAD DE MÁLAGA

de requisitos, como estableceremos en este capítulo. Debido al esfuerzo computacional que conlleva elegir una serie de requisitos que serán añadidos al producto software (si lo hacemos de manera enumerativa), planteamos el problema como un problema de programación matemática. El uso de programación lineal entera para resolver el *NRP* se aplicó desde un principio, incluso antes de que se acuñara este nombre. Por aquel entonces, se resolvían relajaciones lineales del modelo, y también se usaron técnicas de *hill-climbing*. Durante más de quince años se han ido desarrollando tanto algoritmos heurísticos como exactos. Bagnall et al. [7] apuntaron que para instancias pequeñas era suficiente con usar técnicas de ILP, pero que el tiempo de computación era muy elevado para instancias más grandes. Sugirieron que el uso de una combinación de heurísticas e ILP podría generar un conjunto de soluciones suficientemente buenas para el *NRP* [158]. Los métodos de búsqueda basados en Ingeniería del Software [86, 87] (en inglés, *Search Based Software Engineering, SBSE*) también han sido ampliamente utilizados en problemas de ingeniería [146]. De hecho, en la literatura existente, la mayoría de los métodos que abordan este problema utilizan metaheurísticas. En particular, se han diseñado algoritmos aproximados para resolver el *NRP* monoobjetivo que usan algoritmos basados en colonias de hormigas [93].

En el año 2015, Veerapen et al. [174], resolvieron el problema con técnicas ILP tomando como resolutor el software de IBM, CPLEX, usando unas instancias proporcionadas por Xuan et al. [184]. Éstas fueron resueltas en pocas horas cuando años atrás era impensable, obviamente debido a las capacidades de cómputo de la actual generación de computadores y los avances en ILP.

Podemos considerar dos enfoques para modelar el actual *NRP* monoobjetivo. Por un lado, se puede exigir que todas las demandas de los clientes sean satisfechas para poder incluir un requisito en la siguiente versión software [1], o que la satisfacción del cliente sea parcialmente aceptable [56], es decir, que se permitan satisfacer solo ciertas demandas de cierto cliente, y esto suponga también un aumento de la función objetivo. En este capítulo consideramos el primer caso, es decir, será necesario satisfacer todas las demandas del cliente, de acuerdo con la formulación general dada por Bagnall et al. [7].

Es posible que algunos requisitos tengan dependencias. En [187] se definieron tres posibles tipos de relaciones entre requisitos, que se han adaptado dando nombre a cada una de ellas.

- Combinación ($r_i \bigodot r_j$): los requisitos r_i y r_j deben aparecer juntos en cualquier selección.
- Exclusión ($r_i \bigoplus r_j$): los requisitos r_i y r_j no pueden aparecer juntos en una selección.
- Implicación ($r_i \Rightarrow r_j$): el requisito r_i requiere que el requisito r_j se incorpore en la misma selección.

El primer conjunto de dependencias indica que existen requisitos que deben aparecer (o no) conjuntamente. Este tipo de dependencia se puede modelar reemplazando las variables con dependencia mutua por una nueva variable. La nueva variable formará parte de la solución si y solo si cada una de las variables que dependen mutuamente forma parte de la solución. Así, consideraremos que cada variable representa a un único requisito, o directamente a un conjunto de requisitos que debe aparecer conjuntamente. El segundo conjunto de dependencias establece que pueden existir situaciones en las que dos requisitos determinados no pueden ser desarrollados conjuntamente en la siguiente versión. El tercer conjunto de dependencias implica que ciertos requisitos necesitan de la inclusión de otros prerrequisitos previos. Un requisito puede necesitar previamente de la inclusión de otro, u otros, y éstos, a su vez, de otros requisitos.

En relación a los clientes, cada uno de ellos demanda una serie concreta de requisitos, que deben ser completamente satisfechos para poder considerar como beneficio el peso que proporciona dicho cliente. No se admite satisfacción parcial en este caso.

Formulación del *NRP*

A continuación formulamos las restricciones asociadas al *NRP* independientemente de que consideremos la versión monoobjetivo o biobjetivo del problema. Sea R un conjunto de n requisitos que todavía no han sido desarrollados y $r = (x_1, ..., x_n) \in \{0, 1\}^n$ el vector binario

UNIVERSIDAD DE MÁLAGA

de requisitos, donde la componente x_i toma el valor 1 si y solo si el i-ésimo requisito es seleccionado para la siguiente versión. Sea S el conjunto de m clientes y $s = (y_1, ..., y_m) \in \{0,1\}^m$ el vector binario de clientes, donde la k-ésima componente vale 1 si y solo si todos los requisitos del cliente k están incluidos en la siguiente versión. Sea $c = (c_1, \ldots, c_n)$ el vector de costes asociado a los requisitos y $w = (w_1, \ldots, w_m)$ el vector de pesos asociado a los clientes, que representa la importancia de cada uno de ellos. Llamemos b al límite de presupuesto para el desarrollo de la siguiente versión del software. Sea P el conjunto de pares (i, j) donde el requisito i es un prerrequisito para el requisito j; sea Q el conjunto de pares (i, k) donde el requisito i es requerido por el cliente k; y sea E el conjunto de pares (i, j) tales que el requisito i no puede aparecer junto al requisito j. El *NRP* posee las siguientes restricciones:

$$x_i \geq x_j \quad \forall (i,j) \in P, \tag{7.1}$$
$$x_i \geq y_k \quad \forall (i,k) \in Q, \tag{7.2}$$
$$x_i + x_j \leq 1 \quad \forall (i,j) \in E, \tag{7.3}$$
$$x_i \in \{0,1\} \quad \forall i \in \{1, \ldots, n\}, \tag{7.4}$$
$$y_k \in \{0,1\} \quad \forall k \in \{1, \ldots, m\}, \tag{7.5}$$

donde la Ecuación (7.1) representa las restricciones de precedencia en los requisitos, la Ecuación (7.2) fuerza a que todos los requisitos del cliente sean satisfechos, la Ecuación (7.3) impide que dos requisitos excluyentes sean seleccionados conjuntamente y las Ecuaciones (7.4) y (7.5) representan los dominios de definición. Denotaremos por X al conjunto de vectores binarios (r, s) que verifican las Ecuaciones (7.1)–(7.5).

Para la versión monoobjetivo del *NRP*, utilizaremos como objetivo la maximización de la satisfacción de los clientes. El otro objetivo, que consiste en minimizar los recursos empleados, será incluido como restricción adicional al modelo (ver Ecuación (7.7)).

$$\text{máx} \sum_{k=1}^{m} w_k y_k, \tag{7.6}$$
$$s.a. \sum_{i=1}^{n} c_i x_i \leq b, \tag{7.7}$$
$$(r, s) \in X. \tag{7.8}$$

La versión biobjetivo del *NRP* fue inicialmente fue formulada por Zhang et al. [187]. En este caso la cota superior sobre el coste, que se utiliza como restricción en la versión monoobjetivo, se transforma en un nuevo objetivo a optimizar. Después de resolver el problema, no habrá una solución óptima, sino un conjunto de ellas que formará el conjunto eficiente.

$$\text{mín } f(x) = \left(f_1(s) = -\sum_{k=1}^{m} w_k y_k \quad , \quad f_2(r) = \sum_{i=1}^{n} c_i x_i \right), \tag{7.9}$$
$$s.a. \qquad (r, s) \in X. \tag{7.10}$$

Podemos observar que se ha definido el problema usando minimización. Es por ello que el objetivo f_1, que maximiza la satisfacción de los clientes, ha sido multiplicado por -1.

7.2. *NRP* monoobjetivo

En esta sección presentamos un algoritmo híbrido, basado en una conocida metaheurística llamada *Construct, Merge, Solve and Adapt* (CMSA), que resuelve el *NRP* monoobjetivo. Antes de desarrollar dicho algoritmo, describiremos brevemente el funcionamiento de CMSA. Acto seguido mostraremos los resultados computacionales asociados a la resolución de este problema.

UNIVERSIDAD DE MÁLAGA

7.2.1. Algoritmo híbrido usando CMSA

Blum et al. [12] diseñaron un algoritmo genérico que combinaba metaheurísticas con programación lineal entera, que resolvía problemas de optimización combinatoria. Concretamente, su trabajo es una instanciación específica dentro de un marco conocido en la literatura como *Generate-and-Solve* [96], y se conoce como *Construct, Merge, Solve and Adapt* (CMSA).

La idea que proponen para resolver el problema consiste en generar una subinstancia del problema original, donde la solución óptima sea también una solución factible del problema original. La subinstancia se genera mediante la selección de ciertos componentes a tener en cuenta. Después, mediante un resolutor exacto, se obtiene una solución óptima para el subproblema. Esta solución obtenida podría estar muy alejada del verdadero óptimo. Sin embargo, los componentes que forman parte de la solución óptima del subproblema tienen un peso mayor para formar parte de la siguiente subinstancia, al menos durante un tiempo determinado. Así, se irán resolviendo subproblemas y guardando la mejor solución obtenida hasta el momento, hasta que se alcance un tiempo límite de ejecución previamente establecido. La descripción de este proceso general tiene también una fase de adaptación donde aquellos componentes que forman parte del subproblema y no han aparecido como parte de la solución óptima dentro de un número establecido de iteraciones, son desechados, lo cual no implica que puedan ser seleccionados en el futuro para formar parte de la solución. Este método híbrido, donde una heurística se mezcla con el uso de un resolutor exacto, tiene la ventaja de obtener soluciones de alta calidad (debido al uso de algoritmos exactos) a la vez que resultan rápidas, por resolver de manera exacta solo subproblemas de un tamaño reducido. La resolución de instancias de un problema reducido ha sido ya explorada en varios trabajos, como por ejemplo en [32] y [104]. El método CMSA ha sido aplicado a numerosos problemas, como el *Minimum Common String Partition* (MCSP) [134] o el *Minimum Covering Arborescence* (MCA) [155], entre otros. En esta sección, aplicaremos esta técnica para la resolución del *NRP*.

Algoritmo 14 *Construct, Merge, Solve and Adapt (CMSA)*

Entrada: Instancia I, valores de los parámetros n_a y age_{max}
Salida: S_{bsf}

```
1:  S_bsf = NULL
2:  C' = ∅
3:  age[c] = 0 para todo c ∈ C
4:  mientras tiempo de CPU no excedido hacer
5:    para i = 1 hasta n_a hacer
6:      S=ProbabilisticSolutionGenerator(C)
7:      para todo c ∈ S y c ∉ C' hacer
8:        age[c] = 0
9:        C' = C' ∪ {c}
10:     fin para
11:   fin para
12:   S'_opt=ApplyExactSolver(C')
13:   si S'_opt es mejor que S_bsf entonces S_bsf = S'_opt
14:   Adapt (C', S'_opt, age_max)
15: fin mientras
```

El método CMSA, cuyo pseudocódigo puede verse en el Algoritmo 14, parte de la suposición de que cada solución válida para una instancia I de un problema P dado, se puede representar mediante un subconjunto de un conjunto C de componentes. Este concepto de componente puede referirse a un conjunto de variables del problema, o bien puede representar algo distinto. Lo importante es que si tenemos a nuestra disposición todos los componentes para formar una solución y escogemos la mejor de ellas mediante un algoritmo exacto, resolvemos el problema original. Si solo disponemos de un subconjunto de C para formar soluciones, entonces estaremos resolviendo una subinstancia de la instancia original. El conjunto $S \subset C$ representa una solución, no necesariamente óptima, al problema original. Por otro lado, el subconjunto C' es un contenedor que va guardando todos los componentes seleccionados y que serán utilizados para resolver una subinstancia de I. Por tanto, también es $C' \subset C$. El bucle principal de las Líneas 4–15 se ejecuta mientras no se alcance el máximo tiempo de

UNIVERSIDAD DE MÁLAGA

ejecución permitido al algoritmo. Cada iteración del algoritmo se puede dividir en cuatro partes:

1. *Construct*: se generan probabilísticamente n_a soluciones (Línea 6).
2. *Merge*: las componentes de dichas soluciones son añadidas al contenedor $C^{'}$ y, a cada componente c que ha sido nuevamente añadida al contenedor, se le inicializa su edad a cero (Líneas 7–10).
3. *Solve*: se resuelve la subinstancia formada por las componentes del contenedor (Línea 12). Si la solución encontrada es mejor que la obtenida hasta el momento, se actualiza.
4. *Adapt*: el contenedor se adapta en base a la solución obtenida al aplicar el resolutor exacto (Línea 14). Cada componente de la solución óptima del subproblema reinicializa su edad a cero, y el resto de componentes aumentan su edad en una unidad. Si se supera un límite máximo de edad preestablecido, age_{max} el componente es eliminado del contenedor, aunque podrá ser incluido nuevamente en una posterior iteración.

CMSA para *NRP*

Presentamos una adaptación de CMSA para resolver el NRP, suponiendo que no existen requisitos de exclusión. La estructura base del programa ya ha sido expuesta anteriormente y lo único que resta por explicar es el contenido de las funciones *ProbabilisticSolutionGenerator* y *ApplyExactSolver*. En primer lugar, es necesario definir el concepto de componente para resolver una instancia del *NRP*. De manera natural surgen dos posibilidades: que los componentes sean los requisitos o que los componentes sean los clientes. El procedimiento *ProbabilisticSolutionGenerator* debe generar soluciones aleatorias y de gran calidad para servir de base al resolutor exacto. Para conseguir soluciones de calidad sería deseable obtener óptimos locales, para una cierta definición de vecindario. Debemos tener en cuenta que cuando se añade un requisito a la solución parcial, es necesario añadir también todos sus prerrequisitos. Explicamos a continuación como implementar la función *ProbabilisticSolutionGenerator* para cada modalidad sobre el concepto de componente, y después explicamos también como se consideran los prerrequisitos.

Si consideramos que un componente está formado por un requisito, estamos interesados en generar un vector binario $(x_1, \ldots, x_n)$ que sea un máximo local. Para ello, y teniendo en cuenta que el objetivo es maximizar el beneficio de los clientes, hemos optado por ir seleccionando requisitos que vayan saturando las demandas de los clientes, ya que una selección totalmente aleatoria de requisitos podría implicar una baja satisfacción de los clientes, y por tanto, un bajo valor objetivo. Así, iremos seleccionando aleatoriamente clientes, y añadiendo todas sus demandas mientras el coste total de los requisitos y los prerrequisitos asociados a estos requisitos no sobrepasen el límite establecido. Si la selección de un cliente hace que alguna de sus demandas supere el límite de coste total, éste se descarta y se pasa a otro. Este proceso se repite hasta que todos los clientes han sido analizados. En el siguiente paso, una vez analizados todos los clientes, si todavía no se ha alcanzado la cota máxima para el coste, se seleccionan de manera aleatoria más requisitos a formar parte de la solución, siempre que esto sea posible.

Si consideramos que un componente está formado por un cliente, estamos interesados en generar un vector binario $(y_1, \ldots, y_m)$ que sea un máximo local. Para ello procedemos de la misma forma que en el caso anterior, pero teniendo en cuenta que vamos seleccionando clientes, y no requisitos, y que no se seleccionan los requisitos finales para tratar de saturar la restricción de coste total.

Para calcular los prerrequisitos asociados a un requisito usamos inicialmente una estructura de datos consistente en un vector $Predecesor = (p_1, \ldots, p_n)$ donde $p_i = i$, si el requisito no tiene ningún prerrequisito asociado, y $p_i = j$, si el requisito i tiene como prerrequisito a j. De esta forma, resulta sencillo gestionar de manera recursiva el cálculo de todos los prerrequisitos asociados a un requisito. Sin embargo, es posible que un requisito tenga varios prerrequisitos asociados, por lo que el vector *Predecesor* pasa a convertirse en un vector donde cada elemento es una lista de elementos.

UNIVERSIDAD DE MÁLAGA

Algoritmo 15 *add_components(type)*

1: $aux = \emptyset$
2: $X = (0, \ldots, 0)$
3: $S = (0, \ldots, 0)$
4: $totalcost = 0$
5: **mientras** queden clientes por seleccionar **hacer**
6: $i \leftarrow$ elegir cliente aleatoriamente y seleccionarlo
7: $pcost = 0$
8: $aux = \emptyset$
9: **para** cualquier requisito r no incluido previamente y demandado por i **hacer**
10: $X[index(r)] = 1$
11: $pcost = pcost + c_{index(r)}$ //actualizar *pcost*
12: $aux = aux \cup \{index(r)\}$
13: **para** todo prerrequisito r' de r no seleccionado previamente **hacer**
14: $X[index(r')] = 1$
15: $pcost = pcost + c_{index(r')}$ //actualizar *pcost*
16: $aux = aux \cup \{index(r')\}$
17: **fin para**
18: **fin para**
19: **si** $totalcost + pcost \leq b$ **entonces**
20: $totalcost = totalcost + pcost$
21: $S[i] = 1$
22: **si no**
23: **para todo** $j \in aux$ **hacer** $X[j] = 0$
24: **fin si**
25: **fin mientras**
26: **si** *type* = 'requisitos' **entonces return** (X)
27: **si** *type* = 'clientes' **entonces return** (S)

En el Algoritmo 15 se muestra como se añaden los componentes a la solución parcial dependiendo de si son requisitos o clientes, y en el Algoritmo 16 se muestra la función *ProbabilisticSolutionGenerator*. El vector X es un vector binario de tamaño n donde cada componente toma el valor 1 si y solo si el requisito correspondiente ha sido seleccionado. El vector S es un vector binario de tamaño m donde cada componente toma el valor 1 si y solo si el cliente correspondiente es seleccionado. La variable *pcost* almacena el coste parcial que será añadido, en su caso, a la variable *totalcost*, que almacena el coste total acumulado y que nunca debe ser mayor que b. El conjunto aux guarda los índices de los requisitos y prerrequisitos seleccionados para formar parte de la solución. Si se viola la restricción de coste máximo, gracias al conjunto aux es posible restaurar el vector X.

A continuación explicamos la función *ApplyExactSolver*. Suponemos que disponemos de un resolutor que resuelve problemas ILP. De una manera muy sencilla podemos adaptar el contenido de esta función sin más que añadir una restricción al modelo, aquella que no tiene en cuenta los requisitos que no están en el contenedor o los clientes que no están en el contenedor, según el caso. El pseudocódigo de esta función puede verse en el Algoritmo 17.

Cabe la posibilidad de plantearse que los dos enfoques que hemos usado para determinar qué son los componentes, considerando a estos como requisitos, o como clientes, tengan el mismo comportamiento, ya que en realidad estamos maximizando la satisfacción de los clientes, y cuando los componentes son los requisitos, estos se seleccionan teniendo en cuenta también la total satisfacción de los clientes. Sin embargo, el hecho de seleccionar después más requisitos aleatoriamente (ver Línea 5 del Algoritmo 16) para tratar de saturar la restricción de coste total, permite que el subproblema esté formado por distintos elementos, y por tanto, el funcionamiento al aplicar CMSA podrá diferir en los dos enfoques.

7.2.2. Resultados computacionales

Presentamos los resultados computacionales para la versión monoobjetivo del *NRP*. Comparamos los resultados obtenidos usando tres métodos, un algoritmo exacto y dos heurísticos

UNIVERSIDAD DE MÁLAGA

Algoritmo 16 *ProbabilisticSolutionGenerator(type)*

```
1: S = add_components(type)
2: si type = 'requisitos' entonces
3:    marcar como requisito no seleccionado cualquier r tal que S[index(r)] = 0
4:    mientras queden requisitos por seleccionar hacer
5:       r ← elegir requisito aleatoriamente y seleccionarlo
6:       R' ← seleccionar prerrequisitos de r no seleccionados previamente
7:       si el coste añadido no supera el límite entonces
8:          S[index(r)] = 1
9:          S[index(r*)] = 1 para todo r* ∈ R'
10:         actualizar totalcost
11:      fin si
12:   fin mientras
13: fin si
14: return (S)
```

Algoritmo 17 *ApplyExactSolver* $(C^{'})$

```
1: Añadir al problema original la restricción Σ_{x∉C'} index(x) = 0
2: S'_opt ← obtener solución de la subinstancia
3: Borrar la restricción añadida previamente
4: return (S'_opt)
```

(basados en CMSA), para instancias con un gran número de requisitos, clientes y dependencias. Las instancias del *NRP* proporcionadas por Xuan et al. [184] ya han sido resueltas de manera exacta en pocos segundos, así que no tendría sentido aplicar ninguna de las heurísticas en este caso. Es por ello que hemos creado instancias aleatorias con un número muy elevado de requisitos, clientes y dependencias, para poder así comprobar los resultados en relación a los que proporciona la heurística CMSA en sus dos variantes (siendo los componentes los requisitos o los clientes) y la que proporciona el propio resolutor cuando resolvemos el problema exacto durante un tiempo prefijado.

Las instancias aleatorias han sido creadas teniendo en cuenta que no se produzcan bucles en la lectura de los prerrequisitos, lo cual se ha logrado usando estructuras *Union-Find* [33, cap. 21]. No existe ninguna relación de exclusión ni de combinación entre requisitos. Llamamos n al número de requisitos, m al número de clientes y k al número de dependencias entre requisitos. Los costes de los requisitos son números aleatorios enteros entre 1 y 10. Los pesos de los clientes son números aleatorios enteros entre 1 y 100. Cada cliente demanda un número aleatorio de requisitos que varía entre 1 y 8. Se han generado seis grupos de instancias, con los prefijos desde 'a' hasta 'e'. Para nombrar una instancia, se indica el nombre del grupo y los valores n, m, y k, separados por guiones entre ellos. Cada grupo representa una relación distinta entre los parámetros variables n, m y k. Así, se han considerado casos en que n puede ser mayor que m (grupos a y b), mucho mayor (grupos c y d) o similar (grupos e y f), al igual que m puede ser mayor que k (grupos a, b, c ó e) o similar. Como límite total de coste para los requisitos se suele utilizar un coeficiente reductor sobre la suma total de los costes de todos los requisitos. En nuestro caso hemos optado por usar dos valores para este coeficiente: 0.3 y 0.7. El tiempo de ejecución máximo ha sido fijado en 60 segundos. Como resolutor hemos usado la versión 12.6.2. de CPLEX. Los experimentos han sido ejecutados bajo entorno Linux (Ubuntu 16.04 LTS) en un máquina HP con Intel Core 2 Quad (Q9400), velocidad de procesador 2.7 GHz y 4 GB de RAM, usando un máximo de 2GB de RAM y un único núcleo para cada ejecución.

Los tres métodos usados, cuyos resultados serán comparados entre ellos son:

1. Algoritmo exacto usando el problema original y estableciendo tiempo límite de ejecución (*exact*)

2. Algoritmo CMSA utilizando los requisitos como componentes (*cmsa_r*) y parámetros $n_a = 5$, $age_{max} = 2$

UNIVERSIDAD DE MÁLAGA

3. Algoritmo CMSA utilizando los clientes como componentes (*cmsa_s*) y parámetros $n_a = 5$, $age_{max} = 2$

En el caso del algoritmo exacto, se ha ejecutado 10 veces cada instancia, ya que los resultados reportados por CPLEX pueden variar en distintas ejecuciones debido a la diferente carga de la máquina. Las variantes de CMSA se han ejecutado 30 veces para cada instancia, un número considerablemente mayor dado el componente aleatorio de la heurística. En todos los casos se utiliza el valor promedio dado por las funciones objetivo. Los resultados para los distintos coeficientes reductores pueden verse en las Tablas 7.1 y 7.2. Las instancias marcadas con asterisco en la primera columna de las Tablas 7.1 y 7.2 son aquellas en las que el algoritmo exacto obtiene el óptimo global.

Tabla 7.1: Valores objetivo promedio para los tres métodos propuestos, usando instancias generadas aleatoriamente durante 60 segundos y con presupuesto $b = 0.3 \sum_{i=1}^{n} c_i$.

Instancia	*exact*	*cmsa_r*	*cmsa_s*
a20000-15000-4000	146018.0	**263387.1**	253287.2
a40000-30000-8000	287108.0	337520.6	**479641.2**
a80000-60000-16000	**562858.0**	506890.8	558241.5
a120000-90000-24000	**866539.0**	777432.0	852102.8
b20000-15000-15000	89272.0	103588.9	**178089.7**
b40000-30000-30000	166559.0	134943.8	**323032.7**
b80000-60000-60000	**340646.0**	276249.3	291939.1
b120000-90000-90000	360537.8	418888.4	**443369.2**
c20000-5000-4000*	**168572.0**	**168572.0**	**168572.0**
c40000-10000-8000*	**329473.0**	326309.9	325006.1
c80000-20000-16000	290985.0	301780.0	**371700.9**
c120000-30000-24000	449524.0	449683.2	**458293.3**
d20000-5000-5000*	**161697.0**	161696.7	**161697.0**
d40000-10000-10000*	**322819.0**	315514.8	313263.4
d80000-20000-20000	269804.0	281128.0	**295590.2**
d120000-30000-30000	391402.0	392500.8	**398782.4**
e20000-18000-4000	175321.0	**290948.8**	269240.6
e40000-36000-8000	355931.0	494322.9	**525250.4**
e80000-76000-16000	**770482.0**	654382.5	707105.5
e120000-108000-24000	**1078305.9**	926170.8	1004343.5
f20000-18000-18000	87259.0	107617.3	**199510.7**
f40000-36000-36000	177267.0	146402.7	**319233.9**
f80000-75000-75000	**361914.0**	295269.3	315160.4
f120000-108000-108000	-	435019.0	**459930.6**

El análisis para el coeficiente reductor 0.3 de la Tabla 7.1 muestra una variedad en cuanto a rendimientos. Se ha marcado en negrita el mejor resultado para cada una de las instancias, teniendo en cuenta los valores promedio. Puede observarse que para aquellas instancias en las que el algoritmo exacto es mejor, el valor objetivo proporcionado por las heurísticas no es muy distante. Sin embargo, en aquellas instancias en las que la heurística es mejor, la diferencia en relación a la solución obtenida por el algoritmo exacto es, en ocasiones, muy elevada, como por ejemplo en la instancia a40000-30000-8000, donde *cmsa_s* proporciona un objetivo en torno al 67 % mejor que el algoritmo exacto. Esto justifica la utilidad del uso de estos nuevos algoritmos. También aparecen instancias en las que los resultados obtenidos por los tres métodos son iguales, como en el caso de c20000-5000-4000. Obsérvese también que en la instancia f120000-108000-108000, al tener gran cantidad de requisitos, clientes y dependencias, CPLEX es incapaz de encontrar ninguna solución durante el primer minuto de ejecución. Se conjetura que en torno a esos valores elevados de requisitos y clientes empieza a observarse un umbral a partir del cual parece ser más productivo el uso de la heurística CMSA en favor del uso de algoritmos exactos. Si comparamos *cmsa_r* con *cmsa_s*, podemos comprobar que en la mayoría de los casos *cmsa_s* proporciona unos resultados mejores. Este hecho nos hace conjeturar que al considerar los requisitos como componentes, los requisitos extra que se seleccionan aleatoriamente tratando de saturar al máximo la restricción de coste

UNIVERSIDAD DE MÁLAGA

TABLA 7.2: Valores objetivo promedio para los tres métodos propuestos, usando instancias generadas aleatoriamente durante 60 segundos y con presupuesto $b = 0.7 \sum_{i=1}^{n} c_i$.

Instancia	*exact*	*cmsa_r*	*cmsa_s*
a20000-15000-4000	420936.0	417669.1	**438157.3**
a40000-30000-8000	856336.0	867279.3	**888971.7**
a80000-60000-16000	562858.0	557780.3	**564892.4**
a120000-90000-24000	866539.0	858235.1	**869302.4**
b20000-15000-15000	325142.0	330771.0	**343229.2**
b40000-30000-30000	650947.5	664219.9	**677113.9**
b80000-60000-60000	**1237348.0**	1164320.8	1220072.4
b120000-90000-90000	463548.6	438897.5	**733116.3**
c20000-5000-4000*	**249727.0**	**249727.0**	**249727.0**
c40000-10000-8000*	**492957.0**	**492957.0**	**492957.0**
c80000-20000-16000*	**993343.0**	**993343.0**	**993343.0**
c120000-30000-24000*	**1484724.0**	**1484724.0**	**1484724.0**
d20000-5000-5000*	**244459.0**	**244459.0**	**244459.0**
d40000-10000-10000*	**493255.0**	**493255.0**	**493255.0**
d80000-20000-20000*	**993806.0**	**993806.0**	**993806.0**
d120000-30000-30000*	**1485762.0**	**1485762.0**	**1485762.0**
e20000-18000-4000	433501.0	438125.7	**458748.1**
e40000-36000-8000	877810.0	887865.9	**924538.8**
e80000-76000-16000	770482.0	760625.6	**773035.3**
e120000-108000-24000	1077713.5	888231.8	**1082415.2**
f20000-18000-18000	87259.0	**352584.9**	319687.6
f40000-36000-36000	177267.0	704608.2	**719070.1**
f80000-75000-75000	**1383684.0**	1283686.5	1099710.2
f120000-108000-108000	-	-	**158342.4**

total vienen a 'confundir' al resolutor en las subinstancias que resuelve, y le hacen perder tiempo porque son innecesarios.

Analizando ahora la Tabla 7.2 para las mismas instancias de antes, pero aumentado el coeficiente reductor a 0.7, realizando el mismo número de ejecuciones que en el caso anterior, y calculando promedios, se observa como los resultados varían considerablemente. Para los grupos de instancias *c* y *d* los objetivos obtenidos son iguales para los tres casos (la solución óptima), por lo que no se detecta ninguna diferencia en el uso de los tres algoritmos. Sin embargo, para la mayoría de las restantes instancias, es claramente *cmsa_s* el método con el que se obtienen mejores resultados. Si suprimimos los grupos en el que los resultados son iguales, el método exacto solo es mejor en dos casos, mientras que *cmsa_r* solo lo es en un caso. Además, para la instancia f120000-108000-108000, ni el algoritmo exacto, ni *cmsa_r* consiguen encontrar una solución factible dentro del primer minuto de ejecución. Este hecho resalta nuevamente la importancia de una buena elección de componentes, que en este caso parece ser favorable a *cmsa_s*.

Todos los experimentos se han realizado fijando los parámetros $n_a = 5$ y $age_{max} = 2$, aunque podrían haberse obtenido resultados diferentes (posiblemente mejores) para otra configuración de estos parámetros. Tras realizar el test de Wilcoxon con datos apareados a cada pareja de algoritmos y en cada una de las modalidades, los resultados de las Tablas 7.3 y 7.4 muestran que existen diferencias significativas en cuanto al comportamiento del algoritmo *cmsa_s* respecto a los otros dos (al nivel de significancia del 5 %). Realizando nuevamente los tests pero estableciendo como hipótesis alternativa que *cmsa_s* es mejor, se obtienen los resultados que se muestran en la Tabla 7.5. En conclusión, podemos garantizar estadísticamente que, con estas instancias, el algoritmo *cmsa_s* es el mejor de los tres algoritmos analizados.

TABLA 7.3: Test de Wilcoxon (datos apareados) para contrastar si hay diferencias entre los algoritmos propuestos, $b = 0{,}3 \sum_{i=1}^{n} c_i$.

	exact	*cmsa_r*
cmsa_r	0.89110	-
cmsa_s	0.01629	0.00040

UNIVERSIDAD DE MÁLAGA

Tabla 7.4: Test de Wilcoxon (datos apareados) para contrastar si hay diferencias entre los algoritmos propuestos, $b = 0{,}7\sum_{i=1}^{n} c_i$.

	exact	*cmsa_r*
cmsa_r	0.97730	-
cmsa_s	0.01620	0.02449

Tabla 7.5: Contrastes unilaterales con hipótesis alternativa H_1 : $cmsa_s > col$, siendo $col \in \{exact, cmsa_r\}$.

	exact	*cmsa_r*
$b = 0{,}3\sum_{i=1}^{n} c_i$	0.00814	0.00020
$b = 0{,}7\sum_{i=1}^{n} c_i$	0.00810	0.01225

7.3. *NRP* biobjetivo

Veerapen et al. [174] mostraron que hoy día los resolutores ILP pueden resolver la versión biobjetivo del *NRP* en pocas horas, para tamaños razonables de las instancias. Ellos usaron el método ε-*constraint* para obtener el frente de Pareto completo. El inconveniente en el uso de esta técnica es que si el algoritmo se para antes de que finalice su ejecución, el frente de Pareto parcial estaría ubicado en una región específica del espacio objetivo, lo cual no resultaría muy útil al decisor. Es por ello que los algoritmos *anytime* resultan más adecuados en este caso.

7.3.1. Algoritmos clásicos que resuelven el *NRP* biobjetivo

Dedicamos este apartado a la presentación de los métodos clásicos que utilizaremos en los experimentos computacionales para resolver el *NRP* biobjetivo. Tres de ellos ya han sido descritos en la Sección 3.1.1.

- Método ε-constraint. Tiene dos variantes, según cual sea el objetivo a considerar en la función objetivo. Las soluciones encontradas son débilmente eficientes, y se necesita realizar un filtrado de soluciones dominadas. Lo llamaremos *Econst1*.
- Método híbrido. Las soluciones generadas son eficientes. Lo llamaremos *EHybrid*.
- Método ponderado y aumentado de Tchebycheff. Las soluciones generadas son eficientes. Lo llamaremos *Tchebycheff*.

Además de los tres métodos anteriores, presentamos dos nuevos métodos clásicos, basados en el método ε-constraint, y que también serán utilizados en los experimentos computacionales.

Método ε-constraint con dos iteraciones

Calcula soluciones eficientes, utilizando para ello dos llamadas al resolutor. Lo llamaremos *Econst2*. Tiene la ventaja de no tener que ir filtrando soluciones, y el inconveniente de tener que realizar un mayor esfuerzo computacional para hallar los puntos eficientes. Supongamos que queremos resolver el modelo $\min_{x \in \mathcal{X}} \{f_1(x);\ s.a.\ f_2(x) \leq \epsilon_2\}$. Sabemos que la solución obtenida, x^*, es débilmente eficiente. Si ahora resolvemos el modelo $\min_{x \in \mathcal{X}} \{f_2(x);\ s.a.\ f_1(x) \leq f_1(x^*)\}$, entonces podemos garantizar que la solución obtenida, x^{**}, será eficiente. De manera análoga se pueden plantear las dos llamadas al resolutor intercambiando el orden de las funciones f_1 y f_2. Para más detalle, véase Bérubé et al. [10].

Método *Augmecon*

El método *Augmecon* [130, 133] resuelve problemas multiobjetivo (biobjetivo en particular), y también está basado en el método ε-constraint. Con solo añadir una nueva variable y modificar la función objetivo y una restricción, *Augmecon* es capaz de encontrar una solución

UNIVERSIDAD DE MÁLAGA

eficiente en una sola llamada al resolutor. La nueva variable creada tiene coeficiente $\lambda > 0$ en la función objetivo. Este valor es generalmente un parámetro fijo en el intervalo $\left[10^{-6}, 10^{-3}\right]$. Por ejemplo, si estamos interesados en minimizar f_1, entonces el modelo a resolver sería $\min_{x \in \mathcal{X}} \{f_1 - \lambda t;\ s.a.\ f_2(x) + t \leq \varepsilon\}$. Si λ es muy grande, el algoritmo puede omitir soluciones. Si λ es muy pequeño, podría generar soluciones débilmente eficientes, debido a errores numéricos producidos por los redondeos que realiza el resolutor. El valor necesario depende del problema, pero en general funciona bien si lo consideramos en el rango $\left[10^{-6}, 10^{-3}\right]$.

7.3.2. Algoritmos *anytime* eficientes que resuelven el *NRP* biobjetivo

En este apartado presentamos una mejora en los métodos existentes en el estado del arte definiendo seis algoritmos que resuelven el *NRP* biobjetivo. El primero de ellos solo calcula soluciones soportadas. Los cinco algoritmos restantes pueden considerarse *anytime* ya que las soluciones están bien distribuidas por el espacio objetivo y con suficiente tiempo de ejecución son capaces de calcular el frente de Pareto completo.

Antes de comenzar con la descripción de los algoritmos, recuérdense las definiciones de solución eficiente, débilmente eficiente, soportada, y lexicográfica. Todos estos conceptos fueron definidos en el Capítulo 2. En general, en un problema combinatorio biobjetivo existirán dos soluciones óptimas lexicográficas, ya que de lo contrario, el problema sería trivial (solo habría una solución eficiente, o el problema sería infactible). Supongamos que las imágenes de las dos soluciones óptimas lexicográficas son $z^1, z^2 \in PF$ con $z_1^1 < z_1^2$. Entonces, $z_2^1 > z_2^2$, donde $z_1^1 = \min_{x \in \mathcal{X}} \{f_1(x)\}$, $z_2^1 = \min_{x \in \mathcal{X}} \{f_2(x) \mid f_1(x) \leq z_1^1\}$, $z_2^2 = \min_{x \in \mathcal{X}} \{f_2(x)\}$ y $z_1^2 = \min_{x \in \mathcal{X}} \{f_1(x) \mid f_2(x) \leq z_2^2\}$.

Definición 7.1. Sean z^1 y z^2 dos puntos bidimensionales con $z_1^1 < z_1^2$ y $z_2^1 > z_2^2$. Los puntos z^1 y z^2 forman una caja (rectángulo), como puede verse en la Figura 7.1. Consideremos la función

$$\delta(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x), \tag{7.11}$$

donde $\lambda_1 = z_2^1 - z_2^2$ y $\lambda_2 = z_1^2 - z_1^1$. Sea y una solución cuya imagen está en el interior de la caja formada por extremos opuestos z^1 y z^2. Decimos que y está en la *parte cóncava* de la caja (*Concave*) cuando $\delta(y) > \delta_0 = \lambda_1 z_1^1 + \lambda_2 z_2^1 = \lambda_1 z_1^2 + \lambda_2 z_2^2$, y que está en la *parte convexa* de la caja (*Convex*) cuando $\delta(y) \leq \delta_0$. Sin pérdida de generalidad, la caja quedará representada mediante $[z^1, z^2]$.

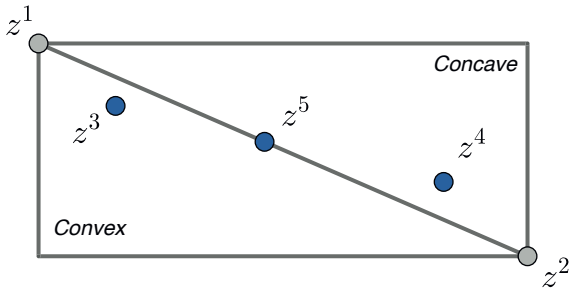


FIGURA 7.1: z^1 y z^2 forman una caja. Los puntos z^3 y z^5 pertenecen a la parte convexa de la caja, y el punto z^4 pertenece a la parte cóncava de la caja.

Nótese que el concepto de caja que hemos definido difiere de la definición original dada en la Ecuación (3.2). En este capítulo la caja quedará definida por los dos puntos del rectángulo que representan la esquina superior-izquierda e inferior-derecha, respectivamente. Los algoritmos que presentamos tienen todos una estructura similar. Comienzan computando la imágenes de las soluciones óptimas lexicográficas y definen un conjunto de cajas, definidas por su esquina

UNIVERSIDAD DE MÁLAGA

superior-izquierda e inferior-derecha. Mientras el conjunto de cajas a explorar no sea vacío, se extrae una de ellas, que será la que posea mayor área, para así tratar de conseguir puntos dispersos por el espacio objetivo mientras vamos obteniendo soluciones. Si la caja explorada tiene solución, dos nuevas cajas son creadas, descartando la región dominada por la última solución encontrada. Todos los algoritmos finalizan cuando se alcanza un tiempo límite de ejecución prefijado o cuando no quedan cajas por explorar. Los Algoritmos *SPF* (Algoritmo 18), *AnyAugmecon* (Algoritmo 19) y *AnyTchebycheff* (Algoritmo 20) pueden considerarse pequeñas modificaciones de trabajos ya publicados. Los Algoritmos *AnyHybrid* (Algoritmo 21), *MixHT* (Algoritmo 22) y *MixSHT* (Algoritmo 23) son nuevas contribuciones.

Algoritmo *SPF*

Veerapen et al. [174] usaron un algoritmo *anytime* basado en el método dicotómico de Aneja y Nair [3], llamado *ADS* (del inglés, *Anytime Dichotomic Search*). La idea es similar a la primera fase del algoritmo de dos fases de Ulungu y Teghem [171]. El algoritmo comienza computando los óptimos lexicográficos, forma una caja con extremos opuestos las imágenes de estos puntos, y añade dicha caja a la lista de regiones a explorar. En cada iteración del algoritmo se selecciona la caja $[z^1, z^2]$ con diagonal de mayor longitud. Se resuelve un problema parametrizando los dos objetivos y ponderándolos, donde los pesos son $\lambda_1 = z_2^1 - z_2^2$ y $\lambda_2 = z_1^2 - z_1^1$. Con estos valores, z^1 y z^2 toman el mismo valor objetivo. Si una nueva solución z es encontrada después de resolver el subproblema, su imagen es añadida al frente de Pareto y es usada para dividir la caja en dos nuevas cajas: $[z^1, z]$ y $[z, z^2]$. Si no hay solución, el algoritmo no hace nada y se extrae una nueva caja de la lista de regiones a explorar. Este proceso se itera hasta que se alcance un tiempo límite o no queden cajas por explorar.

Nuestra primera propuesta algorítmica, llamada *SPF* (del inglés *Supported Pareto Front*), se centra en la búsqueda de soluciones eficientes soportadas. Es una variante del método *ADS* propuesto por Veerapen et al. [174], que también encuentra únicamente soluciones soportadas. La principal diferencia entre ambos métodos es que *ADS* puede omitir algunas soluciones, mientras que *SPF* es capaz de encontrar completamente el frente soportado con suficiente tiempo de ejecución. El código de *SPF* puede verse en el Algoritmo 18. En la Línea 8 podemos observar que la versión escalarizada del problema contiene dos restricciones (una asociada a cada objetivo). Estas restricciones previenen que el algoritmo encuentre una solución previamente encontrada, forzando la búsqueda de una nueva solución en la región a explorar, si es que existe. Aquí reside la principal diferencia con el algoritmo *ADS* de Veerapen et al.

Observando el Algoritmo 18, después de que una caja es analizada, si una nueva solución es encontrada, chequeamos si la imagen se encuentra en la parte convexa de la caja (Línea 12). De ser así, dividimos la caja en dos nuevas cajas. En caso de que la imagen de la solución esté en la parte cóncava de la caja, ésta se descarta ya que no será una solución soportada. En la Línea 5 seleccionamos la caja con mayor área en el conjunto *Boxes*.

Algoritmo *AnyAugmecon*

La versión *anytime* de *Augmecon* se llamará *AnyAugmecon*[1] y puede verse en el Algoritmo 19. Usaremos como parámetro de entrada el valor *obj*, que designa el objetivo a minimizar que emplearemos. El valor *rest* se designa para el segundo objetivo (usado como restricción en la formulación ILP). Por ejemplo, si $obj = 1$, entonces $rest = 2$ y viceversa.

Además de usar la función objetivo de comienzo y el parámetro λ como parámetros de entrada, el algoritmo fija ε como el punto medio de las coordenadas *rest* en los extremos opuestos de la caja actual (Línea 6). Entonces, una vez resuelto el subproblema, si encuentra una solución, comprueba si la imagen fue previamente generada, lo cual es equivalente a comprobar que z^* no está en el interior de la caja (Línea 11). Así, se crean dos nuevas cajas y se añaden al conjunto *Boxes*. En otro caso, solo una caja es generada, teniendo esta la mitad de área que la caja previa.

En la Figura 7.2 analizamos la caja $\left[z^1, z^2\right]$ utilizando *AnyAugmecon*. Si optimizamos f_2 (es decir, f_1 se usa como restricción), el algoritmo busca desde la línea vertical hacia la izquierda, encontrando el nuevo punto no dominado z^3. Así, se crean las cajas $\left[z^1, z^3\right]$ y $\left[z^3, z^2\right]$ (ver Figura 7.2a). Si optimizamos f_1, el algoritmo busca desde la línea horizontal

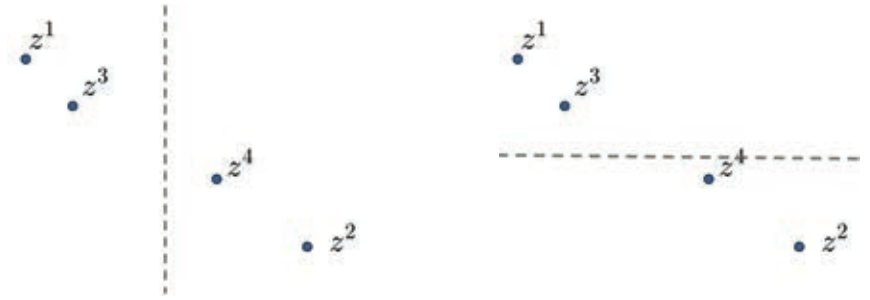
[1] Una versión preliminar de este algoritmo puede verse en Chicano et al. [22].

UNIVERSIDAD DE MÁLAGA

Algoritmo 18 *SPF*

$\left\{z^1, z^2\right\} \leftarrow$ Imágenes de las soluciones óptimas lexicográficas
$Boxes = \left\{\left[z^1, z^2\right]\right\}$
$PF = \{z^1, z^2\}$
mientras $(Boxes \neq \emptyset)$ **hacer**
  $\left[\varepsilon^1, \varepsilon^2\right] \leftarrow$ Extraer la caja de $Boxes$ con mayor área
  $\lambda_1 = \varepsilon_2^1 - \varepsilon_2^2 \quad ; \quad \lambda_2 = \varepsilon_1^2 - \varepsilon_1^1$
  $(l_1, l_2) = \left(\varepsilon_1^2 - 1, \varepsilon_2^1 - 1\right)$
  $P \equiv \text{mín}_{x \in \mathcal{X}}\{\lambda_1 f_1(x) + \lambda_2 f_2(x);\ s.a: \quad f_1(x) \leq l_1 \wedge f_2(x) \leq l_2\}$
  si (P es factible) **entonces**
    $x^* \leftarrow$ Solución óptima de P
    $z = \left(f_1\left(x^*\right), f_2\left(x^*\right)\right)$
    si $\left(\lambda_1 f_1(x^*) + \lambda_2 f_2(x^*) \leq \lambda_1 \varepsilon_1^1 + \lambda_2 \varepsilon_2^2\right)$ **entonces**
      $Boxes = Boxes \cup \left\{\left[z^1, z\right]\right\} \cup \left\{\left[z, z^2\right]\right\}$
      $PF = PF \cup \{z\}$
    fin si
  fin si
fin mientras

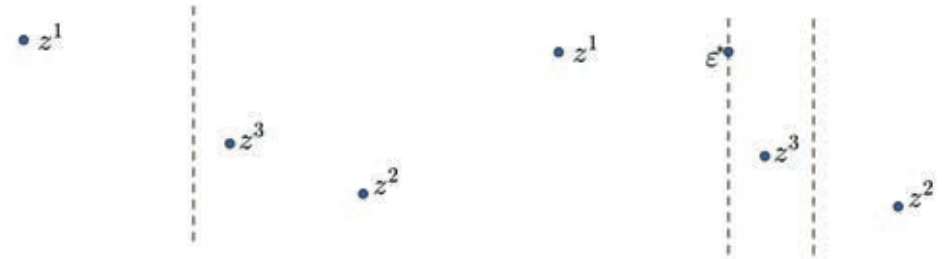
hacia abajo, obteniendo z^4 como nuevo punto no dominado, y generando las cajas $\left[z^1, z^4\right]$ y $\left[z^4, z^2\right]$ para analizar en el futuro (ver Figura 7.2b).



(a) *AnyAugmecon*$(2, \lambda)$. (b) *AnyAugmecon*$(1, \lambda)$.

FIGURA 7.2: Analizando una caja usando *AnyAugmecon*$(2, \lambda)$ y *AnyAugmecon*$(1, \lambda)$.

En la Figura 7.3, analizamos la caja $\left[z^1, z^2\right]$ usando *AnyAugmecon*$(2, \lambda)$. Ningún punto nuevo no dominado ha sido encontrado, así que se genera una nueva caja $\left[\varepsilon^*, z^2\right]$, donde $\varepsilon^* = (\varepsilon, z_2^1)$, que encontrará el punto no dominado z^3 cuando ésta sea analizada, para así volver a generar las dos nuevas cajas, $\left[\varepsilon^*, z^3\right]$ y $\left[z^3, z^2\right]$.



(a) Si exploramos desde la línea horizontal hacia la izquierda, ningún nuevo punto no dominado es encontrado.

(b) Entonces, el algoritmo busca en la región determinada por ε^* y z^2.

FIGURA 7.3: Uso de *AnyAugmecon*$(2, \lambda)$ cuando no encontramos ningún punto nuevo no dominado.

UNIVERSIDAD DE MÁLAGA

Algoritmo 19 *AnyAugmecon*$(obj,\ \lambda)$

$\{z^1,\ z^2\} \leftarrow$ Imágenes de las soluciones óptimas lexicográficas
$Boxes = \{[z^1,\ z^2]\}$
$PF = \{z^1, z^2\}$
mientras $(Boxes \neq \emptyset)$ **hacer**
  $[z^1, z^2] \leftarrow$ Extraer la caja de $Boxes$ con mayor área
  $\varepsilon = \left(z^1_{rest} + z^2_{rest}\right)/2$
  $P \equiv \min_{x\in\mathcal{X}} \{f_{obj}(x) - \lambda t\,;\ \ s.a.\ f_{rest}(x) + t \leq \varepsilon\}$
  si (P es factible) **entonces**
    $x^* \leftarrow$ Resolver P
    $z^* = (f_1(x^*),\ f_2(x^*))$
    si (z^* *está en el interior de la caja*) **entonces**
      $PF = PF \cup \{z^*\}$
      $Boxes = Boxes \cup \{[z^1,\ z^*]\} \cup \{[z^*,\ z^2]\}$
    si no
      si *(obj = 1)* **entonces**
        $Boxes = Boxes \cup \{[z^1, (z^2_1, \varepsilon)]\}$
      si no
        $Boxes = Boxes \cup \{[(\varepsilon, z^1_2), z^2]\}$
      fin si
    fin si
  fin si
fin mientras

Algoritmos *AnyTchebycheff* y *AnyHybrid*

El método aumentado de Tchebycheff ponderado que vimos en la Ecuación (3.5) y el método híbrido de la Ecuación (3.3) se han adaptado a la versión biobjetivo para ser usados como algoritmos *anytime* que resuelven el *NRP*. Estos algoritmos serán llamados *AnyTchebycheff* y *AnyHybrid*, respectivamente. La única diferencia que existe entre cada uno de ellos y su versión clásica radica en la forma en que se selecciona la siguiente caja (rectángulo) a explorar, que en este caso será aquella que tiene mayor área.

El método *AnyTchebycheff*[2] está basado en el trabajo de Dächert et al. [35]. Usa una métrica de Thebycheff ponderada y aumentada para evitar la generación de puntos débilmente no dominados. Su pseudocódigo puede verse en el Algoritmo 20.

Algoritmo 20 *AnyTchebycheff*

$\{z^1,\ z^2\} \leftarrow$ Imágenes de las soluciones óptimas lexicográficas
$Boxes = \{[z^1,\ z^2]\}$
$PF = \{z^1, z^2\}$
mientras $(Boxes \neq \emptyset)$ **hacer**
  $[\varepsilon^1, \varepsilon^2] \leftarrow$ Extraer la caja de $Boxes$ con mayor área
  $D = \left\{x \in \mathbb{R}^2 : x \in \mathcal{X} \wedge\ f_i(x) \leq \varepsilon_i^{3-i},\ i = 1, 2\right\}$
  $P \equiv \min \{\lambda + \rho \sum_{i=1}^{2} (f_i(x) - t_i) \quad s.a. \quad x \in D \quad \wedge\ \ \lambda \geq w_i(f_i(x) - t_i)\ ,\ i = 1, 2\}$
  $x^* \leftarrow$ Solución óptima de P
  $z = (f_1(x^*),\ f_2(x^*))$
  si (z no es dominado) **entonces**
    $PF = PF \cup \{z\}$
    $Boxes = Boxes \cup \{[z^1,\ z]\} \cup \{[z,\ z^2]\}$
  fin si
fin mientras

Comenzamos calculando las imágenes de los óptimos lexicográficos, que serán los extremos opuestos de la caja inicial a explorar. En este caso, los parámetros del modelo están configurados para que siempre se encuentre una solución factible. Por ello, no es necesario añadir

[2] Una versión preliminar de este algoritmo puede verse en Chicano et al. [22].

UNIVERSIDAD DE MÁLAGA

la condición de que el modelo P sea factible o no en el algoritmo. Si el algoritmo encuentra una solución cuya imagen es un extremo de la caja, quiere decir que no hay solución entre los extremos considerados, de ahí la condición que aparece en la Línea 10. En caso contrario, añadimos el nuevo punto al frente de Pareto y creamos las dos nuevas cajas a explorar. La función objetivo a minimizar es $\max\{w_1 z_1(x), w_2 z_2(x)\} + \rho|z(x)|$, donde $z(x) = f(x) - t$, y $t = (t_1, t_2)$ es el punto ideal local de la caja, es decir, si analizamos una caja $[z^1, z^2]$, entonces $t = (z_1^1, z_2^2)$. El vector w determina los pesos asociados a cada función objetivo y el valor positivo real ρ es fijo durante el algoritmo. Este parámetro toma el valor máximo posible de manera que se eviten errores numéricos por parte del resolutor. Los valores de w y ρ dependen de las coordenadas de la caja a analizar. El modelo linealizado puede verse en la Línea 7. Para más información consúltese Dächert et al. [35].

El método *AnyHybrid* combina la parametrización de los dos objetivos junto con el método *ε-constraint*. Dados dos números reales positivos, $\lambda_1, \lambda_2 > 0$, en cada iteración el algoritmo minimiza $\lambda_1 f_1(x) + \lambda_2 f_2(x)$, sujeto a las restricciones del problema, $x \in \mathcal{X}$, y las nuevas restricciones $f_i(x) \leq L_i$ para $i \in \{1, 2\}$, siendo $L = (L_1, L_2)$ un punto dado que marca la esquina superior-derecha de la región a explorar. Si el problema tiene solución óptima, ésta será eficiente. Su pseudocódigo se muestra en el Algoritmo 21.

Algoritmo 21 *AnyHybrid*

1: $\{z^1, z^2\} \leftarrow$ Imágenes de las soluciones óptimas lexicográficas
2: $Boxes = \{[z^1, z^2]\}$
3: $PF = \{z^1, z^2\}$
4: **mientras** $(Boxes \neq \emptyset)$ **hacer**
5: $[\varepsilon^1, \varepsilon^2] \leftarrow$ Extraer la caja de $Boxes$ con mayor área
6: $\lambda_1 = \varepsilon_2^1 - \varepsilon_2^2 \quad ; \quad \lambda_2 = \varepsilon_1^2 - \varepsilon_1^1$
7: $(L_1, L_2) = (\varepsilon_1^2 - 1, \varepsilon_2^1 - 1)$
8: $P \equiv \min_{x \in \mathcal{X}} \{\lambda_1 f_1(x) + \lambda_2 f_2(x); \quad s.a. \quad f_i(x) \leq L_i \ , i = 1, 2\}$
9: **si** (P es factible) **entonces**
10: $x^* \leftarrow$ Solución óptima de P
11: $z = (f_1(x^*), f_2(x^*))$
12: $PF = PF \cup \{z\}$
13: $Boxes = Boxes \cup \{[z^1, z]\} \cup \{[z, z^2]\}$
14: **fin si**
15: **fin mientras**

Comenzamos calculando las imágenes de los óptimos lexicográficos. Para cada región analizada con extremos opuestos z^1 y z^2, siendo $z_1^1 < z_1^2$, definimos $L = (L_1, L_2)$ tal que $L_1 = z_1^2 - \delta$, $L_2 = z_2^1 - \delta$, donde δ es un valor suficientemente pequeño para que no se omitan soluciones (en la práctica, para resolver el *NRP* es suficiente con tomar $\delta = 1$). Si el subproblema es infactible, la caja se descarta. En caso contrario, añadimos el nuevo punto no dominado y generamos las dos nuevas cajas a explorar. Los pesos definidos en la Línea 6 son tales que la dirección de exploración es la del vector normal a la línea que une los extremos opuestos de la caja a explorar. Se pretende así la obtención de un mayor hipervolumen durante la búsqueda de nuevas soluciones.

Algoritmos mixtos

Presentamos por último dos variantes de un algoritmo que combina dos de los métodos *anytime* descritos anteriormente: *AnyHybrid* y *AnyTchebycheff*. En los algoritmos mixtos, la selección de uno de estos dos métodos depende de la posición que ocupe la solución encontrada previamente. Dado que *AnyHybrid* es un método rápido pero no muy bueno cuando el frente de Pareto es cóncavo, y que *AnyTchebycheff* es bueno encontrando dispersión en las soluciones, surge de manera natural el planteamiento de combinar ambas variantes en un mismo algoritmo. Antes de mostrarlo, necesitamos introducir una definición.

Definición 7.2. Sean a y b dos número enteros, con $a < b$. Sea c tal que $a < c < b$. Decimos que c está *cerca de* a si $c - a < \frac{1}{4}(b - a)$. Decimos que c está *cerca de* b si $b - c < \frac{1}{4}(b - a)$.

UNIVERSIDAD DE MÁLAGA

Utilizaremos la definición anterior en esta nueva propuesta. Dependiendo de cómo de cerca esté cada componente de un punto del frente de Pareto respecto de las esquinas opuestas que definen la caja, aplicaremos un algoritmo u otro en la siguiente iteración. Supongamos que la caja actual está definida por $[z^1, z^2]$ y el nuevo punto no dominado $z = (z_1, z_2)$ está en la parte cóncava de la caja. Si z_1 está cerca de z_1^1 o está cerca de z_1^2, la caja $[z^1, z]$ será analizada usando *AnyTchebycheff*. En otro caso, usaremos *AnyHybrid*. Para la caja $[z, z^2]$, comprobamos si z_2 está cerca de z_2^1 o de z_2^2. Si este es el caso, analizaremos la caja usando *AnyTchebycheff*. En caso contrario, usaremos *AnyHybrid*. Para una mejor comprensión mostramos en la Figura 7.4 un ejemplo gráfico.

La primera versión de este algoritmo mixto se llamará *MixHT*. Su pseudocódigo puede verse en el Algoritmo 22. Para indicar qué algoritmo asociado tiene la exploración de cada caja, añadimos un nuevo campo que determina si usaremos *AnyHybrid* (denotado por '*H*') o *AnyTchebycheff* (denotado por '*T*'). En la Línea 4 puede observarse que la caja inicial se explorará utilizando el método *AnyHybrid*.

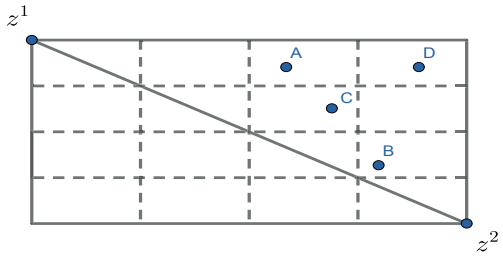


Figura 7.4: Para *MixHT*, si el nuevo punto está en la parte convexa de la caja explorada, las dos nuevas cajas usarán *AnyHybrid* cuando sean exploradas. Llamaremos **H** a *AnyHybrid* y **T** a *AnyTchebycheff*. Si el nuevo punto es A, la caja $[z^1, A]$ usará **H** y $[A, z^2]$ usará **T**. Si el nuevo punto es B, la caja $[z^1, B]$ usará **T** y $[B, z^2]$ usará **H**. Si el nuevo punto es C, las cajas $[z^1, C]$ y $[C, z^2]$ usarán **H**. Si el nuevo punto es D, las cajas $[z^1, D]$ y $[D, z^2]$ usarán **T**.

La segunda variante del algoritmo mixto, que llamaremos *MixSHT* puede verse en el Algoritmo 23. Esta variante es similar a la anterior, con la diferencia de que las soluciones soportadas son calculadas primero. La justificación de esta decisión se sustenta en que las soluciones soportadas suelen proporcionar una alta contribución al hipervolumen.

Comenzamos obteniendo el frente de Pareto soportado usando el algoritmo *SPF*. Cada vez que encontramos una solución no soportada, la guardamos en otro conjunto, que será procesado posteriormente. Por ello, necesitamos definir dos conjuntos de cajas, que llamaremos $Boxes_1$ y $Boxes_2$. Inicialmente, las imágenes de los óptimos lexicográficos se guardan en $Boxes_1$. El conjunto $Boxes_2$ será explorado una vez que $Boxes_1$ esté vacío. Mientras analizamos una caja de $Boxes_1$, si la imagen de la solución está en la parte convexa, las dos nuevas cajas se añadirán a $Boxes_1$. De esta forma en una primera fase vamos obteniendo todas las soluciones soportadas. Por otro lado, si encontramos una solución cuya imagen está en la parte cóncava de la caja, las dos nuevas cajas se guardarán en $Boxes_2$, que será procesada después de que $Boxes_1$ esté vacía. La exploración de las cajas de $Boxes_2$ se realiza usando el algoritmo *MixHT*.

7.3.3. Resultados computacionales

Comentamos en este apartado los resultados computacionales asociados a los experimentos del *NRP* biobjetivo. Describimos en primer lugar las instancias y parámetros asociados a los algoritmos. Continuamos realizando una comparativa de los algoritmos *anytime* propuestos, así como una comparativa entre los algoritmos tradicionales. Finalizaremos discutiendo los resultados, comparando los algoritmos clásicos con los *anytime*.

UNIVERSIDAD DE MÁLAGA

Algoritmo 22 *MixHT*

$\{z^1, z^2\} \leftarrow$ Imágenes de las soluciones óptimas lexicográficas
$H \leftarrow$ método *AnyHybrid*
$T \leftarrow$ método *AnyTchebycheff*
$Boxes = \{[z^1, z^2, H]\}$
$PF = \{z^1, z^2\}$
mientras $(Boxes \neq \emptyset)$ **hacer**
  $[z^1, z^2, alg] \leftarrow$ Extraer la caja de *Boxes* con mayor área
  Explorar caja $[z^1, z^2]$ usando el método *alg*
  si (*hay nueva solución óptima*) **entonces**
    $x^* \leftarrow$ Solución óptima
    $z = (f_1(x^*), f_2(x^*))$
    $PF = PF \cup \{z\}$
    si (z_1 *está cerca de* z_1^1 *o* z_1^2) **y** (z está en la parte cóncava) **entonces**
      $B_1 = [z^1, z, T]$
    si no
      $B_1 = [z^1, z, H]$
    fin si
    si (z_2 *está cerca de* z_2^1 *o* z_2^2) **y** (z está en la parte cóncava) **entonces**
      $B_2 = [z, z^2, T]$
    si no
      $B_2 = [z, z^2, H]$
    fin si
    $Boxes = Boxes \cup \{B_1\} \cup \{B_2\}$
  fin si
fin mientras

Instancias y parámetros

La ejecución de los experimentos computacionales utiliza las mismas instancias que las expuestas en el trabajo de Veerapen et al. [174]. Éstas, a su vez, están previamente descritas en [184]. Se dividen en dos grupos de datos, que denominaremos instancias *clásicas* e instancias *realistas*. El grupo de instancias clásicas está compuesto por cinco conjuntos de datos artificiales o sintéticos, nombrados desde *nrp1* hasta *nrp5*. Las instancias realistas usan repositorios de fallos (*bug repositories*) para los proyectos *open-source* de Eclipse, Gnome, y Mozilla. Se extraen cuatro subconjuntos de fallos de los tres repositorios (*nrp-e1* hasta *nrp-e4*, *nrp-g1* hasta *nrp-g4*, y *nrp-m1* hasta *nrp-m4*), donde la inicial que aparece tras el guión indica la inicial del software correspondiente. Los requisitos para las instancias realistas no tienen prerrequisitos. El número de requisitos y de clientes para cada conjunto de datos puede verse en la Tabla 7.6.

Tabla 7.6: Número de requisitos y clientes para cada conjunto de datos en el *NRP* biobjetivo.

Instancia	Requisitos	Clientes	Instancia	Requisitos	Clientes
nrp1	140	100	*nrp-g1*	2 690	445
nrp2	620	500	*nrp-g2*	2 650	315
nrp3	1 500	500	*nrp-g3*	2 512	423
nrp4	3 250	750	*nrp-g4*	2 246	294
nrp5	1 500	1 000	*nrp-m1*	4 060	768
nrp-e1	3 502	536	*nrp-m2*	4 368	617
nrp-e2	4 254	491	*nrp-m3*	3 566	765
nrp-e3	2 844	456	*nrp-m4*	3 643	568
nrp-e4	3 186	399			

Usaremos el hipervolumen [70] como indicador del espacio dominado por cada una de las soluciones obtenidas. En el caso biobjetivo representa la suma de áreas de varios rectángulos. Como punto de referencia consideraremos el punto nadir, (z_1^2, z_2^1), siendo z^1 y z^2 las imágenes

UNIVERSIDAD DE MÁLAGA

Algoritmo 23 *MixSHT*

$\{z^1, z^2\} \leftarrow$ Imágenes de las soluciones óptimas lexicográficas
$H \leftarrow$ método *AnyHybrid*; $T \leftarrow$ método *AnyTchebycheff*
$Boxes_1 = \{[z^1, z^2, H]\}$; $Boxes_2 = \emptyset$
$PF = \{z^1, z^2\}$
mientras ($Boxes_1 \neq \emptyset$) **hacer**
  $[z^1, z^2, alg] \leftarrow$ Extraer la caja de $Boxes_1$ con mayor área
  Explorar la caja $[z^1, z^2]$ usando el método *alg*
  si (*hay nueva solución óptima*) **entonces**
    $x^* \leftarrow$ Solución óptima ; $z = (f_1(x^*), f_2(x^*))$
    $PF = PF \cup \{z\}$
    si (z está en la parte convexa) **entonces**
      $Boxes_1 = Boxes_1 \cup \{[z^1, z, H]\} \cup \{[z, z^2, H]\}$
    si no
      $Boxes_2 = Boxes_2 \cup \{[z^1, z, H]\} \cup \{[z, z^2, H]\}$
    fin si
  fin si
fin mientras
mientras ($Boxes_2 \neq \emptyset$) **hacer**
  $[z^1, z^2, alg] \leftarrow$ Extraer la caja de $Boxes_2$ con mayor área
  Explorar la caja $[z^1, z^2]$ usando el método *alg*
  si (*hay nueva solución óptima*) **entonces**
    $x^* \leftarrow$ Solución óptima; $z = (f_1(x^*), f_2(x^*))$
    $PF = PF \cup \{z\}$
    si (z_1 *está cerca de* z_1^1 *o* z_1^2) **y** (z está en la parte cóncava) **entonces**
      $B_1 = [z^1, z, T]$
    si no
      $B_1 = [z^1, z, H]$
    fin si
    si (z_2 *está cerca de* z_2^1 *o* z_2^2) **y** (z está en la parte cóncava) **entonces**
      $B_2 = [z, z^2, T]$
    si no
      $B_2 = [z, z^2, H]$
    fin si
    $Boxes_2 = Boxes_2 \cup \{B_1\} \cup \{B_2\}$
  fin si
fin mientras

de los óptimos lexicográficos. Las instancias se computan usando una máquina con CPU de 1 GHz, cuatro núcleos y 16 GB de RAM. Programamos los algoritmos usando el lenguaje C[3] y CPLEX 12.6.2. Establecemos los siguientes valores para los parámetros de CPLEX: CPXPARAMEPGAP = CPXPARAMEPAGAP = CPXPARAMEPINT = 0 , al igual que usaron Veerapen et al. [174]. Todos los algoritmos se ejecutan 30 veces para cada instancia y durante 60 segundos (salvo que se indique lo contrario), tomando valores medios. El coeficiente de Pearson, σ/μ (desviación típia entre media) no supera el 2 % en el peor caso, por lo que los resultados pueden considerarse estables. Si un algoritmo encuentra alguna solución pasados los 60 segundos de ejecución, ésta es descartada.

Comparativa de los métodos *anytime*

Comenzamos mostrando en la Tabla 7.7 los resultados tras la ejecución del algoritmo *SPF*, pero sin límite de ejecución para cada una de las instancias. El número de soluciones soportadas encontradas por *SPF* corresponde al número de elementos del frente soportado, ya que es un algoritmo exacto, mientras que el algoritmo *ADS* de Veerapen et al. solo encuentra un subconjunto de dicho frente soportado. Podemos observar la gran diferencia en el porcentaje

[3]Código fuente disponible en `https://github.com/MiguelAngelDominguezRios/anytime-nrp`.

UNIVERSIDAD DE MÁLAGA

total de soluciones encontrado por cada algoritmo en relación al número total de soluciones del frente. La mayor diferencia se encuentra para *nrp5*, donde *SPF* encuentra alrededor del triple de soluciones soportadas respecto a *ADS*. También podemos observar para estas instancias que el número de soluciones soportadas es muy bajo en relación al número total de puntos del frente.

Tabla 7.7: Número de puntos del frente de Pareto y comparativa del número de soluciones soportadas encontrado por *SPF* y por *ADS*.

Instancia	$\|PF\|$	$\|SPF\|$	$\|ADS\|$	$\frac{\|SPF\|}{\|PF\|}$ %	$\frac{\|ADS\|}{\|PF\|}$ %
nrp1	465	28	27	6.0	5.8
nrp2	4 540	89	70	2.0	1.5
nrp3	6 296	246	172	3.9	2.7
nrp4	13 489	276	195	2.0	1.5
nrp5	2 898	781	264	27.0	9.1
nrp-e1	10 331	826	309	8.0	3.0
nrp-e2	10 573	680	300	6.4	2.8
nrp-e3	8 344	600	268	7.2	3.2
nrp-e4	8 303	454	257	5.5	3.1
nrp-g1	9 280	778	233	8.4	2.5
nrp-g2	6 393	341	209	5.3	3.3
nrp-g3	8 457	603	228	7.1	2.7
nrp-g4	6 171	544	201	8.8	3.3
nrp-m1	13 773	1 252	351	9.1	2.6
nrp-m2	12 933	760	329	5.9	2.5
nrp-m3	12 624	1 059	324	8.4	2.6
nrp-m4	11 547	995	295	8.6	2.6

A continuación realizamos la comparativa para los restantes algoritmos *anytime*. Usaremos dos variantes para el método *AnyAugmecom*, en función del objetivo seleccionado como principal y que aparece en la función objetivo de la parametrización. El parámetro λ considerado es el mismo que en el trabajo de Chicano et al. [22]. De esta forma, habrá un total de seis algoritmos en la comparativa. Como medida de la calidad de las soluciones utilizaremos el porcentaje total de hipervolumen obtenido y el porcentaje total de soluciones encontradas del frente.

Como podemos observar en la Tabla 7.8, cada uno de los algoritmos *anytime* es capaz de resolver la instancia *nrp1* antes de alcanzarse el tiempo límite de ejecución. Además, todos los algoritmos alcanzan más del 99 % del hipervolumen total en la mayoría de las instancias, por lo que para poder apreciar las diferencias entre ellos consideraremos tres cifras decimales. Nótese que para *nrp5*, el algoritmo *AnyAugmecon(1,λ)* no termina su ejecución debido a un desbordamiento de memoria, y sin embargo *AnyAugmecon(2,λ)* obtiene el mejor resultado computacional para esa instancia. A nivel general, vemos que todos los algoritmos se comportan como se esperaba, ya que el hipervolumen obtenido es alto en comparación con el porcentaje total de soluciones encontradas.

Los mejores resultados para el hipervolumen se reparten entre distintos algoritmos, como *AnyAugmecon(1,λ)*, *AnyAugmecon(2,λ)*, *AnyHybrid* y *MixHT*. El mejor porcentaje de soluciones obtenidas se alcanza para los algoritmos *AnyAugmecon(2,λ)*, *AnyHybrid* y *MixSHT*. La principal característica de los algoritmos *anytime* es que consiguen incrementar considerablemente su hipervolumen durante su ejecución. Observemos la gráfica mostrada en la Figura 7.5. En el eje *OX* representamos el tiempo de ejecución durante los 10 primeros segundos. En el eje *OY*, el porcentaje de hipervolumen obtenido. La misma gráfica incluye seis curvas, una para cada uno de los algoritmos utilizados. Vemos que en todos los casos se obtiene un hipervolumen razonablemente bueno en muy pocos segundos de ejecución.

Comparativa frente a los algoritmos clásicos

Antes de mostrar la ventaja que poseen los algoritmos *anytime* frente a los algoritmos clásicos, vamos a comparar estos últimos entre ellos. Ejecutamos cada uno de ellos 30 veces durante 60 segundos y tomamos valores promedios. Los resultados se muestran en la Tabla 7.9. Nuevamente, podemos ver el porcentaje total de hipervolumen obtenido para cada instancia

UNIVERSIDAD DE MÁLAGA

Tabla 7.8: Resultados para los algoritmos *anytime* (excluyendo *SPF*) para 60 segundos de tiempo límite. Los mejores porcentajes obtenidos para cada instancia se marcan en negrita.

		AnyAugmecon (1,λ)	*AnyAugmecon (2,λ)*	*AnyHybrid*	*AnyTcheby-cheff*	*MixHT*	*MixSHT*
nrp1	% Hiperv.	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**
	% PF	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**
nrp2	% Hiperv.	97.673	**98.869**	60.202	97.107	96.403	96.667
	% PF	1.0	2.0	3.6	0.7	1.1	**4.6**
nrp3	% Hiperv.	99.714	99.689	99.694	99.375	**99.740**	99.557
	% PF	4.5	4.1	5.6	2.1	5.4	**8.3**
nrp4	% Hiperv.	**98.862**	97.847	90.546	94.586	97.464	90.396
	% PF	0.6	0.3	0.6	0.1	0.4	**2.0**
nrp5	% Hiperv.	*	**99.969**	99.953	67.838	99.822	99.873
	% PF	*	**45.5**	36.8	0.1	14.6	39.6
nrp-e1	% Hiperv.	99.896	99.872	**99.898**	99.737	99.897	99.882
	% PF	6.5	5.4	7.9	2.7	7.1	**8.5**
nrp-e2	% Hiperv.	99.860	99.758	**99.882**	99.625	99.877	99.855
	% PF	4.7	2.8	5.5	2.0	5.2	**5.7**
nrp-e3	% Hiperv.	**99.931**	99.925	99.922	99.831	99.920	99.896
	% PF	11.5	10.6	**13.8**	5.3	11.3	13.0
nrp-e4	% Hiperv.	**99.911**	99.860	99.900	99.773	99.895	99.878
	% PF	9.2	6.1	**10.6**	4.1	8.8	10.4
nrp-g1	% Hiperv.	**99.955**	99.945	99.948	99.896	99.946	99.925
	% PF	15.4	13.1	**18.2**	7.8	14.5	15.8
nrp-g2	% Hiperv.	**99.956**	99.934	99.951	99.892	99.945	99.941
	% PF	19.2	14.0	**22.7**	9.6	17.5	17.3
nrp-g3	% Hiperv.	**99.963**	99.955	99.955	99.912	99.954	99.943
	% PF	19.1	16.2	**22.1**	9.7	17.3	17.4
nrp-g4	% Hiperv.	**99.969**	99.956	99.957	99.924	99.960	99.948
	% PF	27.0	20.8	**28.9**	14.0	23.6	23.2
nrp-m1	% Hiperv.	99.802	99.792	99.794	99.449	**99.809**	99.791
	% PF	2.8	2.7	3.4	1.0	3.2	**3.7**
nrp-m2	% Hiperv.	99.792	99.721	99.816	99.442	99.825	**99.825**
	% PF	2.8	2.1	3.5	1.1	3.4	**3.7**
nrp-m3	% Hiperv.	99.815	**99.843**	99.777	99.469	99.842	99.834
	% PF	3.3	3.8	4.4	1.2	4.2	**4.9**
nrp-m4	% Hiperv.	99.840	99.805	**99.869**	99.571	99.862	99.825
	% PF	4.1	3.4	5.1	1.7	4.8	**5.3**

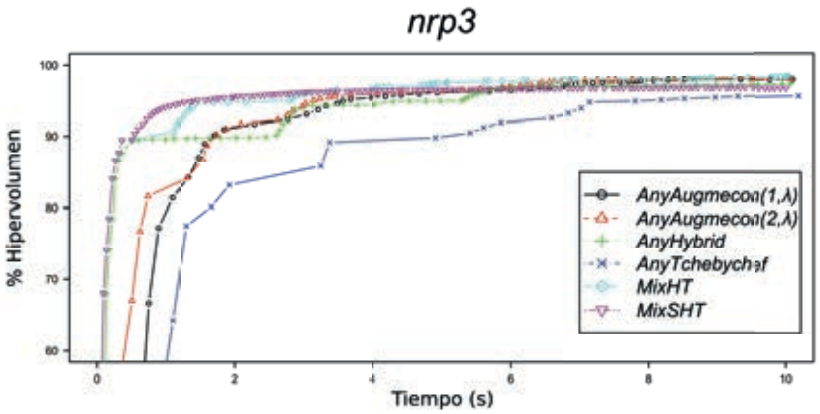


Figura 7.5: Porcentaje total de hipervolumen obtenido por seis algoritmos *anytime* durante los primeros 10 segundos de ejecución en la instancia *nrp3*.

y algoritmo, así como el porcentaje total de soluciones del frente de Pareto encontradas. Para el método ε-constraint, que denotamos por *Econst1*, usamos dos variantes en función de cual es el objetivo a considerar en la función objetivo de la parametrización. Así, *Econst1(1)* significará que minimizamos el objetivo f_1 y *Econst1(2)* que minimizamos el objetivo f_2. Análogamente hacemos lo mismo para el ε-constraint con dos iteraciones, al que llamaremos *Econst2*, y para el algoritmo *Augmecon*, que además de depender de la función objetivo que formará parte del objetivo en la parametrización, depende del valor λ, que será el mismo que se consideró para el caso de *AnyAugmecon*. Como podemos ver en la Tabla 7.9, cada algoritmo resuelve la instancia *nrp1* antes de que el tiempo límite de ejecución se alcance. Por otro lado, algunos métodos proporcionan un hipervolumen bastante pobre, sobre todo aquellos basados en ε-constraint. Como era de esperar, los algoritmos *Econst1* y *Econst2* poseen un

UNIVERSIDAD DE MÁLAGA

mayor porcentaje en cuanto al número de soluciones total obtenido, pero su hipervolumen es bajo, ya que están explorando el frente de Pareto en orden lexicográfico. En relación al hipervolumen, el algoritmo *Tchebycheff* es claramente el mejor, previsiblemente debido a la estructura de sus curvas de nivel (véase Figura 3.6). También podemos observar cómo en el mejor de los casos (exceptuando la instancia *nrp1*), el hipervolumen acumulado a los 60 segundos ronda el 70 % del total.

Tabla 7.9: Resultados para las instancias clásicas del *NRP* durante 60 segundos de tiempo límite.

		Econst1 (1)	*Econst1 (2)*	*Econst2 (1)*	*Econst2 (2)*	*Augmecon (1,λ)*	*Augmecon (2,λ)*	*EHybrid*	*Tchebycheff*
nrp1	% Hiperv.	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	% PF	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
nrp2	% Hiperv.	27.0	27.1	25.1	20.8	23.2	26.0	44.0	**67.2**
	% PF	11.2	**13.0**	10.1	9.2	9.0	12.3	9.0	8.4
nrp3	% Hiperv.	31.1	22.2	26.9	16.7	27.6	17.2	**71.3**	68.7
	% PF	**10.2**	9.3	8.2	6.3	8.5	6.6	4.3	5.3
nrp4	% Hiperv.	12.4	7.6	10.2	5.9	12.9	6.6	64.8	**67.7**
	% PF	2.5	1.4	1.8	1.0	**2.6**	1.1	1.0	1.0
nrp5	% Hiperv.	60.9	66.1	71.0	42.2	*	54.6	**74.6**	71.7
	% PF	22.4	**49.7**	30.5	29.5	*	39.4	30.0	17.7
nrp-e1	% Hiperv.	24.0	14.4	18.9	15.0	19.8	17.5	70.9	**71.8**
	% PF	**7.7**	3.4	5.5	3.6	5.9	4.4	2.2	2.9
nrp-e2	% Hiperv.	19.4	11.6	14.4	12.5	17.4	14.7	70.1	**72.3**
	% PF	**5.7**	1.9	3.5	2.1	4.8	2.6	1.7	2.0
nrp-e3	% Hiperv.	35.6	28.0	29.3	24.0	26.8	24.7	70.7	**72.1**
	% PF	**13.2**	9.4	10.1	7.2	8.9	7.6	4.6	5.7
nrp-e4	% Hiperv.	34.6	17.6	25.6	20.8	27.6	21.5	70.5	**71.8**
	% PF	**13.5**	4.4	8.5	5.6	9.5	5.9	3.7	4.8
nrp-g1	% Hiperv.	38.2	44.7	27.9	36.6	27.4	38.1	68.5	**72.9**
	% PF	**18.4**	16.0	12.1	10.8	11.8	11.6	7.8	8.8
nrp-g2	% Hiperv.	42.8	50.1	29.7	47.6	32.9	45.1	69.7	**73.9**
	% PF	**25.6**	13.7	15.8	12.5	18.3	11.5	8.1	8.9
nrp-g3	% Hiperv.	42.7	48.4	31.5	40.5	32.4	41.6	70.5	**72.8**
	% PF	**20.8**	17.3	13.0	12.7	13.6	13.4	9.0	10.0
nrp-g4	% Hiperv.	54.2	63.8	37.8	54.1	42.6	54.2	70.5	**73.4**
	% PF	**32.5**	26.4	18.5	18.6	22.1	18.7	12.9	14.4
nrp-m1	% Hiperv.	11.9	9.2	8.8	8.9	11.0	12.3	67.9	**71.8**
	% PF	**3.4**	1.5	2.3	1.5	3.1	2.3	0.9	1.5
nrp-m2	% Hiperv.	12.0	10.4	8.4	10.4	10.8	12.3	67.2	**72.1**
	% PF	**3.7**	1.3	2.3	1.3	3.2	1.8	0.8	1.2
nrp-m3	% Hiperv.	14.4	10.7	11.7	9.4	13.1	12.1	68.0	**71.2**
	% PF	**4.0**	2.9	3.0	2.4	3.5	3.5	1.4	2.1
nrp-m4	% Hiperv.	16.4	12.5	11.2	12.7	13.9	14.7	68.0	**71.7**
	% PF	**5.5**	2.3	3.4	2.4	4.4	3.1	1.5	1.9

Para finalizar la comparativa, comparamos para cada instancia al mejor de los algoritmos clásicos con el peor de los algoritmos *anytime*, en términos de porcentaje de hipervolumen obtenido. Omitiendo el análisis de la instancia *nrp1* (en cuyo caso todos los algoritmos son idénticos), apreciamos en la Tabla 7.10 que para todas las instancias existe una mejora de más del 22 % a favor de los métodos *anytime*, excepto para la instancias *nrp2* y *nrp5*, donde el mejor algoritmo clásico supera al peor *anytime*, aunque no supera al mejor *anytime*. Resulta interesante observar que la mayoría de los mejores resultados para los algoritmos clásicos se alcanzan para el método *Tchebycheff* y la mayoría de los peores resultados para los *anytime* se alcanzan en el método *AnyTchebycheff*. Este hecho sugiere que el método aumentado de Tchebycheff se comporta mejor cuando es usado como algoritmo *anytime*.

En las Figuras 7.6, 7.7, 7.8, 7.9 y 7.10 podemos observar el progreso del porcentaje de hipervolumen en función del tiempo para cuatro instancias, considerando el peor algoritmo clásico (WC), el mejor algoritmo clásico (BC), el peor algoritmo *anytime* (WA) y el mejor algoritmo *anytime* (BA). Puede verse que el progreso en hipervolumen es claramente mayor en los algoritmos *anytime* conforme el tiempo avanza.

Después de analizar los resultados computacionales para el *NRP* biobjetivo, aplicamos el test de Friedman para contrastar si existe diferencia significativa entre los métodos *anytime* en cuanto al hipervolumen medio obtenido. El resultado muestra un p-valor de $3.187 \cdot 10^{-7}$, lo que sugiere una fuerte evidencia de que el rendimiento de los algoritmos es diferente. Para averiguar dónde se encuentran esas diferencias realizamos un test *post-hoc* usando el test de Nemenyi de comparaciones múltiples. Los resultados de este test pueden verse en la Tabla 7.11.

Como podemos observar los únicos contrastes significativos (a nivel de significación del 5 %) son aquellos que comparan *AnyTchebycheff* con el resto, excluyendo la comparativa con

UNIVERSIDAD DE MÁLAGA

Tabla 7.10: Porcentaje aproximado de diferencia entre porcentajes de hipervolumen obtenido entre el mejor algoritmo clásico y el peor algoritmo *anytime*. Las diferencias positivas indican una ventaja de estos últimos.

Instancia	Mejor clásico	% Hiperv	Peor *anytime*	% Hiperv	% Diferencia
nrp2	*Tchebycheff*	67.2	*AnyHybrid*	60.2	-7.0
nrp3	*EHybrid*	71.3	*AnyTchebycheff*	99.4	**28.1**
nrp4	*Tchebycheff*	67.7	*MixSHT*	90.4	**22.7**
nrp5	*EHybrid*	74.6	*AnyTchebycheff*	67.8	-6.8
nrp-e1	*Tchebycheff*	71.8	*AnyTchebycheff*	99.7	**27.9**
nrp-e2	*Tchebycheff*	72.3	*AnyTchebycheff*	99.6	**27.3**
nrp-e3	*Tchebycheff*	72.1	*AnyTchebycheff*	99.8	**27.7**
nrp-e4	*Tchebycheff*	71.8	*AnyTchebycheff*	99.8	**28.0**
nrp-g1	*Tchebycheff*	72.9	*AnyTchebycheff*	99.9	**27.0**
nrp-g2	*Tchebycheff*	73.9	*AnyTchebycheff*	99.9	**26.0**
nrp-g3	*Tchebycheff*	72.8	*AnyTchebycheff*	99.9	**27.1**
nrp-g4	*Tchebycheff*	73.4	*AnyTchebycheff*	99.9	**26.5**
nrp-m1	*Tchebycheff*	71.8	*AnyTchebycheff*	99.4	**27.6**
nrp-m2	*Tchebycheff*	72.1	*AnyTchebycheff*	99.4	**27.3**
nrp-m3	*Tchebycheff*	71.2	*AnyTchebycheff*	99.5	**28.3**
nrp-m4	*Tchebycheff*	71.7	*AnyTchebycheff*	99.6	**27.9**

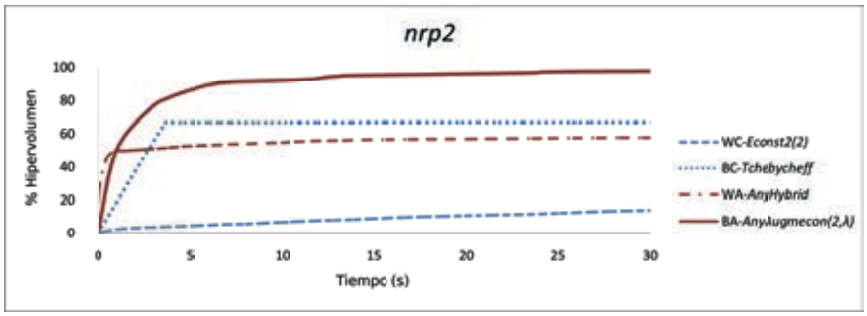


Figura 7.6: Comparando el peor algoritmo clásico (WC), mejor algoritmo clásico (BC), peor algoritmo *anytime* (WA) y mejor algoritmo *anytime* (BA), para la instancia *nrp2*.

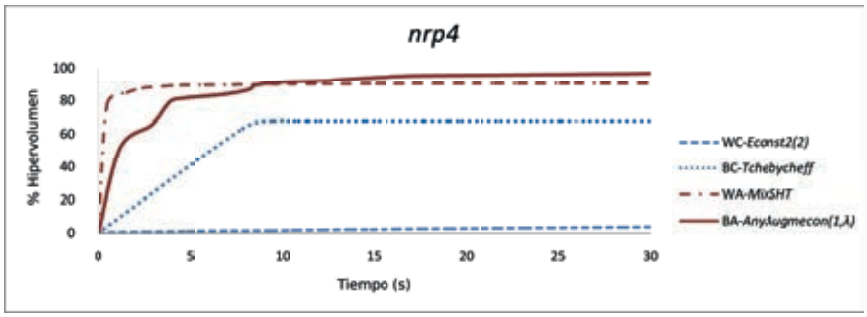


Figura 7.7: Comparando el peor algoritmo clásico (WC), mejor algoritmo clásico (BC), peor algoritmo *anytime* (WA) y mejor algoritmo *anytime* (BA), para la instancia *nrp4*.

Tabla 7.11: Test *post-hoc* de Nemenyi usando comparaciones múltiples.

	AnyAugmecon(2,λ)	*AnyHybrid*	*AnyTchebycheff*	*MixHT*	*MixSHT*
AnyAugmecon(1,λ)	0.15927	0.82732	1.2e-06	0.96213	0.00989
AnyAugmecon(2,λ)	-	0.85074	0.03463	0.62413	0.92569
AnyHybrid	-	-	0.00048	0.99883	0.26315
AnyTchebycheff	-	-	-	8.3e-05	0.34184
MixHT	-	-	-	-	0.11337

MixSHT. También el par *AnyAugmecon(1,λ)* – *MixSHT* proporciona una diferencia significativa. Utilizamos el test de Wilcoxon de los rangos signados para comparar todas las parejas de algoritmos y concluimos que *AnyAugmecon(1,λ)*, *AnyAugmecon(2,λ)*, *AnyHybrid* y *MixHT* son mejores que *AnyTchebycheff*, pero no hay diferencia significativa entre *AnyAugmecon(1,λ)* y *MixSHT*. En conclusión, podemos afirmar que para este conjunto de instancias, el algoritmo *AnyThebycheff* posee un rendimiento menor que el resto, pero para los demás, no hay un

UNIVERSIDAD DE MÁLAGA

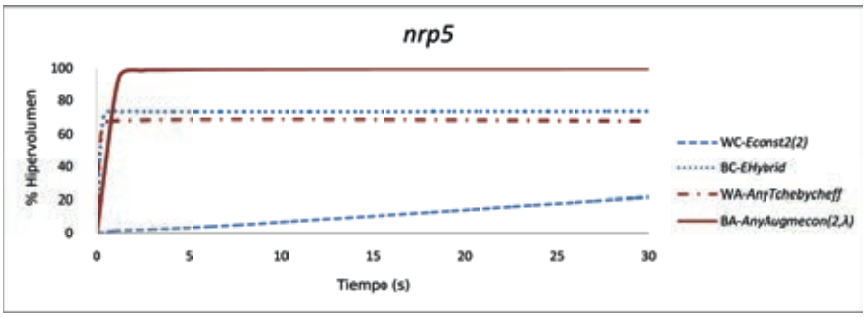


FIGURA 7.8: Comparando el peor algoritmo clásico (WC), mejor algoritmo clásico (BC), peor algoritmo *anytime* (WA) y mejor algoritmo *anytime* (BA), para la instancia *nrp5*.

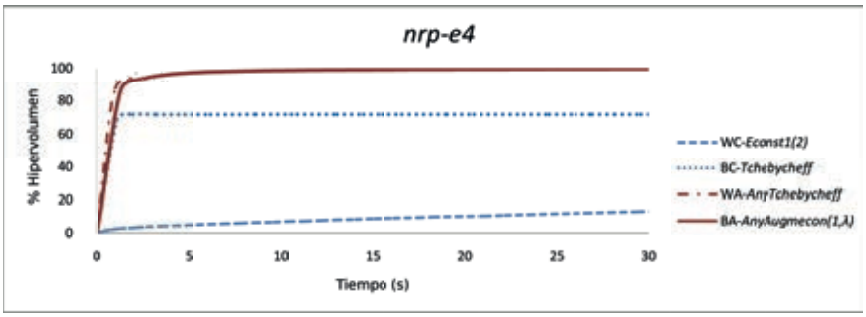


FIGURA 7.9: Comparando el peor algoritmo clásico (WC), mejor algoritmo clásico (BC), peor algoritmo *anytime* (WA) y mejor algoritmo *anytime* (BA), para la instancia *nrp-e4*.

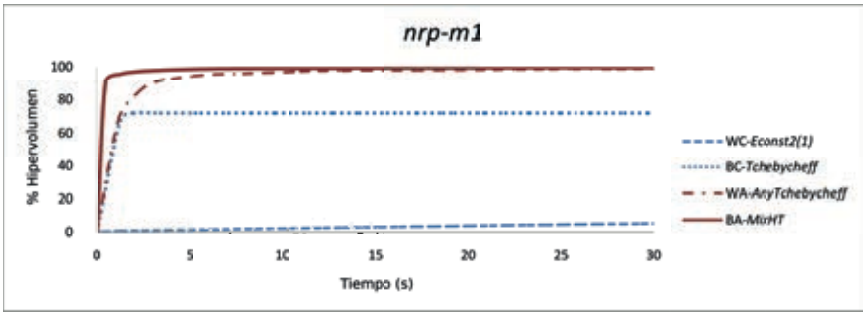


FIGURA 7.10: Comparando el peor algoritmo clásico (WC), mejor algoritmo clásico (BC), peor algoritmo *anytime* (WA) y mejor algoritmo *anytime* (BA), para la instancia *nrp-m1*.

claro ganador, al menos estadísticamente hablando.

7.4. Conclusiones

El presente capítulo está dedicado al problema de selección de requisitos, tanto en su versión monoobjetivo como biobjetivo. Tras la descripción del problema, se presenta un algoritmo híbrido (con dos variantes) que resuelve el problema monoobjetivo de manera aproximada y que usa la metaheurística de Blum et al. [12], conocida como CMSA. En la versión biobjetivo se presentan varios algoritmos *anytime* que resuelven el problema *NRP* biobjetivo de manera exacta, basados la mayoría de ellos en los algoritmos clásicos de resolución para problemas biobjetivo. Una vez comentados los análisis de los resultados en ambos enfoques, a continuación exponemos, de manera separada, las conclusiones obtenidas en cada uno de los casos.

UNIVERSIDAD DE MÁLAGA

Conclusiones y líneas futuras de investigación para las heurísticas monoobjetivo

Después de aplicar los dos algoritmos basados en CMSA a instancias aleatorias con gran número de requisitos, clientes y dependencias, se puede confirmar estadísticamente (a un nivel de significancia del 5 %) que el uso de la heurística *cmsa_ s* es la más adecuada a la hora de resolver instancias grandes del *NRP*.

Una línea futura de investigación sobre este trabajo podría ser aplicar el paquete IRACE [121], que permitiría obtener una buena configuración para los parámetros n_a y age_{max}, dentro de un rango preestablecido. También se podría estudiar más a fondo el problema, con un rango más amplio de requisitos, clientes y dependencias, y estimar a partir de qué valores de estos parámetros se obtienen mejores resultados para un método u otro. Esto se podría conseguir también usando el paquete IRACE y se matizaría en un algoritmo mediante combinación de los anteriores que sirviese como método híbrido que determina cuál es el mejor método a utilizar en función de los parámetros de entrada. Otra línea de investigación comprende la posibilidad de estudiar el comportamiento del algoritmo si se añade la opción de exclusividad entre requisitos.

También se plantea generalizar el método CMSA para el caso multiobjetivo y aplicarlo inicialmente al *NRP* biobjetivo. De hecho, aunque se ha trabajado en esta vía, no se han obtenido unos resultados iniciales satisfactorios, debido principalmente a dos problemas. El primero es que la elección de componentes como aquellos entes más válidos a formar parte de una solución, no viene a ser una tarea fácil para un problema biobjetivo, donde el espacio de soluciones es bidimensional y el comportamiento de las variables de decisión parece variar considerablemente en distintas zonas del espacio objetivo. El segundo problema encontrado es la forma de aplicar la función *ApplyExactSolver* para el caso biobjetivo, ya que un barrido completo por el espacio objetivo en cada iteración no solo consume demasiado tiempo de ejecución, sino que además las soluciones obtenidas suelen estar muy lejos de los valores óptimos, por lo que el prototipo de CMSA biobjetivo resulta poco eficaz. Se plantea la posibilidad de subdividir el espacio objetivo en varias zonas y aplicar CMSA a cada una de ellas.

Conclusiones para los algoritmos *anytime* biobjetivo

Para este problema se han propuesto seis algoritmos exactos, todos ellos de carácter *anytime*. Uno de ellos sólo calcula el frente de Pareto soportado. El resto es capaz de obtener el frente completo, si se supone de suficiente tiempo de ejecución. Consideramos que en el campo de la Ingeniería del Software, al resolver este problema sería suficiente con que el decisor tuviese que elegir ante un grupo reducido de alternativas, cuestión que se consigue en pocos segundos de ejecución, como se observa en los resultados computacionales. Además, en comparación con los algoritmos que resuelven este problema utilizando metaheurísticas, cabe destacar que en este caso las soluciones obtenidas son todas pertenecientes al frente de Pareto, ya que los algoritmos son exactos.

En conclusión, podemos afirmar que para este conjunto de instancias, el algoritmo *AnyTchebycheff* posee un rendimiento menor que el resto, pero para los demás, no hay un claro ganador, al menos estadísticamente hablando.

UNIVERSIDAD DE MÁLAGA

Parte IV

Conclusiones y trabajo futuro

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

La presente tesis doctoral realiza un estudio sobre algoritmos que resuelven problemas de optimización combinatoria multiobjetivo. En ocasiones se realiza una hibridación de estos algoritmos exactos con metaheurísticas, para mejorar su rendimiento. Se busca que dichos métodos sean *anytime*, es decir, que las soluciones se encuentren bien dispersas por el espacio objetivo durante cualquier momento de la ejecución.

Contribuciones y trabajo futuro

El bloque de fundamentos contiene dos capítulos que establecen el marco teórico de esta tesis. Por un lado se realiza una pequeña introducción a la optimización combinatoria, y por el otro se definen las distintas técnicas que resuelven los problemas de optimización multiobjetivo, centrándose en los modelos exactos.

La primera propuesta algorítmica, *TPA*, desarrollada en el Capítulo 4, establece un método exacto que resuelve cualquier problema combinatorio multiobjetivo. Con suficiente tiempo de ejecución, el algoritmo es capaz de identificar el frente de Pareto completo y un punto eficiente asociado a cada punto no dominado. Más aún, si el algoritmo corta su ejecución en un instante cualquiera, los resultados parciales obtenidos muestran una buena dispersión por el espacio objetivo, acorde a las métricas ONVG, HV, Δ^* y ε_+. Por ello, se puede considerar que *TPA* es un algoritmo *anytime*. El método propuesto se ha comparado con otros dos algoritmos existentes en la literatura: SBA, del trabajo de Ceyhan et al. [18], y el propuesto por Holzmann y Smith [91]. Estos dos métodos se pueden considerar también buenos algoritmos *anytime*. El análisis de los resultados computacionales utiliza 480 instancias durante un tiempo máximo de ejecución de 900 segundos. Se han contrastado estadísticamente las diferencias entre los tres métodos, mostrándose que *TPA* resulta tener mejor rendimiento computacional en la mayoría de los casos. En algunas ocasiones los resultados no han resultado concluyentes y en algunos casos concretos, el algoritmo SBA de Ceyhan et al. ha resultado tener el mejor rendimiento en cuanto a la métrica Δ^*. Al margen de la realización de investigación futura con más instancias que engloben una mayor variedad de problemas combinatorios, podemos concluir que el método se comporta bien para aquellos problemas de tamaño mediano o en los que no es muy costoso obtener una solución (en cuanto a tiempo de ejecución). Es evidente que cuanto más duro sea el problema, peor rendimiento tendrá el algoritmo, pero para tamaños intermedios, el hecho de obtener soluciones exactas bien distribuidas puede resultar muy beneficioso en comparación con los algoritmos aproximados. Como trabajo futuro incluimos la extensión a otras instancias para tratar de determinar para qué conjunto de problemas nuestro método funciona bien. También se plantea aplicar el método *TPA* a problemas multiobjetivo en la industria, donde un conjunto pequeño de soluciones bien dispersas puede ser requerido a corto plazo por el decisor. Algunas de las ideas introducidas en este capítulo también se aplican a métodos heurísticos, realizando una hibridación con métodos aproximados, como puede apreciarse en el contenido del Capítulo 5.

En el Capítulo 5 se establecen dos algoritmos híbridos que resuelven problemas combinatorios multiobjetivo. El primero de ellos, *MOFeLS*, está restringido a problemas con variables binarias. Es un algoritmo completamente aproximado. Utiliza un resolutor ILP únicamente para obtener soluciones factibles, y después realiza una búsqueda local para obtener un óptimo local. A pesar de ser aproximado, los resultados obtenidos en los experimentos computacionales han mostrado una mejora notable en cuanto a rendimiento tras compararlo con las metaheurísticas NSGA-II, SPEA2 y MOEA/D en 28 instancias de datos. La principal ventaja de este método es que puede trabajar incluso con restricciones de igualdad, lo cual es un *handicap* cuando se usan metaheurísticas que utilizan la inversión de bits en el proceso de mutación. Además, sus soluciones se encuentran razonablemente cerca de los puntos del frente de Pareto, como muestran los valores de la métrica GD en los resultados computacionales, en tan solo 30 segundos de ejecución. Los valores de HV y de IGD también indican que la dispersión de las soluciones por el espacio objetivo es buena, por lo que el algoritmo puede considerarse *anytime*.

El segundo algoritmo del Capítulo 5, *HBOXES*, surgió como extensión de *MOFeLS*, aunque en este caso su resolución es válida para cualquier problema combinatorio multiobjetivo. Partiendo de la hipótesis inicial de obtener un mayor rendimiento del resolutor ILP, se pretende conseguir una mayor calidad en las soluciones usando únicamente parámetros internos

UNIVERSIDAD DE MÁLAGA

de dicho resolutor, sin necesidad de realizar una búsqueda local. Esta segunda propuesta es también aproximada, aunque con la configuración adecuada de sus parámetros de entrada podría considerarse un algoritmo exacto. Los resultados computacionales han sido diversos. Se extiende la experimentación a 911 instancias, procedentes de diferentes tipos de problemas combinatorios multiobjetivo, y se compara el rendimiento con los algoritmos *MOFeLS* y *TPA*. En la mayoría de los casos, los resultados de *HBOXES* se ven superados por alguno de los otros dos métodos. No obstante, las ideas desarrolladas en la implementación de este algoritmo deja abierta una multitud de combinaciones posibles que podrían generar múltiples algoritmos híbridos distintos. Así, concluimos que existe aún margen de mejora para estos resultados, que se propone como investigación futura.

Con respecto a la aplicación de algoritmos a problemas de Ingeniería del Software, el primero de los capítulos se centra en el problema de Partición de Módulos Software. Se proponen dos algoritmos que lo resuelven de manera exacta. El primero utiliza una técnica de ramificación y poda, aunque en la práctica se comporta casi como un algoritmo enumerativo. El segundo resuelve el problema mediante un modelo de programación lineal monoobjetivo, tras comprobar que su formulación biobjetivo no ofrecía buenos resultados. El algoritmo linealiza un modelo matemático inicial, y es capaz de resolverlo de manera exacta con una única llamada al resolutor. El inconveniente encontrado en este caso ha sido el enorme esfuerzo computacional que posee este problema cuando crece el número de nodos. La complejidad en su resolución crece de una forma más que exponencial, por lo que en la práctica resulta inviable resolverlo de manera exacta. No obstante, se han intentado añadir restricciones que permitan podar ramas en la exploración, y cuyos resultados puedan ser competitivos a la hora de obtener el óptimo. Todavía los resultados obtenidos están lejos de los conseguidos por los métodos heurísticos. Como trabajo futuro se propone seguir investigando en este problema. Se espera obtener un amplio margen de mejora, pues su formulación actual permite introducir muchas variantes en este modelo matemático de programación lineal.

El último capítulo muestra varios algoritmos que resuelven el problema de Selección de Requisitos. Este problema ha sido abordado en sus dos variantes: monoobjetivo y biobjetivo. La versión monobjetivo se resuelve usando dos algoritmos que se basan en una técnica híbrida conocida como CMSA. Los experimentos computacionales se realizan sobre instancias generadas aleatoriamente con un gran número de clientes y de requisitos. Se comparan las dos propuestas con la aplicación de un método exacto. Los resultados muestran que para los casos en que el método exacto es mejor, los algoritmos híbridos proporcionan un valor objetivo relativamente cercano al del método exacto. Sin embargo, en las instancias más duras en las que el algoritmo exacto proporciona una solución pobre, los algoritmos híbridos proporcionan un valor objetivo mucho mayor. Como investigación futura se ha iniciado un estudio de aplicación de la técnica CMSA para el caso biobjetivo, no resultando una cuestión baladí, y siendo complicado encajar esta técnica a la hora de obtener buenos resultados para este tipo de problema.

Para la versión biobjetivo del problema de Selección de Requisitos, se diseñan seis algoritmos que lo resuelven de manera exacta (aunque uno de ellos solo calcula el frente de Pareto soportado). Las soluciones obtenidas están bien distribuidas por el espacio objetivo, así que pueden considerarse buenos métodos *anytime*. Tres de los seis algoritmos propuestos son nuevas contribuciones. El algoritmo *SPF* calcula el frente soportado de Pareto. El algoritmo *AnyHybrid* combina una escalarizacion en las funciones objetivo junto con el metodo ε-constraint. Los algoritmos *MixHT* y *MixSHT* combinan otros métodos *anytime* existentes para obtener una mejor dispersión de las soluciones en menor tiempo. Se comparan estos cuatro algoritmos junto con otros dos existentes, basados en el metodo ε-constraint aumentado (*AnyAugmecon*) y en la metrica de Tchebycheff (*AnyTchebycheff*). Los algoritmos se comparan frente a los métodos exactos clásicos, y se concluye que los *anytime* funcionan mejor que los exactos para la mayoría de las instancias consideradas. Como trabajo futuro se plantea aplicar estos mismos métodos *anytime* a otros problemas de Ingeniería del Software. Por otro lado, existen otras variantes del problema donde el valor de satisfacción del cliente no tiene por qué depender de que se satisfagan todos sus requisitos. En estos casos su formulación matemática resulta más compleja, y se plantea también como trabajo futuro. Otras líneas de investigación incluyen instancias más duras, pudiendo extenderse también el trabajo a problemas con más de dos objetivos.

UNIVERSIDAD DE MÁLAGA

Limitaciones

Los algoritmos *TPA*, *MOFeLS* y *HBOXES* desarrollados en los Capítulos 4 y 5 de la Parte II, resuelven bien de manera exacta, o bien aproximadamente, problemas combinatorios multiobjetivo. En la Parte III se establecen aplicaciones a problemas de Ingeniería del Software. Sin embargo, en estos problemas concretos no se han aplicado los métodos expuestos en la Parte II. Esto es debido a que la formulación de los problemas de Ingeniería del Software considerados usa uno o dos objetivos a optimizar. Las principales propuestas algorítmicas de esta tesis han mostrado poseer un mejor rendimiento en los casos en los que $p > 2$. Así, si bien estos métodos se comportan bien sobre todo para el caso de más de dos objetivos, para estos problemas concretos hemos necesitado diseñar nuevos algoritmos que se comporten mejor. En cierta manera, todos los métodos poseen una característica en común: se van explorando zonas (o cajas) del espacio objetivo en busca de nuevas soluciones. La exploración se realiza de manera que dichas soluciones se encuentren bien dispersas. Incluso en algunas ocasiones se ha constatado que el diseño monoobjetivo ofrece mejores resultados que en el caso biobjetivo.

La Sección 7.3 establece seis algoritmos que resuelven el *NRP* en su versión biobjetivo. Dado que los resultados computacionales no permiten confirmar estadísticamente que un método sea mejor que el resto, esto podría interpretarse también como una limitación en el sentido de que el decisor podría no saber cuál de los algoritmos debería usar para cada problema. No obstante, los resultados expuestos en esta sección pretenden demostrar que el uso de cualquiera de los métodos *anytime* mejora considerablemente al uso de los algoritmos exactos clásicos. Como investigación futura se propone el uso de estos algoritmos en otros problemas de la Ingeniería del Software, donde tal vez los resultados computacionales nos permitan discriminar en qué tipo de problemas puede funcionar mejor un método *anytime* que otro.

Conclusión

Como conclusión final, destacamos la utilidad del uso de la hibridación en el campo de la optimización combinatoria, incluyendo el caso multiobjetivo. Mientras no se demuestre que $\mathcal{P} = \mathcal{NP}$, siempre resultará conveniente el uso métodos que aprovechen las ventajas de los resolutores ILP junto con las que poseen las metaheurísticas a la hora de obtener buenas aproximaciones en un corto espacio de tiempo. El posible trabajo futuro en este ámbito parece no estar acotado. La belleza y la satisfacción que proporciona realizar investigación operativa es la que nos impulsa a seguir profundizando en este ámbito.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Parte V

Apéndices

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Apéndice A

Relación de publicaciones que sustentan la tesis

En este apéndice mostramos la relación de publicaciones en revistas científicas que sustentan esta tesis doctoral. Así mismo, mostramos aquellos trabajos que han sido publicados en congresos nacionales o internacionales, y cuyo contenido ha sido también desarrollado en esta tesis.

A.1. Publicaciones en revistas

La relación de publicaciones en revistas es la siguiente:

- Revista: Journal of Systems and Software (Elsevier).

  Título: Efficient anytime algorithms to solve the bi-objective Next Release Problem.

  Año de publicación: 2019.

  Enlace en la bibliografía: [49].

  Comentario: Este trabajo corresponde al desarrollo de los algoritmos *anytime* que resuelven el probolema *NRP* biobjetivo y que se muestran en el Capítulo 7.

- Revista: Information Sciences (Elsevier).

  Título: Effective anytime algorithm for multiobjective combinatorial optimization problems.

  Año de publicación: 2021.

  Enlace en la bibliografía: [50].

  Comentario: Corresponde al algoritmo *TPA* del Capítulo 4.

A.2. Publicaciones en congresos nacionales e internacionales

Mostramos ahora, por orden cronológico, la relación de trabajos publicados en congresos o conferencias, tanto nacionales como internacionales.

- Congreso nacional: XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD).

  Lugar: Salamanca (España) (15 de septiembre, 2016).

  Título: Dos estrategias de búsqueda *anytime* basadas en programación lineal entera para resolver el problema de selección de requisitos.

  Referencia bibliográfica: [22].

  Comentario: Participación como segundo autor. Corresponde a la descripción de los Algoritmos 19 y 20 del Capítulo 7.

UNIVERSIDAD DE MÁLAGA

- Congreso nacional: XXII Jornadas de Ingeniería del Software y Bases de Datos (JISBD).

  Lugar: Santa Cruz de Tenerife (España) (19 al 21 de julio, 2017).

  Título: Hacia un Algoritmo Exacto para Resolver el Problema de Módulos Software.

  Comentario: Participación como primer autor y como ponente. Corresponde al algoritmo de ramificación y poda descrito en los Algoritmos 11 y 12 del Capítulo 6.

- Conferencia internacional: EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization.

  Título: Anytime Algorithms for Robust Multi-Objective Next Release Problem.

  Lugar: Bologna (Italy) (25^{th} to 27^{th} of June, 2018).

  Comentario: Participación como primer autor. Relacionado con el Capítulo 7 de esta tesis.

- Conferencia internacional: EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization.

  Título: Solving Multi-Objective Hub Location Problems by Hybrid Algorithms.

  Lugar: Bologna (Italy) (25^{th} to 27^{th} of June, 2018).

  Comentario: Participación como segundo autor. Trabajo secundario no directamente relacionado con el contenido de esta tesis.

- Congreso nacional: XVIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA).

  Lugar: Granada (España) (23 al 26 de octubre, 2018).

  Título: Uso de CMSA para resolver el problema de selección de requisitos.

  Comentario: Participación como primer autor. Reseñado en la Sección 7.2, correspondiendo con los algoritmos híbridos utilizados para resolver el problema *NRP* en su versión monoobjetivo.

- Congreso internacional (*online*): 24th International Conference on the Applications of Evolutionary Computation (EVOAPPS).

  Título: Improving Search Efficiency and Diversity of Solutions in Multiobjective Binary Optimization by Using Metaheuristics plus Integer Linear Programming.

  Lugar: Seville (Spain) (7^{th} to 9^{th} of April, 2021).

  Comentario: Participación como primer autor y como ponente. Es el algoritmo *MOFeLS* del Capítulo 5.

UNIVERSIDAD DE MÁLAGA

Apéndice B

Summary of this thesis in English

A combinatorial optimization problem is said to be solved when we find a solution among a finite number of them, which minimizes or maximizes a specific objective function. This objective function assigns a real number to any of its solutions. Combinatorial optimization problems have applications in almost all disciplines in society. Solving them is important when an organization or entity aims to minimize costs or to increase benefits.

In some cases, there exists more than one objective function to optimize. These problems are called multiobjective [58]. If some of the objectives are in conflict, there is no solution that optimize all the objectives, in general. Thus, the concept of Pareto dominance is defined, in order to establish a partial order between solutions. In a general way, a solution x is said to dominate another solution y when the objective vector associated to x is better than the objective vector associated to y in some component, and it is not worse for the rest. From the decision maker's point of view, solution y can be discarded. For these problems it is interesting to obtain a set of solutions that are not dominated by others, usually known as Pareto optimal set. The image of the Pareto optimal set is known as Pareto front. The knowledge of the Pareto front is useful because the decision maker can eventually choose one (or more) of them, according to his/her preferences, taking into account that all of them may be considered as 'optimal'.

Solving a combinatorial optimization problem is in general $\mathcal{NP}$-*Hard* [74]. No algorithm that solves a $\mathcal{NP}$-*Hard* problem in polynomial time is known. In fact, most of the experts in this field conjecture that this algorithm might not exist. In multiobjective combinatorial optimization problems, the number of solutions of the Pareto front may well increase exponentially with the size of the problem, so in this case, it is still less probably to find such an efficient algorithm.

We can classify optimization problems in two categories: exact and approximated. Exact algorithms find the optimal solutions but they are time consuming. On the other hand, approximated algorithms are fast in finding approximated solutions although sometimes the solutions have no good quality at all. Inside the approximated methods, we distinguish between *ad hoc* heuristics, which are designed for specific problems, and metaheuristics, which are able to work on any optimization problem. Analyzing the advantages and disadvantages of exact and approximated algorithms, it is reasonable to think that combining both types in a precise way we can obtain a more beneficial tool, if we exploit appropriately the advantages of them. The main topic of this thesis is to develop algorithms which solve multiobjective combinatorial optimization problems with good spread of the solutions over the objective space. The term 'multiobjective' is considered in a general way, so that we also present in this manuscript some single-objective mathematical models. We use hybridization of exact methods with metaheuristics in some of the exposed techniques.

B.1. Organization of the thesis

Chapter 1 defines the aims and the methodology of this thesis. After that, a summary of the contributions is exposed. We finish the chapter indicating how this thesis is organized. The rest of the thesis is structured in five parts. The first part is dedicated to the theoretical foundations. It has two chapters, where the concepts related to combinatorial optimization problems are defined, and different techniques to solve those problem are described, including single-objective problems as a particular case. The second part establishes the algorithmic

UNIVERSIDAD DE MÁLAGA

proposals, which constitute the contributions of this work. It is divided also in two chapters. Three new algorithms are defined. The first one develops a new exact method to solve any multiobjective combinatorial optimization problem, called *TPA*. The second one is a heuristic algorithm, called *MOFeLS*, restricted to binary linear multiobjective problems. The last one, is a hybrid method, called *HBOXES*, similar to *MOFeLS*, although it works for any multiobjective combinatorial problem and can be considered as an exact method if we configure the input parameters properly. The third part of the thesis contains applications to Software Engineering problems. These are the Software Module Clustering (*SMC*) and the Next Release Problem (*NRP*). Several methods, exact and approximated, are described across the two chapters this part contains. The fourth part shows the conclusions, the limitations, and outlines the future work. The last part contains two Appendices. The first one enumerates the publications that support the thesis, and the second part is this summary of the thesis in English.

Next, we briefly detail the contents of the chapters.

- **Part I. Foundations**
  - Chapter 2 introduces the main concepts used in this thesis, beginning with the definition of algorithmic complexity, defining the classes of problems as well as the concept of combinatorial optimization. Next, we classify different types of mathematical programming problems and define some classical combinatorial problems as examples. One section is dedicated to describe the Branch & Bound technique, used to solve integer linear programming problems (ILP). The last part defines the multiobjective combinatorial optimization concept, showing also some classical examples.
  - Chapter 3 shows the most used scalarization techniques for multiobjective combinatorial problems as well as a generic framework which is capable to solve them. This framework will be the backbone for some of the algorithms that will be exposed in the following chapters. A rapid overview of the heuristics and hybrid metaheuristics is provided. Another section defines all the quality indicators that will be used in the computational experiments associated to the algorithms. Last, we describe some non-parametric tests that will be applied to validate all the computational results.
- **Part II. Algorithmic proposals**
  - Chapter 4 proposes an exact algorithm, called *TPA*, which solves any multiobjective combinatorial optimization problem. First, we conduct a deep revision of the literature in this field. After developing the *TPA* method, proving that it works correctly, we show the computational results and validate them. We finish the chapter with the conclusions.
  - Chapter 5 is divided in three parts. The first two parts provide two different algorithms and the third one finishes with the conclusions. The first method, called *MOFeLS*, is an approximated algorithm which solves multiobjective optimization problems with binary variables and linear constraints. The second method, called *HBOXES*, is developed from *MOFeLS*, but it has no limitation for the type of the variables. They can be integer instead of binary. In the computational experiments, the two algorithms are compared between them, and also with *TPA*.
- **Part III. Applications**
  - Chapter 6 describes a problem in the field of Software Engineering, called Software Module Clustering. The problem is approached in two ways: by a Branch & Bound technique, and by the resolution of a mixed integer linear program (with continuous and discrete variables). The problem is solved using single-objective optimization. There exists also a bi-objective version of this problem, but the preliminary experiments did not provide good results and we focused on the study of the single-objective version. The initial mathematical program, which is nonlinear, is completely linearized. Moreover, the modification of some constraints, allows us

UNIVERSIDAD DE MÁLAGA

to solve the problem in just one call to the ILP solver. Unfortunately, the results are already far from being considered good results, because the time required to certify the optimal solution is too long.

- Chapter 7 discusses another well known problem in Software Engineering: the Next Release Problem. We approach two versions of this problem, with one and with two objectives. For the single-objective version, we propose two hybrid algorithms for its resolution. They are based on a metaheuristic called Construct, Merge, Solve and Adapt [12]. The bi-objective version is solved using an adaptation of five different methods. We also show the computational results and the conclusions for these two approaches.

B.2. Introduction to combinatorial optimization

This chapter starts with the definition of a decision problem and how the algorithmic complexity can measure the difficulty of solving it. Four classes of problems are defined: $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-*Complete* and $\mathcal{NP}$-*Hard*. It is still an open question if $\mathcal{P}$=$\mathcal{NP}$ or not. While we do not have an answer to this conjecture, we are not able to conclude that there always exists an algorithm which can solve any optimization problem in polynomial time.

Next, we define the term *mathematical programming* as those optimization problems which require mathematical tools to solve them. A classification of these problems is done, including the number of objectives to optimize and the character of the variables. After the formal definition of what a combinatorial problem is, we describe the mathematical formulation of five well known combinatorial problems in the field of *Operational Research*: Assignment Problem (AP), Knapsack Problem (KP), Multidimensional Knapsack Problem (MKP), Quadratic Assignment Problem (QAP), and Set Covering Problem (SCP).

The third section of this chapter provides the definitions of convex set, convex function, hyperplane, polyhedra and polytope, so that we have the minimum background for the comprehension of the linear programming theory. Then, after an illustrative graphical example of how the simplex method works, integer linear programming (ILP) is introduced. The main technique to solve ILP problems, called Branch & Bound, is also explained with an example.

Last section introduces the term of multiobjective optimization. After defining what a relationship between two sets is, we talk about some partial orders in the set $\mathbb{R}^p$, and get a general overview of the different types of optimization we may use when we have more than one objective to optimize. The most common relation used is the optimization in the *Pareto* sense, widely covered in the literature. The background is defined here: the concepts of dominance, efficient solution and weakly efficient solution, Pareto set, Pareto front, convex hull of a set, supported efficient solution, lexicographical optimal solution, and the ideal and nadir points in a set of non-dominated points. We conclude with some examples of multiobjective combinatorial optimization problems (MOCO): MultiObjective Assignment Problem ($MOAP$), MultiObjective Knapsack Problem ($MOKP$), MultiObjective Multidimensional Knapsack Problem ($MOMKP$), MultiObjective Integer Linear Programming (MOILP), and MultiObjective Set Covering Problem ($MOSCP$).

B.3. Techniques to solve multiobjective combinatorial optimization problems

This chapter focuses on different techniques to solve MOCO problems. Starting with the exact methods, the following scalarizations are described:

- *Weighted sum*. We give some importance to each objective, and scale all of them through a weighted sum involving positive weights. The solution will always be efficient, but its image will lie in the supported Pareto front. Non-supported solutions are not obtained using this method.

- ε-*constraint*. As for the weighted sum, this is probably the most well known scalarization technique. It uses only one objective function to optimize, and the rest are transformed into constraints. These new constraints are bounded by certain numbers, which represent

UNIVERSIDAD DE MÁLAGA

different points in $\mathbb{R}^{p-1}$. Depending on these bounds, we are exploring different regions in the objective space. All the solutions are guaranteed to be weakly efficient. For this method, we need to do another call to the solver if we want to assure the efficiency of the solution.

- *Hybrid method*. This method combines a weighted scalarization of all the objectives, and establishes p new constraints, one per objective. In some way, it is a mixture of the weighted sum and the ε-constraint method. Using this approach, all the generated solutions are efficient. With the appropriate configuration of the parameters, we can easily obtain the complete Pareto front, including the non-supported solutions.

- *Weighted Tchebycheff method*. In this case, we take a point as a reference point (usually the ideal point), and we try to get solutions as close to this reference point as possible. In the formulation, the $\|\cdot\|_\infty$ metric is used, so the model must be linealized. The solutions obtained by the weighted Tchebycheff method are weakly efficient.

- *Augmented weighted Tchebycheff method*. Adding a new term in the formulation of the weighted Tchebycheff method, that uses the $\|\cdot\|_1$ norm, we obtain a new model which is capable to obtain efficient solutions. With the appropriate configuration of the parameters, it is also possible to obtain the complete Pareto front.

All the scalarization techniques described above can be summarized in a generic formulation, which is described in [59] and is also reproduced in this thesis.

Next, we show a generic framework (see [36]) that is able to solve any multiobjective combinatorial problem. This is a basic reference for our new contributions and is used in some of the algorithms described in the thesis. An important concept is the definition of a *box*, which is a hyperrectangle in $\mathbb{R}^p$, that is, a generalization of a rectangle to any dimension.

To obtain efficient solutions when solving the optimization problems, we need an ILP solver. In this thesis, we use the CPLEX solver, and we dedicate one subsection to describe some of its parameters.

The second part of this chapter covers a brief description of the most important metaheuristics. After a classification of the approximated algorithms as *ad hoc* heuristics and metaheuristics, the former can be divided into decomposition methods, constructive methods and local search methods; and the latter are divided into trajectory-based methods and population methods. Some of trajectory-based methods are *Simulated Annealing*, *Tabu Search*, *Greedy Randomized Adaptive Search Procedure*, *Variable Neighborhood Search* or *Iterated Local Search*. On the other hand, some of the population methods are *Evolutionay Algorithms*, *Estimaton of Distribution Algorithms*, *Scatter Search*, *Ant Colony Optimization* and *Particle Swarm Optimization*. We do not deepen in these methods because our focus is on the scalarization methods. The three approximated multiobjective metaheuristics algorithms used in this thesis are NSGA-II, SPEA2 and MOEA/D. We briefly comment how they work.

The next section provides a brief description of some hybrid metaheuristics. When we combine a metaheuristic with mathematical programming, we use the term *matheuristic*. The algorithms described in Chapter 5 are matheuristics.

The last section describes *quality indicators*, which are needed to measure how well the obtained solutions are. They are described using formal and technical definitions, including the concepts of *compatibility* and *completeness*. The metrics defined are: the number of solutions (ONVG), the hypervolume (HV), the additive epsilon indicator (ε_+), the general spread (Δ^*), the generational distance (GD), and the inverted generational distance (IGD). The concept of a *Pareto compliant* metric is also introduced.

A desirable property of the algorithms solving MOCO problems is that the images of the efficient solutions obtained are well-spread over the objective space at any time during the execution. We call *anytime* to those algorithms satisfying the previous property and we dedicate one subsection to this idea, followed by the definition of two other quality indicators: the *Overall Non-dominated Vector Generation Ratio* (ONVGR) an the *HyperVolume Ratio* (HVR).

Finally, a small description of the non-parametric tests used in the computational experiments is provided. They are the signed rank Wilcoxon test, the Friedman test and the post-hoc Nemenyi test.

UNIVERSIDAD DE MÁLAGA

B.4. *TPA* algorithm

This chapter contains one of the algorithms that is supported by a publication in a scientific journal (see Domínguez et al. [50]). It starts defining a generic framework to solve MOCO problems. Analyzing that framework, the following five questions arise:

- What mathematical program do we need to use when analyzing a box?
- What is the criterion for selecting the next box to explore in the algorithm?
- When we find a solution, how do we split the boxes in the objective space so that we cannot find the same solution in the future?
- When we obtain a solution, we split some boxes and discard a region in the objective space. Can we ensure that the region we exclude does not contain any new non-dominated point?
- How can we eliminate redundant regions in order to improve the efficiency?

All the questions are solved during the chapter and after that the algorithm is described. The second subsection makes a deep literature review of exact algorithms that solve multi-objective problems. It is organized in four subsections, including the first exact algorithms, those which are based on recursion, those based on Branch & Bound techniques, and concludes with the modern algorithms. After the revision, we selected two competitive algorithms to compare with. We choose them because they were not outperformed by any other algorithms and can be considered anytime methods.

Preliminary experiments with some selected instances show us the great influence that the mathematical program has when we want to obtain well-spread solutions. Thus, we decided to use a variant of the augmented Tchebycheff method, but weighing also the augmented part that contains the $\|\cdot\|_1$ norm.

The split of boxes in the generic method is named *full p-split*. After we find a new solution, z, we split one or various boxes. For each box, this split creates p new boxes, so that the region that z dominates is discarded. The definition of *full p-split* let us to define the concept of *direction* of a box, which will be used as the criteria of the next box to select in future iterations. The idea is to select boxes with different directions at each iteration. We focus on one direction each time, which is, intuitively, a good way to explore different regions in the objective space.

After a new solution is found, a new way of splitting the boxes is defined, called *p-partition*. This method is similar to *full p-split* but provides better results in the computational experiments. It does a partition of the boxes excluding the region dominated by the last solution found. Since the new boxes have no common points, the volume of them represent the potential number of non-dominated points inside that box in a more precise way. Finally, by the use of *p-partition*, we can ensure that no repeated solutions are found in the future and there are no missing points in the discarded zones.

To improve the efficiency, the redundant regions should be eliminated. For the general method this is done using the *Redundancy Elimination* (RE) algorithm [103]. In this case, we adapt this idea to *p-partition*, introducing the operation of *joining* boxes. This is a necessary condition if we do not want to skip some solutions during the search.

The selection of the new box to explore is done by means of an alternation of directions. With one direction fixed, we select the box with the highest *reduced volume*, which is a new function estimating the potential number of non-dominated points inside the box.

The *TPA* algorithm contains the following contributions:

- A new strategy to select the appropriate search space region as the next box to explore.
- A new way of partitioning the search space after finding a new non-dominated point.
- The definition of a new quality function to set the priority for the new regions to explore.

It is compared with two other methods, due to Ceyhan et al. [18], and Holzmann and Smith [91]. Our proposal is called *TPA* because it uses an augmented and weighted Tchebycheff method, using *p-Partition* and Alternating the directions in the search.

UNIVERSIDAD DE MÁLAGA

In the results, we compared the three methods in several ways. A deep performance analysis was done using 480 instances and executing the algorithms 30 times in each of them. We compared the algorithms using four different metrics: the number of solutions, the hypervolume, the general spread, and a variant of the additive epsilon indicator (normalizing the values according to each objective). An initial analysis shows the number of times that each algorithm obtains the best average value without exclusivity, and with exclusivity, for the metrics, and for different execution times. Then, we computed a rank for each of them in each instance. We did this at different stopping times from 90 s to 900 s, with a step of 90 s. Moreover, we analyze the anytime behavior for three of the hardest instances during the first 900 seconds, measuring the metrics every five seconds. Finally, we checked if the observed differences between the three algorithms are statistically significant using the non-parametric Friedman test, the Nemenyi multiple comparison test, and the Wilcoxon signed rank test. We can statistically confirm (for this benchmark of instances) that *TPA* is better than the state-of-the-art methods in almost all the cases, improving the number of solutions, hypervolume and additive epsilon indicator for all the time cut-points used, which indicates a better anytime behavior. No good results were obtained from the general spread indicator, which is non Pareto compliant.

B.5. Hybrid algorithms based on multiobjective combinatorial optimization

Metaheuristics and, in particular, evolutionary algorithms, have been very successful solving multiobjective optimization problems using only the information about the fitness function and the constraint violations. Most of the time, in a real context, we have more information than just the evaluation of each solution. The structure of the objective and constraint functions is usually available to be exploited. This idea is what this chapter is about. We propose two hybrid methods in separate sections. The first one, called *MOFeLS*, was presented in the International Congress EvoApps 2021 (see Domínguez et al. [51]). The second one, called *HBOXES*, is an unpublished work. We are looking for some configuration of the algorithm that provides competitive results, which are not already achieved.

MOFeLS algorithm

With an increasing number of constraints, finding feasible solutions can be difficult for evolutionary computation, where, in many cases, the strategy to get feasibility is based on some kind of penalty: either in the objective function or during the selection or replacement of the solutions in the population. We propose the use of integer linear programming (ILP) solvers for this purpose. The combination of metaheuristics and ILP solvers is not new. A great number of papers on matheuristics have been published. One prominent example is Construct, Merge, Solve and Adapt (CMSA), by Blum et al. In a matheuristic, the ILP solver is commonly used to optimize some subproblem for which an optimal solution can be found in a short time. This differs with our proposal here in which we propose the use of ILP to find a feasible solution that is also located in a region of the objective space determined by a high level strategy to find well-spread solutions (exploration). We leave the optimization to a local search algorithm (exploitation). Finding a well-located feasible solution is much easier than optimizing a constrained problem, and the ILP solver is able to do it in a very short time, improving the efficiency of the search. In short, the contributions of this new algorithm are:

- The use of ILP solvers to find feasible solutions in particular regions of the objective space very fast for linear multiobjective constrained binary optimization problems.
- The combination of ILP solvers with a high level exploration technique and an efficient local search based on Δ-evaluation.
- The comparison of our algorithm (MultiObjective search based on integer linear programming for Feasibility and Local Search, *MOFeLS*), with three well known evolutionary algorithms used in the literature: NSGA-II, SPEA2 and MOEA/D.

UNIVERSIDAD DE MÁLAGA

A total of 28 problem instances from two different benchmarks were used. The quality indicators utilized to evaluate the solutions were ONVG, HV, GD, IGD and the normalized ε_+. Before the execution of the algorithms we used the iterated racing for automatic algorithm configuration (IRACE) [121], to tune and obtain good parameter values for each algorithm: ours, and the three metaheuristics.

The numerical results show a very good performance for *MOFeLS*, compared to the other three metaheuristics. The higher differences were produced in instances with equality constraints (for the multiobjective assignment problem), where the metaheuristics were not able to find any solutions, and *MOFeLS* provided some solutions with good spread in a short time.

HBOXES algorithm

Based on the idea of *MOFeLS*, this new proposal uses the ILP solver as an approximated algorithm. While the solver might have to do a great computational effort to solve some subproblems in an exact way, if we limit some internal parameters to get approximated solutions, we may achieve good results with an acceptable effort. The method can be applied to any multiobjective combinatorial optimization problem. Three internal parameters of CPLEX are managed in order to establish the stopping conditions. The first one, CPX_PARAM_EPGAP, sets a relative tolerance on the gap between the best integer objective and the objective of the best node remaining in the tree used in the Branch & Cut. The default value is 10^{-4}. Below this threshold, the solver stops its execution giving the best solution found so far. We can tune this parameter and save computational effort while maintaining good quality of the solutions. The second parameter is CPX_PARAM_INTSOLLIM, which sets a fixed number of feasible solutions found by the solver before stopping the execution. For instance, if it is set to the value six, once the solver have found six feasible solutions, it stops. Depending of the problem, this could be a good approximation. We try to balance quality with effort. The last parameter, CPX_PARAM_TILIM, sets a maximum execution time for the solver, measured in seconds. In a preliminary computational experiments, we observed in some cases how fast the solver found an optimal solution, but how it needed a big extra time to confirm that the solution was, in fact, optimal. This observation justify the tune of these CPLEX parameters.

The idea of the algorithm is the following. As the starting point we use the general framework given by Dächert et al. [36]. We explore one search region or box at each iteration. Although the approximated solution found may not be efficient, we split the boxes using *full p-split*, and also the method RE to eliminate redundant boxes. The boxes are analyzed until we find the optimal solution or a stopping condition is given. Thus, we explore more search regions in less time than using the *TPA* algorithm. Some of the explored boxes do not provide any feasible solution. In those cases we defer its exploration for the future, saving that box in another list, which will be explored when the actual list has been completely explored.

We analyze the computational results using 911 instances for different benchmarks of datasets. We compare *HBOXES* with *MOFeLS* and with *TPA*. The results were outperformed by *MOFeLS* and *TPA*, in general. This lead us to consider a deeper research, with the use of other internal parameters of CPLEX, because we strongly believe that we can take advantage using this idea. This is commented in the conclusions of the chapter.

B.6. Software Module Clustering problem

The Software Module Clustering problem (*SMC*) is a problem in the field of Software Engineering. It was initially addressed by Mancoridis et al. [124]. The main idea consists in providing a modular structure in a software system from its source code. This modularization is treated as an optimization problem. Dividing the source code into clusters improves the quality of its releases, as well as its maintenance. The problem arises when we have code with thousands of lines of code and too many interactions between modules, so that it becomes impossible to do the clustering manually.

Starting with an initial graph, called *Module Dependency Graph* (MDG), we have identified every relationship between modules (these are the arcs or edges, and the modules are the nodes). We need to do a partition of the set of nodes into clusters, taking into account two metrics: the intra-connectivity, which measures the cohesion between modules which are in the same cluster; and inter-connectivity, which measures the coupling between distinct clusters. A

UNIVERSIDAD DE MÁLAGA

high degree of intra-connectivity indicates good subsystem partitioning because the modules grouped within a common subsystem share many software-level components. A low degree of intra-connectivity indicates poor subsystem partitioning because the modules assigned to a particular subsystem share few software-level components (limited cohesion). On the other hand, a high degree of inter-connectivity is undesirable because it indicates that clusters are highly dependent on each other. Conversely, a low degree of inter-connectivity is desirable because it indicates that individual clusters are largely independent of each other. Combining these two metrics we obtain the *Modulatization Quality* (MQ), which aim is to be maximized. In this chapter we use the definition of MQ given by [148].

We propose two exact algorithms to address this problem. The first algorithm uses a Branch & Bound technique, exploring the partition of the nodes for a fixed number of clusters, and considering also the maximum difference between the cluster with more and with less elements. In practise, this algorithm is behaved as quasi enumerative, since the upper bound used to prune the branches is not very tight. The second algorithm uses mathematical programming. The initial formulation of the problem is a non-linear model with a sum of linear fractions as objective function. Based on the linearization proposed by Charnes and Cooper [21] and other variables' changes, we completely linearize the model. The linear model has binary and continuous variables, so it is a mixed integer linear problem. Then, it can be solved using an ILP solver, with one execution for each fixed number of clusters. Discarding the extreme cases, when all clusters are in the same group, or all clusters have isolated nodes, we have to do a total of $n-2$ calls to the solver, being n the number of nodes in the MDG. One improvement is that we relaxed one group of constraints in the model and proved that the *SMC* can be solved just with one call to the solver using the relaxed model.

The *SMC* can also be modelled as a multiobjective problem. We investigated with the bi-objective version, but the results were not competitive. Using real instances for this problem, we found two important issues. On one hand, we detected that the algorithm using the ILP solver had an out of memory error after a few hours of execution, so we needed to allocate a great amount of RAM memory. On the other hand, the results were outperformed by other metaheuristics.

B.7. Next Release Problem

In this chapter we propose another useful problem in the field of Software Engineering: the Next Release problem (*NRP*). It consists in selecting a subset of requirements to develop in the next release of a software product. The selection should be done in a way that maximizes the satisfaction of the stakeholders while the development cost is minimized and the constraints of the requirements are fulfilled. In the bi-objective version we optimize the two objectives. In the single version, we transform one of the objectives into a constraint. We have taken into account that the *NRP* only focuses in the next release. The planning over several versions is a more complex problem included in the theory of Release Planning Problem, which contains the *NRP* as a particular case.

There are two approaches to model the problem. In the first approach one stakeholder is satisfied when all his/her requirements are included in the next release of the software. In the second approach we allow partial satisfaction of the stakeholders (not all their requirements need to be included). In this thesis we focus on the first approach, due to Bagnall et al. [7]. There exist three types of relationships between requirements. The first kind of relationship is combination, and it appears when two requirements must be included simultaneously in the software. The second kind of relationship is the exclusion between two requirements, which means that they cannot be included together (at most one of them is included). Finally, some requirements need other requirements to be included in the software (prerequisite relationship).

In the first section of the chapter, we developed two hybrid algorithms that solve the single-objective version of the *NRP*. They are based on the metaheuristic Construct, Merge, Solve and Adapt (CMSA), due to Blum et al. [12]. The algorithms consider either the requirements or the stakeholders as components. Thus, adapting the CMSA framework, we apply it to solve the problem. We compare both versions with the exact version consisting in solving the

UNIVERSIDAD DE MÁLAGA

problem using an ILP solver with a limited time budget (the same time used for the two CMSA versions).

Recent works have solved the bi-objective problem using exact methods based on Integer Linear Programming. In practice, there is no need to compute all the efficient solutions of the problem, a well-spread set in the objective space is more convenient for the decision maker. The exact methods used in the past to find the complete Pareto front explore the objective space in a lexicographic order or use a weighted sum of the objectives to solve a single-objective problem, finding only supported solutions. In the second section of this chapter, we propose six methods that maintain a well-spread set of solutions at any time during the search, so that the decision maker can stop the algorithm when a large enough set of solutions is found. All the algorithms but one can find both supported and non-supported solutions, and can obtain the whole Pareto front if the time provided is long enough. The content of the section is supported by a publication in a scientific journal (see Domínguez et al. [49]). The first one is called *SPF*, and obtains supported efficient solutions only. The second one is called *AnyAugmecon* and it is based on the augmented ε-constraint, which uses only one iteration to obtain an efficient solution. The third algorithm is based on the *hybrid* method (see Section 3.1.1), which combines a parameterization in the objective function with the ε-constraint. The fourth algorithm is based on the augmented Tchebycheff method. The last algorithm, which is a new contribution, has two versions. Both of them use a combination of the previous methods.

In the computational results for the bi-objective problem, we first compare the supported solutions obtained by *SPF* with those obtained by Veerapen et al. [174]. Next, we provide a comparison between the rest of the algorithms for a benchmark of instances existing in the literature. Finally, we show a comparison between exact methods and *anytime* ones.

B.8. Conclusions and future work

The present PhD thesis carries out a design of exact and heuristic algorithms to solve multiobjective combinatorial optimization problems where the solutions computed are well-spread over the objective space at any moment during the search. When this property holds, we talk about *anytime* algorithms. After the introductory chapter, the following two chapters contain the theoretical framework of the thesis. On the one hand, a short introduction to combinatorial optimization is made, and on the other hand, different techniques that solve multiobjective optimization problems are defined, focusing on the exact strategies.

The first algorithmic proposal, developed in Chapter 4, called *TPA*, establishes an exact method that solves any multiobjective combinatorial optimization problem. This method obtains solutions well-spread through the objective space at any time during the execution, which makes it more interesting at a practical level. We can conclude that the method behaves well for medium problems or those in which it is not very expensive to obtain a solution in terms of execution time. Obviously, the harder the problem, the poorer the algorithm will perform, but for intermediate sizes, obtaining well-spread exact solutions can be very beneficial compared to approximated algorithms. Future work can explore the extension of the experimental study to other benchmarks, to analyze in which kind of problems our proposed algorithm works best. We can also apply *TPA* to multiobjective industrial problems, where a few efficient solutions covering the objective space are required by decision makers in a short time. The ideas introduced in this chapter can also be applied to heuristic algorithms, in the field of evolutionary multiobjective optimization, for example, to improve the performance of multiobjective metaheuristics. We also plan to combine heuristics with exact methods to create new hybrids methods, taking the advantages of both fields.

In Chapter 5, two hybrid algorithms are proposed that solve multiobjective combinatorial problems. The first one, called *MOFeLS*, works for problems with binary variables and is an approximated algorithm. It uses the ILP solver to obtain feasible solutions, and then performs a local search to obtain a local optimum. The results obtained in the computational experiments have been successful, so it is concluded that the method is useful, even more if we are interested in obtaining approximated Pareto fronts of good quality (well-spread). The algorithm is able to deal with equality constraints, which is a handicap using other metaheuristics. *MOFeLS* provides the best approximated Pareto fronts in terms of number

UNIVERSIDAD DE MÁLAGA

of solutions, hypervolume, generational distance and inverted generational distance for some benchmark of instances. For the ε_+ indicator, the best results are shared by our proposed algorithm and MOEA/D. We think these preliminary results encourage the investigation of other hybrids combining exact methods and metaheuristics. In the future, it would be interesting to compare this algorithm with other matheuristics existing in the literature using additional benchmark problems. We also plan to extend the research and consider higher order objective and constraint functions (not only linear), using the advances in gray-box optimization for constrained multiobjective problems. We also want to analyze the behaviour of the algorithms as time progresses.

The second proposed algorithm in Chapter 5, called *HBOXES*, emerged as an extension of the first, although it is valid to solve any multiobjective combinatorial optimization problem. It is based on the idea of trying to get a better performance from the use of the ILP solver, tuning some parameters, in order to obtain a higher quality solutions directly from the solver, without the need of performing a local search. After the computational experiments we conclude that this second method needs a deep study to understand the kind of problems in which it could be beneficial. The presented ideas opens the door to a variety of possible combinations that could generate multiple different hybrid algorithms.

Chapter 6 focuses on the Software Module Clustering problem (SMC), a classical problem in Software Engineering. Two algorithms are proposed that solve it exactly. The first one is based on a Branch & Bound technique, although in practice it behaves almost as an enumerative algorithm. The second algorithm solves the problem by means of a single-objective mixed integer linear programming problem. We verified that its bi-objective formulation did not provide good results, that is why we focused on the single-objective version. The single-objective version linearizes the initial formulation, and with some transformations, is able to solve the SMC with a single call to the solver. One drawback is that the complexity of the problem grows faster than exponential in terms of the number of nodes of the MDG. Thus, the method is very time consuming. However, we tried to add constraints to the model that allow us to prune some branches in the exploration, looking for a more competitive method when obtaining the optimum. However, the computational results are still far from those obtained by heuristic methods.

Future work can investigate new ideas to help the ILP solver to find good solutions and proof that the best solution found is really optimal.

The last chapter shows six algorithms that solve the Next Release Problem (*NRP*). This problem has been approached in its two variants: single-objective and bi-objective version. The single-objective version is solved using two hybrid algorithms that are based on a matheuristic technique known as Construct, Merge, Solve and Adapt (CMSA). The computational results for this version can be considered satisfactory. A new study of the CMSA technique has been developed for the bi-objective case. We found that this was not an easy task, and it was quite difficult to generalize the CMSA technique for this problem.

Regarding the bi-objective version of the *NRP*, finding the whole Pareto front for these problems is time consuming and unnecessary in most of the cases, since the decision maker only needs a few well-spread solutions in the objective space to take the decision. We propose here six exact algorithms to find a well-spread set of solutions at any time from the beginning of the search. The algorithms obtain a set of well-spread solutions in the objective space within a few seconds, while the complete front require several hours of computation for most of the instances used. We claim that these *anytime* algorithms should be the preferred ones by the decision makers in Software Engineering, since they allow them to play with different parameters and have exact answers in seconds. In the literature of the Next Release Problem, however, most of the works use metaheuristic algorithms, that cannot guarantee that efficient solutions are found. We have worked here with the bi-objective Next Release Problem, but the same idea can be applied to other Software Engineering problems as a future work. The main key ingredient for a successful application of anytime algorithms is an efficient exact method to find the efficient solutions. Regarding the *NRP*, there are other variants where the value or satisfaction are not certain or depend on the presence/absence of other requirements. These variants would require a different, more complex, formulation to be solved with our anytime algorithms that can be addressed also in future work. Other lines for future work include solving largest instances, probably combining exact methods and heuristics, improving the anytime algorithms, and extending them to more than two objectives.

UNIVERSIDAD DE MÁLAGA

As a final conclusion, we highlight the utility of using hybrid methods in the field of combinatorial optimization, also for the multiobjective case. Since it is not proved that $\mathcal{P}$=$\mathcal{NP}$, it is reasonable to think that combining a good trade-off between quality of solutions (provided by the solver) and the quantity of them (thanks to the approximate methods), is a good choice in the design of these algorithms. The research in multiobjective combinatorial optimization seems to be unlimited. The beauty and satisfaction of investigating in this area is what drive us to deepen in this research field.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Símbolos, siglas y acrónimos

$\mathcal{NP}$ Clase de problemas *NP*.

$\mathcal{NP}$***-Complete*** Clase de problemas *NP-Completo*.

$\mathcal{NP}$***-Hard*** Clase de problemas *NP-Difícil*.

$\mathcal{P}$ Clase de problemas $\mathcal{P}$.

$\mathcal{X}$ Espacio de decisiones o conjunto factible (*Decision space or feasible set*).

NRP Problema de la siguiente versión (*Next Release Problem*).

SMC Partición de módulos software (*Software Module Clustering*).

HBOXES Algoritmo aproximado para problemas combinatorios multiobjetivo (*Heuristic based on exploration of BOXES*).

MOFeLS Algoritmo aproximado para problemas binarios multiobjetivo con restricciones lineales (*MultiObjective search based on ILP for Feasibility and Local Search*).

PF Frente de Pareto (*Pareto Front*).

TPA Algoritmo exacto para problemas combinatorios multiobjetivo (*Tchebycheff-Partition-Alternation*).

ILP Programación lineal entera (*Integer Linear Programming*).

LP Programación lineal (*Linear Programming*).

MOBO Optimización binaria multiobjetivo (*MultiObjective Binary Optimization*).

MOCO Optimización combinatoria multiobjetivo (*MultiObjective Combinatorial Optimization*).

MOILP Programación lineal entera multiobjetivo (*MultiObjective Integer Linear Programming*).

MOMILP Programación lineal multiobjetivo mixta (*MultiObjective Mixed Integer Linear Programming*).

MOO Optimización multiobjetivo (*MultiObjective Optimization*).

MOP Programación multiobjetivo (*MultiObjective Programming*).

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Bibliografía

[1] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz'ri Mahrin, *A systematic literature review of software requirements prioritization research*, Information and Software Technology **56** (2014), no. 6, 568–585.

[2] Maria Joao Alves and João Clímaco, *A review of interactive methods for multiobjective integer and mixed-integer programming*, European Journal of Operational Research **180** (2007), no. 1, 99–115.

[3] Yash P. Aneja and Kunhiraman P.K. Nair, *Bicriteria transportation problem*, Management Science **25** (1979), no. 1, 73–78.

[4] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook, *The traveling salesman problem: A computational study*, Princeton University Press, 2006.

[5] David L. Applegate, William Cook, Sanjeeb Dash, and Daniel G Espinoza, *Exact solutions to linear programming problems*, Operations Research Letters **35** (2007), no. 6, 693–699.

[6] Md Asafuddoula, Tapabrata Ray, Ruhul Sarker, and Khairul Alam, *An adaptive constraint handling approach embedded MOEA/D*, 2012 IEEE Congress on Evolutionary Computation, IEEE, 2012, pp. 1–8.

[7] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M Whittley, *The next release problem*, Information and Software Technology **43** (2001), no. 14, 883–890.

[8] Jayanth Balasubramanian and Ignacio E. Grossmann, *Scheduling optimization under uncertainty—an alternative approach*, Computers & Chemical Engineering **27** (2003), no. 4, 469–490.

[9] Richard Bellman, *On the theory of dynamic programming*, Proceedings of the National Academy of Sciences of the United States of America **38** (1952), no. 8, 716.

[10] Jean-François Bérubé, Michel Gendreau, and Jean-Yves Potvin, *An exact e-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits*, European Journal of Operational Research **194** (2009), no. 1, 39–50.

[11] Christian Blum and Jordi Pereira, *Extension of the CMSA algorithm: an LP-based way for reducing sub-instances*, Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 285–292.

[12] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A. Lozano, *Construct, merge, solve & adapt A new general algorithm for combinatorial optimization*, Computers & Operations Research **68** (2016), 75–88.

[13] Christian Blum, Jakob Puchinger, Günther R. Raidl, and Andrea Roli, *Hybrid metaheuristics in combinatorial optimization: A survey*, Applied Soft Computing **11** (2011), no. 6, 4135–4151.

[14] Christian Blum and Andrea Roli, *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM computing surveys (CSUR) **35** (2003), no. 3, 268–308.

UNIVERSIDAD DE MÁLAGA

[15] Marco A. Boschetti, Vittorio Maniezzo, Matteo Roffilli, and Antonio Bolufé Röhler, *Matheuristics: Optimization, simulation and control*, International Workshop on Hybrid Metaheuristics, Springer, 2009, pp. 171–177.

[16] Stephen Boyd and Lieven Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[17] James A. Carlson, Arthur Jaffe, and Andrew Wiles, *The millennium prize problems*, Citeseer, 2006.

[18] Gökhan Ceyhan, Murat Köksalan, and Banu Lokman, *Finding a representative nondominated set for multi-objective mixed integer programs*, European Journal of Operational Research **272** (2019), no. 1, 61–77.

[19] L.G. Chalmet, L. Lemonidis, and D.J. Elzinga, *An algorithm for the bi-criterion integer programming problem*, European Journal of Operational Research **25** (1986), no. 2, 292–300.

[20] Vira Chankong and Yacov Y. Haimes, *Multiobjective decision making: Theory and methodology*, Elsevier Science Publishing Co. New York, NY, 1983.

[21] A. Charnes and W. W. Cooper, *Programming with linear fractional functionals*, Naval Research Logistics Quarterly **9** (1962), no. 3-4, 181–186.

[22] Francisco Chicano, Miguel Ángel Domínguez, Isabel M. del Águila, José del Sagrado, and Enrique Alba, *Dos estrategias de búsqueda anytime basadas en programación lineal entera para resolver el problema de selección de requisitos*, Actas de las XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD), 2016, http://hdl.handle.net/11705/JISBD/2016/041.

[23] Francisco Chicano et al., *Metaheurísticas e Ingeniería del Software*, Ph.D. thesis, Dtpo. Lenguajes y Ciencias de la Computación. Universidad de Málaga, 2007.

[24] Francisco Chicano, Darrell Whitley, and Andrew M. Sutton, *Efficient identification of improving moves in a ball for pseudo-boolean problems*, Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 437–444.

[25] Francisco Chicano, Darrell Whitley, and Renato Tinos, *Efficient hill climber for constrained pseudo-boolean optimization problems*, Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 309–316.

[26] Christian Cintrano, Francisco Chicano, and Enrique Alba, *Using metaheuristics for the location of bicycle stations*, Expert Systems with Applications **161** (2020), 113684.

[27] Carlos A. Coello Coello, Gary B. Lamont, David A. Van Veldhuizen, et al., *Evolutionary algorithms for solving multi-objective problems*, vol. 5, Springer, 2007.

[28] Stephen A. Cook, *The complexity of theorem-proving procedures*, Proceedings of the third annual ACM symposium on Theory of computing, 1971, pp. 151–158.

[29] William Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver, *Combinatorial optimization*, Wiley, 1997.

[30] William Cook, Sanjeeb Dash, Ricardo Fukasawa, and Marcos Goycoolea, *Numerically safe gomory mixed-integer cuts*, INFORMS Journal on Computing **21** (2009), no. 4, 641–649.

[31] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter, *An exact rational mixed-integer programming solver*, International Conference on Integer Programming and Combinatorial Optimization, Springer, 2011, pp. 104–116.

[32] William Cook and P. Seymour, *Tour merging via branch-decomposition*, INFORMS Journal on Computing **15** (2003), no. 3, 233–248.

UNIVERSIDAD DE MÁLAGA

[33] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, MIT press, 2009.

[34] Gerard Cornuejols and Reha Tütüncü, *Optimization methods in finance*, vol. 5, Cambridge University Press, 2006.

[35] Kerstin Dächert, Jochen Gorski, and Kathrin Klamroth, *An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems*, Computers & Operations Research **39** (2012), no. 12, 2929–2943.

[36] Kerstin Dächert and Kathrin Klamroth, *A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems*, Journal of Global Optimization **61** (2015), no. 4, 643–676.

[37] Kerstin Dächert, Kathrin Klamroth, Renaud Lacour, and Daniel Vanderpooten, *Efficient computation of the search region in multi-objective optimization*, European Journal of Operational Research **260** (2017), no. 3, 841–855.

[38] George B. Dantzig, *Maximization of a linear function of variables subject to linear inequalities*, Activity analysis of production and allocation **13** (1951), 339–347.

[39] George B. Dantzig and Mukund N. Thapa, *Linear programming 1: introduction*, Springer Science & Business Media, 2006.

[40] T. Dean and M. Boddy, *Solving time-dependent planning problems*, Proceedings of the 7th National Conference on Artificial Intelligence, 1988, pp. 49–54.

[41] Walter Dean, *Computational complexity theory*, 2015.

[42] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan, *A fast and elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II*, International conference on parallel problem solving from nature, Springer, 2000, pp. 849–858.

[43] Janez Demšar, *Statistical comparisons of classifiers over multiple data sets*, The Journal of Machine Learning Research **7** (2006), 1–30.

[44] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera, *A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*, Swarm and Evolutionary Computation **1** (2011), no. 1, 3–18.

[45] Clarisse Dhaenens, Julien Lemesre, and El-Ghazali Talbi, *K-PPM: A new exact method to solve multi-objective combinatorial optimization problems*, European Journal of Operational Research **200** (2010), no. 1, 45–53.

[46] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), no. 1, 269–271.

[47] Emanuele Dilettoso, Santi Rizzo, and Nunzio Salerno, *A weakly Pareto compliant quality indicator*, Mathematical and Computational Applications **22** (2017), no. 1, 25.

[48] Miguel Ángel Domínguez-Ríos, Francisco Chicano, and Enrique Alba, *Hacia un algoritmo exacto para resolver el problema de agrupamiento de módulos software*, Jornadas de Ingeniería del Software y Bases de Datos, JISBD, 2017.

[49] Miguel Ángel Domínguez-Ríos, Francisco Chicano, Enrique Alba, Isabel del Águila, and José del Sagrado, *Efficient anytime algorithms to solve the bi-objective next release problem*, Journal of Systems and Software **156** (2019), 217–231.

[50] Miguel Ángel Domínguez-Ríos, Francisco Chicano, and Enrique Alba, *Effective anytime algorithm for multiobjective combinatorial optimization problems*, Information Sciences **565** (2021), 210–228.

UNIVERSIDAD DE MÁLAGA

[51] Miguel Ángel Domínguez-Ríos, Francisco Chicano, and Enrique Alba, *Improving search efficiency and diversity of solutions in multiobjective binary optimization by using metaheuristics plus integer linear programming*, International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2021, pp. 242–257.

[52] Marco Dorigo and Thomas Stützle, *Handbook of metaheuristics, volume 57 of international series in operations research and management science, chapter the ant colony optimization metaheuristics: Algorithms, applications and advances*, 2003.

[53] D. Doval, S. Mancoridis, and B. S. Mitchell, *Automatic clustering of software systems using a genetic algorithm*, Software Technology and Engineering Practice, 1999. STEP '99. Proceedings, 1999, pp. 73–81.

[54] Daniel Duque, Leonardo Lozano, and Andrés L. Medaglia, *An exact method for the biobjective shortest path problem for large-scale road networks*, European Journal of Operational Research **242** (2015), no. 3, 788 – 797.

[55] Juan J. Durillo and Antonio J. Nebro, *jMetal: A java framework for multi-objective optimization*, Advances in Engineering Software **42** (2011), no. 10, 760–771.

[56] Juan J. Durillo, Y. Zhang, E. Alba, and Antonio J. Nebro, *A study of the multi-objective next release problem*, Search Based Software Engineering, 2009 1st International Symposium on, IEEE, 2009, pp. 49–58.

[57] Juan J. Durillo, Yuanyuan Zhang, Enrique Alba, Mark Harman, and Antonio J. Nebro, *A study of the bi-objective next release problem*, Empirical Software Engineering **16** (2011), no. 1, 29–60.

[58] Matthias Ehrgott, *Multicriteria optimization*, vol. 491, Springer Science & Business Media, 2005.

[59] Matthias Ehrgott, *A discussion of scalarization techniques for multiple objective integer programming*, Annals of Operations Research **147** (2006), no. 1, 343–360.

[60] Matthias Ehrgott, *Multiobjective optimization*, AI Magazine **29** (2008), no. 4, 47–47.

[61] Matthias Ehrgott and Xavier Gandibleux, *Hybrid metaheuristics for multi-objective combinatorial optimization*, Hybrid metaheuristics, Springer, 2008, pp. 221–259.

[62] Matthias Ehrgott and Stefan Ruzika, *Improved ε-constraint method for multiobjective programming*, Journal of Optimization Theory and Applications **138** (2008), no. 3, 375.

[63] Matthias Ehrgott and David M. Ryan, *Constructing robust crew schedules with bicriteria optimization*, Journal of Multi-Criteria Decision Analysis **11** (2002), no. 3, 139–150.

[64] Matthias Ehrgott and Dagmar Tenfelde-Podehl, *Computation of ideal and nadir values and implications for their use in MCDM methods*, European Journal of Operational Research **151** (2003), no. 1, 119–139.

[65] Agoston E. Eiben and James E. Smith, *Introduction to evolutionary computing*, vol. 53, Springer, 2003.

[66] Michael T.M. Emmerich and André H. Deutz, *A tutorial on multiobjective optimization: fundamentals and evolutionary methods*, Natural Computing **17** (2018), no. 3, 585–609.

[67] Reza Zanjirani Farahani, Masoud Hekmatfar, Alireza Boloori Arabani, and Ehsan Nikbakhsh, *Hub location problems: A review of models, classification, solution techniques, and applications*, Computers & Industrial Engineering **64** (2013), no. 4, 1096–1109.

[68] Thomas A. Feo and Mauricio G.C. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization **6** (1995), no. 2, 109–133.

[69] Kostas Florios and George Mavrotas, *Generation of the exact pareto set in multiobjective traveling salesman and set covering problems*, Applied Mathematics and Computation **237** (2014), 1–19.

UNIVERSIDAD DE MÁLAGA

[70] Carlos M. Fonseca, Luís Paquete, and Manuel López-Ibánez, *An improved dimension-sweep algorithm for the hypervolume indicator*, Proceedings of the IEEE Congress on Evolutionary Computation, 2006. CEC 2006, IEEE, 2006, pp. 1157–1163.

[71] Carlos M. Fonseca, Luis Paquete, Thomas Stützle, Manuel López-Ibáñez, and Marco Chiarandini, *Package 'eaf'*, 2015.

[72] Milton Friedman, *A comparison of alternative tests of significance for the problem of m rankings*, The Annals of Mathematical Statistics **11** (1940), no. 1, 86–92.

[73] Giorgio Gallo and Stefano Pallottino, *Shortest path algorithms*, Annals of Operations Research **13** (1988), no. 1, 1–79.

[74] Michael R. Gary and David S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, 1979.

[75] Fred Glover, *Future paths for integer programming and links to artificial intelligence*, Computers & Operations Research **13** (1986), no. 5, 533–549.

[76] Fred Glover, *A template for scatter search and path relinking*, European Conference on Artificial Evolution, Springer, 1997, pp. 1–51.

[77] Fred Glover, Manuel Laguna, and Rafael Marti, *Principles of tabu search*, Approximation algorithms and metaheuristics **23** (2007), 1–12.

[78] Ralph Gomory, *An algorithm for the mixed integer problem*, Tech. report, RAND CORP SANTA MONICA CA, 1960.

[79] Juan José Salazar González, *Programación matemática*, Díaz de Santos, 2001.

[80] Harvey J. Greenberg, William E. Hart, and Giuseppe Lancia, *Opportunities for combinatorial optimization in computational biology*, INFORMS Journal on Computing **16** (2004), no. 3, 211–231.

[81] Des Greer and Guenther Ruhe, *Software release planning: an evolutionary and iterative approach*, Information and Software Technology **46** (2004), no. 4, 243–253.

[82] Jürgen Guddat, *Multiobjective and stochastic optimization based on parametric optimization*, vol. 26, Akademie-Verlag, 1985.

[83] Andreia P. Guerreiro, Carlos M. Fonseca, and Luís Paquete, *The hypervolume indicator: Problems and algorithms*, arXiv preprint arXiv:2005.00515, 2020.

[84] Yacov Haimes, *On a bicriterion formulation of the problems of integrated system identification and system optimization*, IEEE transactions on systems, man, and cybernetics **1** (1971), no. 3, 296–297.

[85] Mark Harman and Bryan F. Jones, *Search-based software engineering*, Information and Software Technology **43** (2001), no. 14, 833–839.

[86] Mark Harman and Bryan F. Jones, *Search-based software engineering*, Information and Software Technology **43** (2001), no. 14, 833–839.

[87] Mark Harman, Jens Krinke, Inmaculada Medina-Bulo, Francisco Palomo-Lozano, Jian Ren, and Shin Yoo, *Exact scalable sensitivity analysis for the next release problem*, ACM Transactions on Software Engineering and Methodology **23** (2014), no. 2, 1–31.

[88] Keld Helsgaun, *An effective implementation of the lin–kernighan traveling salesman heuristic*, European Journal of Operational Research **126** (2000), no. 1, 106 – 130.

[89] Arne Herzel, Stefan Ruzika, and Clemens Thielen, *Approximation methods for multiobjective optimization problems: A survey*, INFORMS Journal on Computing (2021), 1–16.

[90] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan, 1975.

UNIVERSIDAD DE MÁLAGA

[91] T. Holzmann and J.C. Smith, *Solving discrete multi-objective optimization problems using modified augmented weighted Tchebychev scalarizations*, European Journal of Operational Research **271** (2018), no. 2, 436–449.

[92] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima, *Modified distance calculation in generational distance and inverted generational distance*, International conference on evolutionary multi-criterion optimization, Springer, 2015, pp. 110–125.

[93] He Jiang, Jingyuan Zhang, Jifeng Xuan, Zhilei Ren, and Yan Hu, *A hybrid ACO algorithm for the next release problem*, Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on, IEEE, 2010, pp. 166–171.

[94] Siwei Jiang, Yew-Soon Ong, Jie Zhang, and Liang Feng, *Consistencies and contradictions of performance metrics in multiobjective optimization*, IEEE transactions on cybernetics **44** (2014), no. 12, 2391–2404.

[95] Yunchol Jong, *Practical global optimization algorithm for the sum-of-ratios problem*, 2012.

[96] Janusz Kacprzyk, *Studies in computational intelligence, volume 153*, 2008.

[97] Joachim Karlsson, Stefan Olsson, and Kevin Ryan, *Improved practical support for large-scale requirements prioritising*, Requirements Engineering **2** (1997), 51–60.

[98] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of computer computations, Springer, 1972, pp. 85–103.

[99] Dean Kelley and María Luisa Díez Platas, *Teoría de autómatas y lenguajes formales*, vol. 22, Prentice Hall, 1995.

[100] James Kennedy, *Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance*, Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol. 3, IEEE, 1999, pp. 1931–1938.

[101] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), no. 4598, 671–680.

[102] Gokhan Kirlik and Serpil Sayın, *A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems*, European Journal of Operational Research **232** (2014), no. 3, 479–488.

[103] Kathrin Klamroth, Renaud Lacour, and Daniel Vanderpooten, *On the representation of the search region in multi-objective optimization*, European Journal of Operational Research **245** (2015), no. 3, 767–778.

[104] Gunnar W. Klau, Ivana Ljubić, Andreas Moser, Petra Mutzel, Philipp Neuner, Ulrich Pferschy, Günther Raidl, and René Weiskircher, *Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem*, Genetic and Evolutionary Computation Conference, Springer, 2004, pp. 1304–1315.

[105] Dieter Klein and Edward Hannan, *An algorithm for the multiple objective integer linear programming problem*, European Journal of Operational Research **9** (1982), no. 4, 378–385.

[106] Joshua D. Knowles, Lothar Thiele, and Eckart Zitzler, *A tutorial on the performance assessment of stochastic multiobjective optimizers*, Tech. report, ETH Zurich, 2006.

[107] Tjalling C. Koopmans and Martin Beckmann, *Assignment problems and the location of economic activities*, Econometrica: journal of the Econometric Society (1957), 53–76.

[108] Harold W. Kuhn, *The hungarian method for the assignment problem*, Naval research logistics quarterly **2** (1955), no. 1-2, 83–97.

UNIVERSIDAD DE MÁLAGA

[109] Renaud Lacour, Kathrin Klamroth, and Carlos M. Fonseca, *A box decomposition algorithm to compute the hypervolume indicator*, Computers & Operations Research **79** (2017), 347–360.

[110] Marco Laumanns, Lothar Thiele, and Eckart Zitzler, *An adaptive scheme to generate the pareto front based on the epsilon-constraint method*, Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005.

[111] Marco Laumanns, Lothar Thiele, and Eckart Zitzler, *An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method*, European Journal of Operational Research **169** (2006), no. 3, 932–942.

[112] Julien Lemesre, Clarisse Dhaenens, and El-Ghazali Talbi, *Parallel partitioning method (PPM): A new exact method to solve bi-objective problems*, Computers & Operations Research **34** (2007), no. 8, 2450–2462.

[113] Chen Li, Marjan van den Akker, Sjaak Brinkkemper, and Guido Diepen, *Integrated Requirement Selection and Scheduling for the Release Planning of a Software Product*, Requirements Engineering: Foundation for Software Quality (2007), 93–108.

[114] Chen Li, Marjan van den Akker, Sjaak Brinkkemper, and Guido Diepen, *An integrated approach for requirement selection and scheduling in software release planning*, Requirements Engineering **15** (2010), no. 4, 375–396.

[115] M. Li and X. Yao, *Quality evaluation of solution sets in multiobjective optimisation: A survey*, ACM Computing Surveys **52** (2019), no. 2, 1–38.

[116] Arnaud Liefooghe and Bilel Derbel, *A correlation analysis of set quality indicator values in multiobjective optimization*, Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 581–588.

[117] S. Lin and B. W. Kernighan, *An effective heuristic algorithm for the traveling-salesman problem*, Operations Research **21** (1973), no. 2, 498–516.

[118] Qi Liu, Xiaofeng Li, Haitao Liu, and Zhaoxia Guo, *Multi-objective metaheuristics for discrete optimization problems: A review of the state-of-the-art*, Applied Soft Computing **93** (2020), 106382.

[119] Banu Lokman and Murat Köksalan, *Finding all nondominated points of multi-objective integer programs*, Journal of Global Optimization **57** (2013), no. 2, 347–365.

[120] Manuel López-Ibáñez and Thomas Stützle, *Automatically improving the anytime behaviour of optimisation algorithms*, European Journal of Operational Research **235** (2014), no. 3, 569–582.

[121] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle, *The irace package: Iterated racing for automatic algorithm configuration*, Operations Research Perspectives **3** (2016), 43–58.

[122] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle, *Handbook of metaheuristics, chapter iterated local search*, 2002.

[123] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, *Bunch: a clustering tool for the recovery and maintenance of software system structures*, Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on, 1999, pp. 50–59.

[124] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, *Using automatic clustering to produce high-level system organizations of source code*, Proceedings of the 6th International Workshop on Program Comprehension (Washington, DC, USA), IWPC '98, IEEE Computer Society, 1998, pp. 45–52.

[125] Dirk Männel and Andreas Bortfeldt, *A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints*, European Journal of Operational Research **254** (2016), no. 3, 840 – 858.

UNIVERSIDAD DE MÁLAGA

[126] Silvano Martello, *Knapsack problems: algorithms and computer implementations*, John Wiley & Sons Ltd., 1990.

[127] Rafael Martí, *Procedimientos metaheurísticos en optimizaci@n combinatoria*, Matemátiques, Universidad de Valencia **1** (2003), no. 1, 3–62.

[128] Michael Masin and Yossi Bukchin, *Diversity maximization approach for multiobjective optimization*, Operations Research **56** (2008), no. 2, 411–424.

[129] Evgeny Maslov, Mikhail Batsyn, and Panos M. Pardalos, *Speeding up branch and bound algorithms for solving the maximum clique problem*, Journal of Global Optimization **59** (2014), no. 1, 1–21.

[130] George Mavrotas, *Effective implementation of the ε-constraint method in multi-objective mathematical programming problems*, Applied Mathematics and Computation **213** (2009), no. 2, 455–465.

[131] George Mavrotas and Danae Diakoulaki, *A branch and bound algorithm for mixed zero-one multiple objective linear programming*, European Journal of Operational Research **107** (1998), no. 3, 530–541.

[132] George Mavrotas and Danae Diakoulaki, *Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming*, Applied Mathematics and Computation **171** (2005), no. 1, 53–71.

[133] George Mavrotas and Kostas Florios, *An improved version of the augmented ε-constraint method (augmecon2) for finding the exact Pareto set in multi-objective integer programming problems*, Applied Mathematics and Computation **219** (2013), no. 18, 9652–9669.

[134] Claudio N. Meneses, Carlos A.S. Oliveira, and Panos M. Pardalos, *Optimization techniques for string selection and comparison problems in genomics*, IEEE Engineering in Medicine and Biology Magazine **24** (2005), no. 3, 81–87.

[135] Efrén Mezura-Montes and Carlos A. Coello Coello, *Constraint-handling in nature-inspired numerical optimization: Past, present and future*, Swarm and Evolutionary Computation **1** (2011), no. 4, 173 – 194.

[136] Michel Minoux, *Mathematical programming: theory and algorithms*, John Wiley & Sons, 1986.

[137] Nenad Mladenović and Pierre Hansen, *Variable neighborhood search*, Computers & Operations Research **24** (1997), no. 11, 1097–1100.

[138] Heinz Mühlenbein, *The equation for response to selection and its use for prediction*, Evolutionary Computation **5** (1997), no. 3, 303–346.

[139] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne, *Redesigning the jMetal multi-objective optimization framework*, Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation, 2015, pp. 1093–1100.

[140] Peter Bjorn Nemenyi, *Distribution-free multiple comparisons.*, Princeton University, 1963.

[141] Melih Özlen and Meral Azizoğlu, *Multi-objective integer programming: a general approach for generating all non-dominated solutions*, European Journal of Operational Research **199** (2009), no. 1, 25–35.

[142] Melih Özlen, Benjamin A. Burton, and Cameron A.G. MacRae, *Multi-objective integer programming: An improved recursive algorithm*, Journal of Optimization Theory and Applications **160** (2014), no. 2, 470–482.

[143] Özgür Özpeynirci and Murat Köksalan, *An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs*, Management Science **56** (2010), no. 12, 2302–2315.

UNIVERSIDAD DE MÁLAGA

[144] Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham (eds.), *Handbook of combinatorial optimization*, Springer, 2013.

[145] Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham, *Handbook of combinatorial optimization*, Springer, 2013.

[146] Antônio Mauricio Pitangueira, Rita Suzana P. Maciel, and Márcio Barros, *Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature*, Journal of Systems and Software **103** (2015), 267–280.

[147] Thorsten Pohlert and Maintainer Thorsten Pohlert, *Package 'PMCMR'*, 2018.

[148] K. Praditwong, M. Harman, and X. Yao, *Software module clustering as a multi-objective search problem*, IEEE Transactions on Software Engineering **37** (2011), no. 2, 264–282.

[149] K. Praditwong and X. Yao, *A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm*, 2006 International Conference on Computational Intelligence and Security, vol. 1, Nov 2006, pp. 286–291.

[150] A. Przybylski, Xavier Gandibleux, and Mathiass Ehrgott, *A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme*, INFORMS Journal on Computing **22** (2010), no. 3, 371–386.

[151] A. Przybylski, Xavier Gandibleux, and Matthias Ehrgott, *A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives*, Discrete Optimization **7** (2010), no. 3, 149–165.

[152] Jakob Puchinger and Günther R. Raidl, *Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification*, International work-conference on the interplay between natural and artificial computation, Springer, 2005, pp. 41–53.

[153] Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy, *The multidimensional knapsack problem: Structure and algorithms*, INFORMS Journal on Computing **22** (2010), no. 2, 250–265.

[154] Ted K. Ralphs, Matthew J. Saltzman, and Margaret M. Wiecek, *An improved algorithm for solving biobjective integer programs*, Annals of Operations Research **147** (2006), no. 1, 43–70.

[155] V. Venkata Rao and R. Sridharan, *The minimum weight rooted arborescence problem: Weights on arcs case*, 1992.

[156] Nery Riquelme, Christian Von Lücken, and Benjamin Baran, *Performance metrics in multi-objective optimization*, 2015 Latin American Computing Conference (CLEI), IEEE, 2015, pp. 1–11.

[157] Günther Ruhe and Des Greer, *Quantitative studies in software release planning under risk and resource constraints*, Proceedings of the International Symposium on Empirical Software Engineering, ISESE 2003, 2003, pp. 262–270.

[158] Günther Ruhe and Moshood Omolade Saliu, *The Art and Science of software release planning*, IEEE Software (2005), no. Nov/Dec, 47–53.

[159] David J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*, crc Press, 2020.

[160] S.M. Sinha, *Chapter 1 - Introduction*, Mathematical Programming (S.M. Sinha, ed.), Elsevier Science, Burlington, 2006, pp. 1–9.

[161] N. J. A. Sloane, *Bell or exponential numbers: number of ways to partition a set of n labeled elements*, Online, 1964.

UNIVERSIDAD DE MÁLAGA

[162] Daniel A. Spielman and Shang-Hua Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, Journal of the ACM (JACM) **51** (2004), no. 3, 385–463.

[163] Ralph E. Steuer, *Multiple criteria optimization: Theory, computation and applications*, John Wiley & Sons, New York, NY, 1986.

[164] Thomas Stidsen, Kim Allan Andersen, and Bernd Dammann, *A branch and bound algorithm for a class of biobjective mixed integer programs*, Management Science **60** (2014), no. 4, 1009–1032.

[165] John Sylva and Alejandro Crema, *A method for finding the set of non-dominated vectors for multiple objective integer linear programs*, European Journal of Operational Research **158** (2004), no. 1, 46–55.

[166] John Sylva and Alejandro Crema, *A method for finding well-dispersed subsets of non-dominated vectors for multiple objective mixed integer linear programs*, European Journal of Operational Research **180** (2007), no. 3, 1011–1027.

[167] El-Ghazali Talbi, *A taxonomy of hybrid metaheuristics*, Journal of heuristics **8** (2002), no. 5, 541–564.

[168] Robert Endre Tarjan, *Efficiency of a good but not linear set union algorithm*, Journal of the ACM (JACM) **22** (1975), no. 2, 215–225.

[169] Dagmar Tenfelde-Podehl, *A recursive algorithm for multiobjective combinatorial optimization problems with q criteria*, Universität Graz/Technische Universität Graz. SFB F003-Optimierung und Kontrolle, 2003.

[170] Alan Mathison Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London mathematical society **2** (1937), no. 1, 230–265.

[171] Ekunda L. Ulungu and Jacques Teghem, *The two-phase method: An efficient procedure to solve bi-objective combinatorial optimization problems*, Foundations of Computing and Decision Sciences **20** (1995), no. 2, 149–165.

[172] Ekunda L. Ulungu and Jacques Teghem, *The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems*, Foundations of Computing and Decision Sciences **20** (1995), no. 2, 149–165.

[173] Robert J. Vanderbei et al., *Linear programming*, vol. 3, Springer, 2015.

[174] Nadarajen Veerapen, Gabriela Ochoa, Mark Harman, and Edmund K. Burke, *An integer linear programming approach to the single and bi-objective next release problem*, Information and Software Technology **65** (2015), 1–13.

[175] Thomas Vincent, Florian Seipp, Stefan Ruzika, Anthony Przybylski, and Xavier Gandibleux, *Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case*, Computers & Operations Research **40** (2013), no. 1, 498–509.

[176] Eric W. Weisstein, *Algorithm*, Wolfram Research, Inc., 2002.

[177] Lyndon While, Lucas Bradstreet, and Luigi Barone, *A fast way of calculating exact hypervolumes*, IEEE Transactions on Evolutionary Computation **16** (2012), no. 1, 86–95.

[178] Andrzej P. Wierzbicki, *A mathematical basis for satisficing decision making*, Mathematical modelling **3** (1982), no. 5, 391–405.

[179] T. A. Wiggerts, *Using clustering algorithms in legacy systems remodularization*, Proceedings of the Fourth Working Conference on Reverse Engineering, Oct 1997, pp. 33–43.

[180] Frank Wilcoxon, *Individual comparisons of grouped data by ranking methods*, Journal of economic entomology **39** (1946), no. 2, 269–270.

UNIVERSIDAD DE MÁLAGA

[181] H. Paul Williams, *Model building in mathematical programming*, John Wiley & Sons, 2013.

[182] W.H. Wolpert and W.G. Macready, *No free lunch theorems for optimization*, IEEE Transactions on Evoutionary Computation **1** (1997), no. 1, 67–82.

[183] Yong Xia and Ya-Xiang Yuan, *A new linearization method for quadratic assignment problems*, Optimisation Methods and Software **21** (2006), no. 5, 805–818.

[184] Jifeng Xuan, He Jiang, Zhilei Ren, and Zhongxuan Luo, *Solving the large scale next release problem with a backbone-based multilevel algorithm*, IEEE Transactions on Software Engineering **38** (2012), no. 5, 1195–1212.

[185] Chian-Son Yu and Han-Lin Li, *A robust optimization model for stochastic logistic problems*, International journal of production economics **64** (2000), no. 1-3, 385–397.

[186] Qingfu Zhang and Hui Li, *MOEA/D: A multiobjective evolutionary algorithm based on decomposition*, IEEE Transactions on evolutionary computation **11** (2007), no. 6, 712–731.

[187] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri, *The multi-objective next release problem*, Proceedings of the 9th annual conference on Genetic and Evolutionary Computation, ACM, 2007, pp. 1129–1137.

[188] Aimin Zhou, Yaochu Jin, Qingfu Zhang, Bernhard Sendhoff, and Edward Tsang, *Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion*, 2006 IEEE International Conference on Evolutionary Computation, IEEE, 2006, pp. 892–899.

[189] Shlomo Zilberstein, *Using anytime algorithms in intelligent systems*, AI magazine **17** (1996), no. 3, 73–73.

[190] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele, *The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration*, International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2007, pp. 862–876.

[191] Eckart Zitzler, Joshua Knowles, and Lothar Thiele, *Quality assessment of Pareto set approximations*, Multiobjective Optimization, Springer, 2008, pp. 373–404.

[192] Eckart Zitzler, Marco Laumanns, and Lothar Thiele, *SPEA2: Improving the strength Pareto evolutionary algorithm*, Tech. report, Eidgenössische Technische Hochschule Zürich (ETH), 2001.

[193] Eckart Zitzler and Lothar Thiele, *Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach*, IEEE transactions on Evolutionary Computation **3** (1999), no. 4, 257–271.

[194] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert Da Fonseca, *Performance assessment of multiobjective optimizers: An analysis and review*, IEEE Transactions on evolutionary computation **7** (2003), no. 2, 117–132.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Índice de figuras

2.1. Clases $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-*Complete* y $\mathcal{NP}$-*Hard*. 12
2.2. Resolución gráfica de un problema de programación lineal. 17
2.3. Ejemplo ilustrativo de problema de programación lineal entera. 18
2.4. Resolución de un problema de PLE usando ramificación y corte. 19
2.5. Conjunto discreto y su envoltura convexa. 23
2.6. Puntos dominados, (débilmente) eficientes, ideal, nadir y óptimos lexicográficos. 24

3.1. Ejemplo de espacio objetivo para $p = 2$. 28
3.2. Escalarización mediante suma ponderada. 29
3.3. Escalarización mediante el método ε-constraint. 30
3.4. Método híbrido usando $w = (1, 4)$ y $\varepsilon = (19, 16)$. 31
3.5. Escalarización mediante el método de Tchebycheff. 32
3.6. Escalarización usando el método aumentado de Tchebycheff. 33
3.7. Clasificación de algoritmos aproximados. 39
3.8. Cálculo de métricas asociadas a un frente aproximado para $p = 2$. 44

4.1. Partición de una caja usando *full p-split*. 56
4.2. Características del algoritmo genérico que resuelve problemas MOCO. 58
4.3. Grafo asociado a la traza representada en la Tabla 4.1. 61
4.4. Partición de una caja usando *p-partition*. 66
4.5. Valor medio para rangos usando ONVGR y HVR. 74
4.6. Valor medio para rangos usando Δ^* y ε_+. 75
4.7. ONVGR y HVR en función del tiempo para tres instancias seleccionadas. . . 76
4.8. Δ^* y ε_+ en función del tiempo para tres instancias seleccionadas. 77
4.9. Definición de características asociadas al algoritmo *TPA*. 78

6.1. Ejemplo de grafo de dependencias de módulos (MDG). 102
6.2. Valores de *MQ* y del *gap* para la instancia mtunis, en función del tiempo. . . 115

7.1. Parte cóncava y parte convexa en una caja bidimensional. 129
7.2. Analizando una caja usando *AnyAugmecon*$(2, \lambda)$ y *AnyAugmecon*$(1, \lambda)$. 131
7.3. Uso de *AnyAugmecon*$(2, \lambda)$ cuando no encontramos una nueva solución 131
7.4. Definición de *cercanía* de un punto respecto a los extremos de una caja bidimensional. 134
7.5. Porcentaje total de hipervolumen obtenido por seis algoritmos *anytime* en la instancia *nrp3*. 138
7.6. Comparando los mejores y peores algoritmos clásicos y *anytime*, para la instancia *nrp2*. 140
7.7. Comparando los mejores y peores algoritmos clásicos y *anytime*, para la instancia *nrp4*. 140
7.8. Comparando los mejores y peores algoritmos clásicos y *anytime*, para la instancia *nrp5*. 141
7.9. Comparando los mejores y peores algoritmos clásicos y *anytime*, para la instancia *nrp-e4*. 141
7.10. Comparando los mejores y peores algoritmos clásicos y *anytime*, para la instancia *nrp-m1*. 141

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Índice de tablas

2.1. Ejemplos de cota superior del tiempo computacional empleado en la resolución de ciertos problemas. 10
2.2. Comparativa de crecimiento de la función potencial y la función exponencial. 10
2.3. Algunas relaciones de orden en $\mathbb{R}^p$ con $p > 1$. 22

3.1. Parámetros asociados a los distintos métodos de escalarización. Cuando es indiferente el valor del parámetro ρ, se indica medinte un guión. 34
3.2. Algunas funciones y parámetros en CPLEX 12.6.2. 36
3.3. Cálculo de $\max_{i=1,2}(y_i - x_i)$ para cada combinación de vectores en N y PF según el ejemplo de la Figura 3.8. 45
3.4. Definición de relaciones de comparación entre conjuntos aproximados. 46
3.5. Características de las métricas usadas en esta tesis, indicado su función de interpretación. 48

4.1. Traza del Ejemplo 4.2. Problema de asignación triobjetivo. 60
4.2. Instancias usadas en los experimentos computacionales. Clase, dimensión, número de variables o agentes, número de instancias y nombre. 71
4.3. Número de veces que cada algoritmo toma el mejor valor medio, sin exclusividad (*best*) y con exclusividad (*excl*), para ONVGR y HV. 73
4.4. Número de veces que cada algoritmo toma el mejor valor medio, sin exclusividad (*best*) y con exclusividad (*excl*), para Δ^* y ε_+. 73
4.5. Tests estadísticos para cada clase y métrica considerando diferentes tiempos de ejecución. Se muestra la inicial del algoritmo en los casos en que es significativamente mejor que los otros dos con nivel de significación $\alpha = 0.01$. Si además las diferencias son significativas a nivel $\alpha = 0.001$, dicha inicial se marca en negrita. 77

5.1. Configuración de los parámetros para NSGA-II, SPEA2 y MOEA/D usando IRACE. 86
5.2. Resultados computacionales para las instancias con frente de Pareto conocido. Los mejores resultados se marcan en negrita. Los valores indeterminados se marcan con guión. 86
5.3. Resultados computacionales para las instancias con frente de Pareto desconocido. Los mejores resultados se marcan en negrita. 87
5.4. Valores medios para HVR y IGD usando *MOFeLS* para cada instancia con frente de Pareto conocido. 88
5.5. Instancias usadas en los resultados computacionales de esta sección. Las columnas indican el tipo de problema multiobjetivo, la dimensión, el número de instancias que contiene, en cuantas de ellas conocemos el frente de Pareto completo, así como los rangos del número de variables y restricciones en cada clase. 91
5.6. Análisis de resultados al ejecutar las 911 instancias y evaluar con ONVG. Cuanto mayor es el valor de '+' y mayor es el valor de $\overline{\text{Dif}}$, mejor son los resultados para *HBOXES*. 93
5.7. Análisis de resultados al ejecutar las 911 instancias y evaluar con HVR. Cuanto mayor es el valor de '+' y mayor es el valor de $\overline{\text{Dif}}$, mejor son los resultados para *HBOXES*. 94

UNIVERSIDAD DE MÁLAGA

5.8. Análisis de resultados al ejecutar las 911 instancias y evaluar con IGD+. Cuanto mayor es el valor de '−' y menor es el valor de $\overline{\text{Dif}}$, mejor son los resultados para *HBOXES*. 94
5.9. Análisis de resultados al ejecutar las 911 instancias y evaluar con GD. Cuanto mayor es el valor de '−' y menor es el valor de $\overline{\text{Dif}}$, mejor para *HBOXES*. . . 95

6.1. Instancias consideradas para el problema *SMC*. La columna *Links* determina el número de relaciones entre módulos considerando el grafo MDG dirigido. . 112
6.2. Cota superior del tiempo en el cual el resolutor no ha sido capaz de continuar la ejecución hasta las 8 horas debido a un desbordamiento de memoria al considerar 8 GB de RAM para las distintas instancias, en función de los valores del parámetro CPX_PARAM_MIPEMPHASIS. 113
6.3. Resultados computacionales para el problema *SMC* tras un máximo de 48 horas de ejecución utilizando el algoritmo que usa programación matemática y el que usa ramificación y poda (*BB*). 114
6.4. Restricciones a añadir al problema *SMC* cuando unimos nodos a un mismo subsistema, o forzamos a que estén separados. 116

7.1. Valores objetivo promedio para los tres métodos propuestos, usando instancias generadas aleatoriamente durante 60 segundos y con presupuesto $b = 0.3\sum_{i=1}^{n} c_i$. 126
7.2. Valores objetivo promedio para los tres métodos propuestos, usando instancias generadas aleatoriamente durante 60 segundos y con presupuesto $b = 0.7\sum_{i=1}^{n} c_i$. 127
7.3. Test de Wilcoxon (datos apareados) para contrastar si hay diferencias entre los algoritmos propuestos, $b = 0{,}3\sum_{i=1}^{n} c_i$. 127
7.4. Test de Wilcoxon (datos apareados) para contrastar si hay diferencias entre los algoritmos propuestos, $b = 0{,}7\sum_{i=1}^{n} c_i$. 128
7.5. Contrastes unilaterales con hipótesis alternativa $H_1 : cmsa_s > col$, siendo $col \in \{exact, cmsa_r\}$. 128
7.6. Número de requisitos y clientes para cada conjunto de datos en el *NRP* biobjetivo. 135
7.7. Número de puntos del frente de Pareto y comparativa del número de soluciones soportadas encontrado por *SPF* y por *ADS*. 137
7.8. Resultados para los algoritmos *anytime* (excluyendo *SPF*) para 60 segundos de tiempo límite. Los mejores porcentajes obtenidos para cada instancia se marcan en negrita. 138
7.9. Resultados para las instancias clásicas del *NRP* durante 60 segundos de tiempo límite. 139
7.10. Porcentaje aproximado de diferencia entre porcentajes de hipervolumen obtenido entre el mejor algoritmo clásico y el peor algoritmo *anytime*. Las diferencias positivas indican una ventaja de estos últimos. 140
7.11. Test *post-hoc* de Nemenyi usando comparaciones múltiples. 140

UNIVERSIDAD DE MÁLAGA