Algoritmos de búsqueda con retroceso para problemas multicriterio

TESIS DOCTORAL

Javier Coego Botana

Universidad de Málaga

23 de Octubre de 2015

Publicaciones y Divulgación Científica
UNIVERSIDAD DE MÁLAGA

AUTOR: Javier Coego Botana
http://orcid.org/0000-0003-3393-8604

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

Algoritmos de búsqueda con retroceso para problemas multicriterio

Memoria que presenta el doctorando

Javier Coego Botana

para optar al grado académico de Doctor Ingeniero en Informática

Dirigida por el Doctor

Lorenzo Mandow Andaluz

Tribunal de la tesis

Rafael Morales Bueno - Universidad de Málaga

María Victoria Belmonte Martínez - Universidad de Málaga

Camino Rodríguez Vela - Universidad de Oviedo

Eva Onaindía de la Rivaherrera - Universidad Politécnica de Valencia

Raquel Fuentetaja Pizán - Universidad Carlos III de Madrid

Universidad de Málaga

23 de Octubre de 2015

E-MAIL: jcoego@lcc.uma.es

Este trabajo está financiado por:

Consejería de Economía, Innovación, Ciencia y Empresa.

Junta de Andalucía (España)

Referencia: P07-TIC-03018

El Dr. D. Lawrence Mandow Andaluz, Profesor Titular de Universidad, del Área de Ciencias de la Computación e Inteligencia Artificial de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Málaga ,

Certifica que,

D. Javier Coego Botana, Licenciado en Informática, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga , bajo su dirección, el trabajo de investigación correspondiente a su Tesis Doctoral titulada:

Algoritmos de búsqueda con retroceso para problemas multicriterio

Revisado el presente trabajo, estima que puede ser presentado al tribunal que ha de juzgarlo, y autoriza la presentación de esta Tesis Doctoral en la Universidad de Málaga .

Fdo.: Dr. Lawrence Mandow Andaluz

Málaga, 23 de Octubre de 2015

A Nieves y a nuestros hijos, Dani y Javi.

Agradecimientos

Finalmente, muchos años después de zarpar, este barco llega ya a puerto. Unas veces ha sido un crucero, otras el barco del holandés errante y en ocasiones un viaje en galeras. Pero siempre he tenido gente a mi lado que se merece un reconocimiento por haber remado altruistamente conmigo, compartiendo esta singladura.

En primer lugar y por encima de todo, gracias a Lawrence Mandow, capitán, brújula y director, por haberme regalado esta tesis, pues es más suya que mía. Gracias por haber recogido a este náufrago del proceloso mar del Doctorado; por ser maestro y compañero antes que director; por su luz cuando recorríamos caminos de investigación un tanto oscuros; por ser el timonel que ha corregido la derrota del barco cuando iba camino de ser pecio; por haber tirado del carro cuando yo me empeñaba en procrastinar; por sus buenos consejos; por su buen humor y paciencia. Y por un millón de cosas más, GRACIAS.

Gracias a José Luis Pérez de la Cruz, teórico, analítico, pragmático, matemático, catedrático y británico. Ha ejercido de padrino de esta criatura, velando por su buen rumbo y aportando esa flema y capacidad de resolución que tantas veces me han dejado con la boca abierta.

Gracias a mis contemporáneos de tesis, los doctores Enrique Machuca y Francisco Javier Pulido, por su ayuda desinteresada siempre que me ha hecho falta, compartiendo información y trucos, y regalándome la *luna* cuando se la he pedido. Gracias y buena suerte en vuestros viajes.

Gracias a mis compañeros de trabajo de estos últimos años en la Junta de Andalucía y en la UMA, que han sabido tolerar, y muchas veces fomentar, mis cada vez más frecuentes distracciones doctorales, sobre todo en este tramo final.

Gracias a mis grandes amigos en las tres grandes etapas de mi vida: Coruña, Madrid y Málaga. En especial a Darío, Justo, Mari Cruz, Álvaro, Chelo e Isa, todos ellos condiciones necesarias y suficientes para que yo haya llegado donde estoy ahora. En mayor o menor medida habéis sido testigos de esta deriva y el apoyo y ánimo que me habéis brindado han hecho más llevadero este largo camino.

Gracias a mi familia de Málaga, por haber sido mi Norte en este Sur. En especial a José y Nieves, por haberme acogido y tratado todos estos años como si fuera un hijo más.

Gracias infinitas a mis padres, por tantos esfuerzos durante tanto tiempo, por darlo

todo, por privarse ellos para que tuviera yo y por haberme inculcado valores a los cuales espero haber sido fiel. Y como no, a mi hermano Jose, mi Gran Hermano, por ser el espejo de honestidad, trabajo y humanidad en el que me quiero ver reflejado.

Gracias a Nieves, la mujer del millón de sonrisas, Doctora en Paciencia y Catedrática en Psicología de Tercer Ciclo: por haber estado siempre a mi lado, no sólo en este camino académico, sino en otro aún más largo y difícil que es la vida, soportándome, quitando piedras del camino y echándose a la espalda cualquier distracción. Mil párrafos no le harían justicia.

Gracias a mis hijos, Dani y Javi, por aguantar tantos *"ahora no puedo"* o *"estoy muy liado"*, e incluso gruñidos, con su inagotable alegría, su resignación y sus muchos besos. Aunque vuestras respectivas felicidades me hayan salido como funciones a maximizar, tened por seguro que resolveré este problema biobjetivo. En esta vida os ha tocado un padre *NP-hard* de aguantar, pero prometo compensaros con tiempo de juego exponencial en plazo polinómico.

Resumen

La búsqueda en grafos, con multitud de aplicaciones en el mundo real, ha propiciado el diseño de una gran cantidad de algoritmos centrados en el procesamiento de un único objetivo, magnitud representativa del coste. Sin embargo, un tratamiento realista de estos problemas requiere en muchas ocasiones contemplar diferentes objetivos de modo simultáneo. Además, es habitual que estos objetivos sean antagónicos, de tal modo que la optimización de uno de ellos se traduzca en el empeoramiento de uno o varios de los objetivos restantes. Esto hace que el coste óptimo no sea único, sino que generalmente existe un conjunto de soluciones óptimas cuyas componentes de coste están compensadas entre sí.

Esta naturaleza multiobjetivo de los problemas provoca que el rendimiento de los algoritmos empeore de modo considerable, ya que al procesamiento habitual de los nodos generados durante el proceso de búsqueda hay que añadir el tratamiento de vectores de coste (de dimensión igual al número de objetivos considerado) y el manejo de un conjunto de soluciones óptimas (cuyo tamaño en el peor de los casos será exponencial), siendo este tipo de operaciones muy costosas desde el punto de vista de tiempo y memoria.

De las dos principales clases de algoritmos exactos multiobjetivo, la correspondiente a un enfoque *best-first* ha sido ampliamente estudiada, dando lugar a una gran cantidad de algoritmos que persiguen reducir la complejidad espacial y temporal del proceso de búsqueda. Asimismo existen numerosas y detalladas comparativas de rendimiento entre estos algoritmos. Sin embargo la clase de algoritmos *depth-first*, aún siendo de gran utilidad en la resolución de problemas con grafo de búsqueda en forma de árbol, presentaba un reducido número de propuestas, careciendo además de análisis comparativos entre las mismas.

Esta tesis pretende cubrir dicho hueco, realizando un estudio sistemático de algoritmos exactos multiobjetivo de tipo *depth-first*. Para ello, por un lado se realiza una caracterización detallada de dichos algoritmos, contribuyendo con nuevos diseños multiobjetivo, variantes de algoritmos para un objetivo que aplican los conceptos de *Frontera de Pareto* y *Punto Ideal* a la cota de expansión empleada en la búsqueda

depth-first. De todos estos nuevos algoritmos se aporta su correspondiente análisis formal. Asimismo se realizan adaptaciones de algoritmos multiobjetivo ya existentes, de tal modo que sean aplicables a árboles infinitos, el principal tipo de problemas usado en el análisis de rendimiento realizado en este trabajo.

Por otro lado, en esta tesis se realiza un análisis empírico detallado de los algoritmos mencionados. Para ello en primer lugar se determinan los principales parámetros de rendimiento a tener en cuenta en función de la naturaleza multiobjetivo de los problemas considerados. Concretamente el tiempo de ejecución, en combinación con el tamaño de la cota de expansión, se perfilan como los factores clave de rendimiento. Los resultados obtenidos en las pruebas realizadas presentan a la variante multiobjetivo del algoritmo *Branch & Bound* como la opción más eficiente para árboles de búsqueda tanto finitos como infinitos, precisando de una cota adicional en este último caso.

Índice

Índice de figuras

Índice de Tablas

Parte I

Alcance y fundamentos

Esta parte presenta el alcance, objetivos y contribuciones de esta tesis, incluyendo también una descripción de los conceptos básicos de la búsqueda para un objetivo y multiobjetivo, así como los diferentes algoritmos de tipo *depth-first* ya desarrollados para este tipo de problemas. Concretamente se describen los algoritmos $IDMOA^*$, $MOMA^*0$ y *DF-BnB* multiobjetivo.

- El capítulo 1 presenta una introducción a los contenidos, objetivos y contribuciones de esta tesis.
- El capítulo 2 presenta los aspectos básicos de la búsqueda para problemas de un objetivo, así como las categorías de los algoritmos más utilizados habitualmente en este tipo de problemas.
- El capítulo 3 presenta los aspectos básicos de la búsqueda multiobjetivo. Asimismo presenta las generalizaciones para el caso multiobjetivo de los algoritmos exactos de tipo *depth-first* más representativos.

Capítulo 1

Introducción

El contexto de esta tesis es un problema ampliamente tratado en las áreas de la Inteligencia Artificial y la Investigación Operativa: la búsqueda en grafos multiobjetivo. La búsqueda del camino más corto es un problema que se puede encontrar en muchas situaciones del mundo real, desde el cálculo de rutas entre dos puntos de un mapa hasta el diseño de circuitos VLSI. La gran mayoría de algoritmos diseñados para este tipo de problemas contemplaban originalmente la existencia de un único objetivo a considerar. Por ejemplo, en el cálculo de rutas se tendía únicamente en cuenta la distancia a recorrer o el tiempo como factor a minimizar. Con el paso del tiempo, la naturaleza de estos problemas cambió sustancialmente, al comprobarse que un enfoque realista de los mismos requería el tratamiento simultáneo de varios objetivos. Así, por ejemplo, en el cálculo de rutas puede ser igualmente interesante minimizar además el coste económico asociado a la ruta calculada. La complejidad de estos problemas multiobjetivo hizo necesario el diseño de nuevos algoritmos, los cuales, en muchos casos, no dejaban de ser variantes de sus equivalentes para un único objetivo. El objetivo general de esta tesis es profundizar en el análisis de la familia de algoritmos exactos de búsqueda multiobjetivo de tipo *depth-first*, aportando nuevas técnicas de búsqueda y realizando además una evaluación exhaustiva de estos nuevos algoritmos y otros ya existentes.

En la sección 1.1 de este capítulo se destaca la motivación de este trabajo. El alcance del mismo se presenta en la sección 1.2. Los objetivos y contribuciones de la tesis se resumen en las secciones 1.3 y 1.4 respectivamente. Las publicaciones derivadas de esta investigación se enumeran en la sección 1.5. Por último un resumen de la estructura de esta memoria aparece en la sección 1.6.

1.1. Motivación y relevancia

La búsqueda del camino más corto ha sido uno de los problemas más ampliamente estudiados en el ámbito de la Inteligencia Artificial y la Investigación Operativa en su variante para un único objetivo. La resolución de estos problemas radica en localizar,

dentro de un grafo, el camino con menor coste entre un nodo inicial y uno o varios nodos finales. El coste está asociado a los arcos del grafo, de modo que el coste de un camino corresponde a la suma de los costes de sus arcos. Por ejemplo, en los mapas de carreteras el coste puede reflejar la cantidad de kilómetros de una ruta o el coste económico asociado a dicha ruta (p.ej. combustible y peajes).

Tradicionalmente han existido dos grandes clases de algoritmos para tratar este tipo de problemas: los algoritmos exactos, que localizan la solución óptima del problema (es decir, aquella con el menor coste posible), y los algoritmos aproximados, que localizan soluciones que pueden ser subóptimas, aunque idealmente tienen una calidad alta. Asimismo, dentro de los algoritmos exactos se pueden distinguir también dos grandes bloques: los algoritmos *best-first* y los algoritmos *depth-first*. Los algoritmos *best-first* mantienen un árbol de búsqueda en memoria donde se guardan todos los caminos de interés explorados hasta el momento, expandiendo siempre la rama más prometedora. Los algoritmos *depth-first*, por el contrario, únicamente mantienen en memoria el camino que se está procesando, con lo cual la reconsideración del camino a explorar puede dar lugar a una reexpansión de nodos en ramas que fueron anteriormente examinadas.

Sin embargo muchos problemas reales precisan del tratamiento simultáneo de varios objetivos, siendo estos objetivos, en la mayoría de los casos, antagónicos. Si en el tradicional problema del mapa de carreteras contemplamos como objetivos a minimizar la distancia recorrida y el coste económico, la mejora de una componente suele conllevar el empeoramiento de la otra. Es por ello que los problemas multiobjetivo no suelen tener una única solución óptima, sino un conjunto de soluciones óptimas, denominadas óptimos de Pareto.

Una forma de abordar los problemas multiobjetivo es emplear algoritmos para un objetivo, incorporando un preprocesamiento previo de los objetivos: p.ej. combinación lineal de los mismos o minimización de uno de los objetivos aplicando restricciones a los valores del segundo. Estos enfoques, sin embargo, no siempre garantizan que se pueda encontrar cualquier solución Pareto-óptima y pueden en ocasiones ser ineficientes. Se ha propuesto también una alternativa denominada $IDMOA^*$ (Harikumar y Kumar, 1996) que aplica la técnica de profundización iterativa para procesar *secuencialmente* los objetivos considerados. Esta técnica permite encontrar todas las soluciones óptimas de Pareto.

Los algoritmos multiobjetivo exactos, como $IDMOA^*$, devuelven el conjunto completo de soluciones óptimas del problema. Por este motivo el coste computacional en el caso multiobjetivo es mayor que en caso de un objetivo, tanto a nivel de tiempo como de memoria consumida. Esto hace necesario un refinamiento de los algoritmos para reducir de algún modo el alto coste de procesamiento.

Los algoritmos multiobjetivo de tipo *depth-first* reducen la cantidad de memoria usada durante el proceso de búsqueda. Sin embargo, tal y como se verá a lo largo de este trabajo, las principales ineficiencias que presentan son la reexpansión de nodos durante la generación del árbol de búsqueda y el procesamiento de los vectores de coste asociados a dichos nodos.

Existen trabajos previos que realizan análisis y evaluaciones de algoritmos de búsqueda tanto *best-first* como *depth-first*, pero la gran mayoría se centran en el estudio del caso de un único objetivo (Vempaty et al., 1991) (Zhang y Korf, 1993) (Zhang y Korf, 1995). En (Zhang, 1999) el análisis realizado sobre los algoritmos *depth-first* se basa en una clasificación de los mismos que comprende las estrategias de búsqueda *DF-BnB* (*Depth-First Branch & Bound*), profundización iterativa y búsqueda recursiva *best-first*.

El principal objetivo de esta tesis es realizar un análisis sistemático de los diferentes enfoques *depth-first* en la búsqueda multiobjetivo exacta, basándonos en la clasificación ya mencionada de los diferentes algoritmos (Zhang, 1999). Concretamente se analiza la idoneidad de diferentes parámetros de rendimiento y se realiza una evaluación de los algoritmos fundamentalmente sobre problemas con un espacio de estados infinito, ámbito en el cual los algoritmos *depth-first* son más susceptibles de ver degradado su rendimiento, e incluso algunos no pueden garantizar su finalización.

Este análisis sistemático implica por una parte la revisión y evaluación de los algoritmos *depth-first* multiobjetivo propuestos en la literatura. Por otra parte, nuestro trabajo incluye el diseño de nuevos algoritmos de búsqueda multiobjetivo *depth-first*, destacando los basados en el concepto de *Punto Ideal*, así como el refinado de otros algoritmos ya existentes, mejorando notablemente su rendimiento.

1.2. Alcance

El alcance de esta tesis se puede definir en base a los siguientes términos:

Optimalidad de Pareto El tratamiento simultáneo de diferentes objetivos en un problema de búsqueda cambia radicalmente el concepto de optimalidad de las soluciones. No existe un coste mínimo, sino que el problema puede tener diferentes soluciones óptimas en las cuales los costes de los objetivos estén compensados entre sí.

Algoritmos exactos *depth-first* Los algoritmos evaluados, tanto los ya existentes como los aportados en este trabajo, son del tipo *depth-first* y recuperan el conjunto completo de soluciones óptimas.

Punto Ideal La aplicación del concepto de *Punto Ideal* en nuevos algoritmos multiobjetivo desarrollados en esta tesis permite reducir la alta cantidad de comparaciones vectoriales que se realizan en los enfoques *depth-first*, debidas éstas últimas en gran parte al uso de una cota de expansión multivector.

Análisis empírico A pesar de existir diferentes algoritmos multiobjetivo *depth-first* propuestos en la literatura, no se disponía hasta la fecha, que tengamos constancia, de una evaluación conjunta y exhaustiva de un grupo significativo de los mismos.

Árboles de búsqueda infinitos Algunos algoritmos de tipo *depth-first* (concretamente los que no emplean una cota de expansión actualizable durante la búsqueda) pueden presentar problemas de completitud cuando se enfrentan a problemas con un espacio de estados infinito. Incluso un algoritmo como *DF-BnB*, que sí realiza actualizaciones de cota, puede no ser capaz de localizar una solución base que le permita ir podando el árbol de búsqueda. Es por ello que las variantes con iteración basadas en cota son las más adecuadas para este tipo de problemas.

Parámetros de rendimiento de algoritmos multiobjetivo El procesamiento de los nodos durante la exploración de un grafo multiobjetivo difiere radicalmente de la realizada en las búsquedas con un objetivo. Las comparaciones de coste para determinar la expansión de un nodo se realizan entre vectores de tamaño igual al número de objetivos considerado. Además, muchos algoritmos usan cotas de expansión multivector. Esto hace que una medida de rendimiento ampliamente aceptada en el caso de un objetivo, el número de nodos expandidos, no sea la adecuada para valorar la calidad de algoritmos de búsqueda multiobjetivo.

1.3. Objetivos de la investigación

El objetivo general de esta tesis es un análisis sistemático de las distintas variantes de algoritmos exactos *depth-first* aplicados al problema de la búsqueda multiobjetivo. Para la consecución del mismo se han considerado los siguientes subobjetivos:

Diseño de nuevos algoritmos multiobjetivo *depth-first* Uno de los principales objetivos de esta tesis es ahondar en el diseño de nuevos algoritmos *depth-first*, área en la cual las soluciones diseñadas hasta la fecha no aportan rendimientos destacables. Para ello se introducen mejoras en variantes multiobjetivo de la profundización iterativa y la búsqueda *best-first* recursiva. Además se explora una nueva línea de algoritmos basados en el concepto de *Punto Ideal*, que permite mitigar uno de los grandes cuellos de botella de los actuales algoritmos iterativos: la masiva comparación de vectores de coste durante la generación del árbol de búsqueda.

Análisis formal de los algoritmos Dado que el eje central de esta tesis son los algoritmos multiobjetivo exactos de tipo *depth-first*, es necesario demostrar su completitud (incluso en el procesamiento de espacios de estados infinitos), así como su admisibilidad (deben devolver el conjunto completo de óptimos de Pareto).

Adaptación de algoritmos actuales Existen algoritmos como *DF-BnB* cuyo rendimiento es netamente superior a otros algoritmos multiobjetivo *depth-first*. Sin embargo *DF-BnB* puede presentar incompletitud (es decir, el algoritmo puede no finalizar) cuando trata de resolver problemas con espacios de estados infinitos. Debido a ello es necesario adaptar este algoritmo para que reciba una cota

superior inicial que le permita finalizar el proceso de búsqueda en cualquier caso. Asimismo es de interés el análisis del impacto de la calidad de dicha cota en el rendimiento final.

Evaluación empírica La cantidad de algoritmos multiobjetivo exactos *depth-first* incluídos en este trabajo hace necesaria una recopilación y evaluación exhaustiva del rendimiento de los mismos, previo establecimiento de los parámetros de rendimiento más significativos para dicho análisis.

1.4. Contribuciones de la tesis

Las principales contribuciones de esta tesis son las siguientes:

Categorización de los algoritmos multiobjetivo exactos *depth-first* En el ámbito de la búsqueda multiobjetivo se han realizado diversos esfuerzos en el diseño de nuevos algoritmos, pero se echa en falta un estudio detallado que englobe y analice el conjunto de estos algoritmos. En esta tesis se realiza una categorización de los diferentes algoritmos multiobjetivo exactos de tipo *depth-first*, tanto los preexistentes como los obtenidos en este trabajo. Se distinguen tres grandes grupos de algoritmos: los basados en profundización iterativa, los basados en búsqueda recursiva *best-first* y un tercer grupo que engloba los algoritmos secuenciales o por fases.

Diseño de nuevos algoritmos Los algoritmos de tipo *depth-first* se han revelado como una alternativa eficiente en la resolución de problemas de búsqueda en árboles. Sin embargo en general ven penalizado su rendimiento en este tipo de problemas como consecuencia de la reexpansión de partes del árbol de búsqueda. Este comportamiento es intrínseco a la expansión basada en profundización progresiva, al almacenarse en memoria únicamente la rama del árbol que está siendo explorada. En el caso multiobjetivo estas reexpansiones repercuten de modo mucho más negativo, al ser mayores los tiempos de procesamiento de un nodo. Es por ello que se hace necesario disminuir bien el número de nodos reexpandidos, bien el tiempo de procesamiento de cada uno de los mismos. Esto es lo que se persigue con el diseño de los nuevos algoritmos multiobjetivo *depth-first* incluídos en esta tesis: $LEXIDMOA^*$, $PIDMOA^*$, $IPID^*$, *Pareto-MO-RBFS* e *IP-MO-RBFS*. Los tres primeros están basados en el paradigma de profundización iterativa, manteniendo cotas de expansión por iteración. Los dos últimos algoritmos están basados en *RBFS*, empleando cotas de expansión calculadas para cada nodo. Estas cotas son multivectoriales en el caso de $PIDMOA^*$ y *Pareto-MO-RBFS*, lo cual reduce el número de reexpansiones al realizar un avance más rápido sobre el árbol de búsqueda. En los algoritmos $LEXIDMOA^*$, $IPID^*$ e *IP-MO-RBFS* la cota está formada por un único vector, lo que reduce el

tiempo de comparación de los diferentes nodos con dicha cota. Dos de estos algoritmos ($IPID^*$ e $IP\text{-}MO\text{-}RBFS$) introducen el concepto de *Punto Ideal*. Estos algoritmos tienen como principal característica el uso de una cota de expansión formada por un único vector, actualizable durante las diferentes iteraciones, que pondera equitativamente todos los objetivos. Esta sencilla cota permite reducir en gran medida el número de operaciones de comparación de los nodos generados contra la misma a costa de incrementar en parte el número de nodos considerados, mejorando el rendimiento de modo notable frente a opciones similares como $LEXIDMOA^*$. Para todos los algoritmos comentados se aportan demostraciones formales tanto de su completitud como de su admisibilidad.

Análisis de rendimiento No hay estudios de los que tengamos constancia que realicen un análisis comparativo detallado de los algoritmos multiobjetivo exactos. En esta tesis hemos optado, dado su volumen, por restringir este estudio a los algoritmos de tipo *depth-first*. Ha sido necesario previamente determinar la bondad de los diferentes parámetros de rendimiento a tener en cuenta, pues la naturaleza de los problemas multiobjetivo hace que los análisis basados en número de nodos procesados, algo habitual en el caso para un objetivo, no sea totalmente significativa. La evaluación de los algoritmos se ha realizado empleando fundamentalmente espacios de estados infinitos, analizando el rendimiento en cada una de las categorías anteriormente mencionadas. Las variantes multiobjetivo basadas en *DF-BnB* han demostrado ser las más eficientes, siempre y cuando incluyan una cota inicial que permita asegurar la completitud del algoritmo.

1.5. Publicaciones

Algunas de las contribuciones presentadas en esta tesis han sido publicadas en conferencias y revistas internacionales:

- **Revistas:**
  - Coego, J., Mandow, L. y Pérez de la Cruz, J. L. A comparison of multiobjective depth-first algorithms. *Journal of Intelligent Manufacturing*, 2010
- **Conferencias:**
  - Coego, J., Mandow, L. y Pérez de la Cruz, J. L. A new approach to iterative deepening multiobjective A*. En *AI*IA 2009, LNCS 5883*, páginas 264–273. 2009
  - Coego, J., Mandow, L. y Pérez de la Cruz, J. L. Ideal point guided iterative deepening. En *ECAI 2012, LNCS 5883*, páginas 246–251. 2012
- **Doctoral Consortium:**

- Coego, J. Backtracking search algorithms for multicriteria problems. En *Doctoral Consortium, XIII Conference of the Spanish Association for Artificial Intelligence (CAEPIA'09)*. 2009

1.6. Estructura de la tesis

Esta tesis está estructurada en nueve capítulos agrupados en tres partes. La primera parte incluye este capítulo introductorio y los capítulos 2 y 3. El capítulo 2 presenta los aspectos básicos de la búsqueda en grafos para el caso de un objetivo, con especial énfasis en los algoritmos *depth-first*, mientras que el capítulo 3 extiende la búsqueda anterior a problemas con varios objetivos, incluyendo categorización de algoritmos, una explicación detallada de un representante de cada categoría y algunas consideraciones acerca del rendimiento en la búsqueda multiobjetivo.

La segunda parte agrupa todas las contribuciones fundamentales presentadas en esta tesis, abarcando los capítulos del 4 al 8. En el capítulo 4 se analiza la búsqueda con profundización iterativa y se presentan tres nuevos algoritmos ($LEXIDMOA^*$, $PIDMOA^*$ e $IPID^*$) basados en este paradigma. La diferencia de estos algoritmos con propuestas ya existentes radica en el procesamiento de las diferentes cotas de expansión vectoriales calculadas para cada una de las iteraciones realizadas durante el proceso de búsqueda.

En el capítulo 5 se analiza la búsqueda *best-first* recursiva ($RBFS$) y se presentan dos nuevos algoritmos ($Pareto\text{-}MO\text{-}RBFS$ e $IP\text{-}MO\text{-}RBFS$), que generalizan el concepto de $RBFS$ al caso multiobjetivo. De modo análogo a los algoritmos anteriores, el tratamiento que realizan de la cota de expansión es la que determina el mejor rendimiento de estas nuevas propuestas con respectos a otros algoritmos ya existentes.

El capítulo 6 presenta variantes del algoritmo $DF\text{-}BnB$ multiobjetivo para conseguir que éste sea completo aún tratando con espacios de estados infinitos. El capítulo 7 incluye las demostraciones formales de completitud y admisibilidad de los nuevos algoritmos presentados en este trabajo.

En el capítulo 8 se presenta el entorno de pruebas usado para la evaluación del conjunto de algoritmos, así como un análisis de los parámetros de rendimiento más adecuados atendiendo a la naturaleza especial del caso multiobjetivo, donde el procesamiento de un nodo implica una mayor dificultad que en el caso de un solo objetivo. Finalmente se comentan los diferentes resultados obtenidos de las pruebas.

La última parte de esta tesis resume en el capítulo 9 las principales conclusiones de este trabajo de investigación, así como posible trabajo futuro.

Capítulo 2

Búsqueda con un objetivo

La búsqueda heurística aplicada a problemas de camino mínimo (*Shortest Path*) ha sido, desde hace muchos años, un tema central de estudio en el área de la Inteligencia Artificial. Muchas situaciones reales pueden modelarse en forma de espacios de estados. Por ejemplo, el juego del Puzle-8 representa cada posible combinación de fichas en el puzle mediante un estado distinto y los diferentes movimientos válidos mediante un conjunto de operadores. Un problema definido por un espacio de estados consiste en localizar, a partir de un estado inicial (en nuestro ejemplo, una configuración de las fichas en el puzle), una serie de operadores que nos permitan alcanzar un estado final (otra configuración de las fichas). Este problema es equivalente a un grafo en el cual los nodos son los diferentes estados y las transiciones o arcos entre nodos están definidas por los diferentes operadores. Por lo tanto resolver el problema consiste en localizar en el grafo un camino entre los nodos que representan los estados inicial y final, pasando por nodos (estados) intermedios mediante la aplicación de operadores. Por lo general, es de interés que este camino sea el más corto posible. La longitud de los caminos se ha medido normalmente en términos de una función de coste que combina los costes escalares de los arcos que componen el camino. El caso más frecuente y más estudiado es el de la combinación aditiva de los costes de los arcos.

La formalización mediante espacios de estados ha propiciado el desarrollo de algoritmos genéricos de búsqueda. Las diferencias entre unos algoritmos y otros radican fundamentalmente en el rendimiento del proceso de búsqueda en términos de tiempo y memoria y en la naturaleza de las soluciones buscadas. Hay algoritmos que localizan una solución cualquiera al problema, otros que calculan todas las soluciones posibles y otros, los más habituales, que calculan la mejor solución, aquella con menor coste.

Este capítulo presenta un detallado resumen de todo lo concerniente a la búsqueda en problemas que involucran a un único objetivo. En la sección 2.1 se mostrará la formalización de problemas como espacios de estados, así como la equivalencia de los mismos con grafos. La sección 2.2 presenta un algoritmo general de búsqueda sobre un espacio de estados. A continuación, la sección 2.3 define una serie de propiedades a tener en cuenta en la evaluación de algoritmos de búsqueda y la sección 2.4 presenta

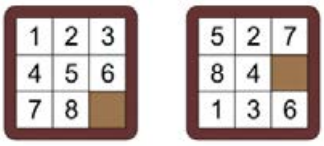


Figura 2.1: Problema del Puzle 8

una de las clasificaciones de algoritmos más habituales, la basada en el consumo de memoria. La sección 2.5 explica el uso de información heurística en el proceso de búsqueda para optimizar el rendimiento de los algoritmos. Las secciones 2.6 y 2.7 presentan los algoritmos más comunes de cada clase, profundizando más en los que presentan un consumo lineal. Por último, en la sección 2.8 se realizará un pequeño resumen del rendimiento de los algoritmos comentados.

2.1. Formalización de problemas

Muchos de los problemas tratados en el área de la Inteligencia Artificial se caracterizan por carecer de una algoritmia específica para su resolución. Esto ha propiciado el desarrollo de técnicas y algoritmos genéricos los cuales precisan de una formalización previa de situaciones reales. La formalización mediante **espacios de estados** es una de las técnicas más empleadas.

Tal y como se adelantaba en la introducción del presente capítulo, un problema sobre un espacio de estados es equivalente a un grafo en el cual hay que encontrar una secuencia de acciones u operadores que permitan llegar a una situación (estado, nodo) final deseada partiendo de una situación inicial determinada. Por ejemplo, en el problema del Puzle-8 representado en la figura 2.1, la situación inicial es el tablero representado a la derecha de la imagen, y la situación final deseada es el tablero de la izquierda de la imagen. Para resolver el problema es necesario aplicar una serie de operadores hasta llegar a la situación final. En nuestro caso, estas operaciones son los diferentes movimientos de las fichas por el tablero, que consisten en intercambiar el hueco con una ficha adyacente.

Un problema definido en forma de espacio de estados se caracteriza por la siguiente información:

- Un conjunto de estados, cada uno de los cuales representa la situación del problema en un instante determinado.
- Un conjunto de reglas u operadores, las cuales nos permite pasar de una situación del problema a otra. Generalmente estos operadores tienen asignado un coste en función de su complejidad.

Dentro del conjunto de estados se incluyen algunos con una significación especial: el estado inicial del problema (la situación de partida) y el estado o estados finales del

problema (la situación deseada). En el caso del problema del Puzle-8 de la figura 2.1, los estados inicial y final podrían ser las imágenes derecha e izquierda respectivamente. En el caso del ajedrez, por ejemplo, el estado inicial es único pero puede haber diferentes estados finales. El conjunto de estados del problema viene dado por cualquier combinación de fichas en el Puzle-8 y los operadores describen los diferentes movimientos de las fichas por el tablero. Los operadores deben cumplir ciertas precondiciones para ser aplicados. Por ejemplo, en el tablero de la derecha de la figura 2.1, la ficha 5 no puede moverse hacia la izquierda, ya que la situación resultante no sería válida.

Una de las principales ventajas de la formalización como espacio de estados es la equivalencia de este espacio con un grafo con las siguientes características:

- Un conjunto de nodos N, que representan los diferentes estados por los que puede transitar un problema.
- Un conjunto de nodos $S \subseteq N$ que representa los estados iniciales del problema. Generalmente (y así lo contemplaremos en esta tesis) este conjunto S está formado por un único estado.
- Un conjunto de nodos $\Gamma \subseteq N$ que representan los estados finales o nodos meta[1] del problema
- Un conjunto A de arcos $(n_i, n_j) \mid n_i, n_j \in N$ que conectan pares de nodos, de tal modo que existe un arco (n_i, n_j) si en el espacio de estados hay un operador tal que al aplicarlo al estado n_i pasamos al estado n_j.
- Una función de coste $c(n_i, n_j), n_i, n_j \in N$, cuyo dominio es el conjunto $A(N \times N)$ y que devuelve el coste del operador necesario para ir del estado n_i al estado n_j.

Dependiendo de la naturaleza del problema, los grafos equivalentes resultantes pueden ser:

- No dirigidos: si para todo arco (n, n') del grafo existe el arco recíproco (n', n). Es el caso del Puzle-8.
- Dirigidos: si existen arcos para los cuales no hay recíproco. Es el caso del ajedrez, donde, por ejemplo, el operador *Mover peón una casilla hacia delante* no tiene recíproco, al no poder moverse el peón hacia atrás.
- Dirigidos acíclicos: si en el grafo no existen ciclos. Por ejemplo, consideremos el problema de las 8-reinas, consistente en colocar las 8 reinas en un tablero de ajedrez de tal modo que ninguna reina esté amenazada por otra. Si consideramos una formalización en la que los operadores consisten en colocar reinas en

[1] A efectos de nomenclatura, en esta Tesis se considerará *nodo hoja* todo aquel nodo terminal (sin sucesores) del grafo que no representa un estado final del problema y *nodo final* o *nodo meta* todo aquel nodo (habitualmente terminal) que sí representa un estado final del problema.

el tablero, no en quitarlas ni en moverlas, no pueden darse ciclos. En cambio sí podrían darse estados repetidos durante la búsqueda (p.ej. la secuencia de operadores *Reina 1 a Casilla 1-1*, *Reina 2 a Casilla 2-3* tiene como estado resultante el mismo que la secuencia *Reina 2 a Casilla 2-3*, *Reina 1 a Casilla 1-1*.

- Árbol: si el grafo no tiene ciclos y a cada nodo sólo se puede llegar a través de un único camino. Sería el caso del problema de las 8-reinas, si obligamos a que antes de colocar la reina de la fila n estén colocadas las reinas de las $n-1$ filas anteriores.

Una instancia de un problema sobre un espacio de estados viene definida por un estado inicial del problema y un conjunto de estados finales. Resolver una instancia de un problema consiste en encontrar una secuencia de operadores que permitan pasar desde el estado inicial al estado final (o uno de los estados finales en el caso de que hubiera varios). De modo análogo, resolver un problema en un grafo es encontrar una secuencia de arcos o camino $P = (s, n_1, n_2, \ldots, n_{k-2}, n_{k-1}, \gamma)$ que va desde un nodo inicial $s \in S$ hasta un nodo final $\gamma \in \Gamma$.

Definición 2.1.1. *Dado un camino $P = (s, n_1, \ldots, n_{k-1}, \gamma)$ se calcula su coste $g(P)$ como la suma de los costes de los arcos que atraviesa, es decir,*

$$g(P) = c(s, n_1) + c(n_1, n_2) + \ldots + c(n_{k-2}, n_{k-1}) + c(n_{k-1}, \gamma). \tag{2.1}$$

La forma del espacio de estados tiene una repercusión directa sobre el rendimiento y el resultado final de los algoritmos de búsqueda empleados. Si un algoritmo no comprueba la existencia de ciclos y el problema es un grafo no dirigido, el algoritmo podría caer en un ciclo y no finalizar. Asimismo, si el problema es un grafo acíclico dirigido con estados repetidos y el algoritmo no comprueba esta repetición, su rendimiento puede verse afectado por la exploración repetida de los subgrafos de búsqueda ubicados bajo dicho estado repetido.

Un aspecto muy importante a tener en cuenta en los algoritmos de búsqueda es la calidad de la solución (o soluciones) que devuelven. Dado un problema basado en un espacio de estados, se define una solución óptima como aquella secuencia de operadores tal que permite pasar de un estado inicial a un estado final y no existe ninguna otra secuencia que tenga un coste inferior. De modo equivalente se define la optimalidad en el grafo equivalente al espacio de estados:

Definición 2.1.2. *Dado un camino $P^* = (s, n_1, \ldots, n_{k-1}, \gamma)$, se dice que P^* es un camino óptimo si no existe ningún otro camino P' del nodo inicial a un nodo meta tal que el coste $g(P')$ de dicho camino sea menor que $g(P^*)$, es decir,*

$$P^* optimo \Leftrightarrow \nexists P' = (s, n'_1, \ldots, n'_{k-1}, \gamma_i), \gamma_i \in \Gamma \quad g(P') < g(P^*) \tag{2.2}$$

Definición 2.1.3. *Dado un camino $P_k = (s, n_1, \ldots, n_{k-1}, n_k)$ y un subcamino $P_{k-1} = (s, n_1, \ldots, n_{k-1})$ contenido en P_k, es decir, $P_{k-1} \subset P_k$, se dice que una función de coste g es monótona si verifica:*

$$\forall P_k \subset P, \quad g(P_k) < g(P) \tag{2.3}$$

```
busqueda* (G, s, Γ, sucesores )
nodo ← s
frontera ← ∅
WHILE nodo ∉ Γ
     Frontera ← Frontera ∪ sucesores(nodo)
     nodo ← seleccionar-nodo-a-expandir (Frontera)
     Frontera ← Frontera / {nodo}
return (nodo)
```

Tabla 2.1: Algoritmo general de búsqueda

Es decir, el coste a lo largo de un camino es estrictamente creciente, por lo cual no puede haber arcos con coste cero o costes negativos.

Nótese que un problema puede tener varias soluciones óptimas, ya que puede haber varios caminos con el mismo coste escalar mínimo. Muchos de los algoritmos genéricos de búsqueda han sido diseñados para devolver la solución óptima, es decir, aquella con el menor coste posible. Por lo general, estos algoritmos finalizan cuando localizan la primera solución óptima, asumiendo que si hubiera más caminos con coste óptimo, la elección de uno u otro entre el conjunto completo de caminos óptimos es indiferente para el centro decisor.

También se han diseñado algoritmos que no devuelven estas soluciones óptimas, sino que devuelven, por ejemplo, la primera solución encontrada, todas las soluciones del problema o una solución con buena calidad (bajo coste, pero no necesariamente el mínimo). En muchos de estos casos estas variantes tienen su origen en la necesidad de disminuir el esfuerzo de la búsqueda en términos de tiempo de procesamiento o memoria empleada.

2.2. Algoritmo general de búsqueda para un único objetivo

En la sección 2.1 hemos visto cómo formalizar un problema de búsqueda en forma de espacio de estados y, por lo tanto, en forma de grafo. Esto permite definir algoritmos genéricos de búsqueda que puedan explorar estos grafos para localizar un camino desde un nodo inicial a un nodo final.

En la tabla 2.1 se muestra un algoritmo general de búsqueda que explora el espacio de estados representado por el grafo G, donde s es el estado inicial, Γ es el conjunto de estados finales y *sucesores* es una función que, dado un nodo n, devuelve el conjunto de estados en G que se pueden obtener directamente a partir de n aplicando los diferentes operadores disponibles, es decir, todos los nodos n_i para los cuales existe un arco (n, n_i) en G.

Este algoritmo selecciona un nodo (inicialmente el estado inicial s). Si este nodo es un estado final, el camino a dicho nodo es una solución al problema, finalizando la

búsqueda; en caso contrario, genera sus sucesores y los añade al conjunto *Frontera*, el cual contiene todos los nodos que se han encontrado pero todavía no se han procesado. A continuación se escoge un nodo de este conjunto y se repite el proceso. El criterio de selección es el que determina el orden de expansión de los distintos caminos posibles, cuyo conjunto tiene forma de árbol y se denomina generalmente árbol de búsqueda o árbol explorado. En caso de que el espacio de estados sea finito y no haya nodos finales, el algoritmo termina y devuelve ∅ (siempre y cuando no haya ciclos en el espacio de estados o éstos sean podados). Este algoritmo básico no contempla, entre otras cosas, el tratamiento de un espacio de estados infinito o la posibilidad de ciclos dentro de dicho espacio de estados.

2.3. Propiedades de los algoritmos de búsqueda

Tomando como base un esquema de búsqueda similar al de la tabla 2.1, se han desarrollado una gran cantidad de algoritmos de propósito general. A la hora de valorar las propiedades de un algoritmo se suelen tener en cuenta, entre otras, las siguientes características:

- Completitud: si existe una solución, el algoritmo la localiza y además finaliza.
- Calidad de la solución: se tiene en cuenta si el algoritmo devuelve una solución óptima, una solución cualquiera o una solución cuya calidad pueda ser medible en términos de comparación con la solución óptima.
- Coste temporal: tiempo de procesamiento requerido por el algoritmo para resolver el problema. En el caso de un único objetivo, es habitual tomar como medida del tiempo el número de nodos procesados por el algoritmo, aunque como veremos más adelante, esta equiparación no es suficiente para evaluar la complejidad de un algoritmo multiobjetivo.
- Coste espacial: se toma habitualmente como medida la cantidad máxima de nodos que el algoritmo necesita mantener en memoria simultáneamente para resolver el problema, la cual tampoco es suficiente en el caso multiobjetivo.

Algunos aspectos a tener en cuenta a la hora de valorar la aplicación de un algoritmo a un problema son los siguientes:

- *Detección de ciclos.* Si el grafo asociado al problema presenta ciclos, un algoritmo que no compruebe esta situación puede caer en uno de estos ciclos y no finalizar nunca su ejecución.
- *Infinitud del espacio de estados.* Si el espacio de estados es infinito, o de una dimensión lo suficientemente grande como para considerarse inexplorable en su totalidad (por ejemplo, desplazamiento de un robot sobre una superficie muy

extensa), podría darse el caso de que el algoritmo tampoco llegue a finalizar si no pone algún tipo de cota de expansión al árbol de búsqueda.

- *Naturaleza de las soluciones.* Aunque muchos de los algoritmos más usados devuelve una solución óptima, en ocasiones puede ser de interés otro tipo de información como determinar si el problema tiene solución, obtener una solución cualquiera, obtener todas las soluciones del problema, obtener todas las soluciones óptimas del problema u obtener una solución subóptima pero con una calidad que sea medible con respecto al coste óptimo.

Por todo ello es muy importante aclarar qué información requerimos del algoritmo y qué casuística presenta el espacio de estados para poder determinar el algoritmo que mejor se ajuste a esa configuración.

2.4. Clasificación de algoritmos de búsqueda para un único objetivo

El trabajo de (Korf, 2010) presenta una descripción y clasificación detallada de diferentes estrategias de búsqueda. Esta clasificación está basada en el mecanismo de expansión de los nodos durante la búsqueda. Distingue dos grandes grupos de algoritmos:

- Algoritmos *best-first*: basados en el principio de optimalidad postulado por Bellman (Bellman, 1954), son aquellos que mantienen el árbol de búsqueda completo en memoria, lo que les permite expandir los nodos según un orden que de algún modo evalúa lo prometedores que son para alcanzar un nodo final. Por ello los nodos se expanden en orden creciente del valor de su estimación de coste.
- Algoritmos *depth-first*: mantienen en memoria exclusivamente la rama del árbol de búsqueda que están expandiendo, con lo cual tienen un consumo de memoria lineal con la profundidad del árbol y el factor de ramificación (número medio de sucesores de un nodo en el grafo del problema) (véase p.ej. (Korf, 1985b)).

En el caso de los algoritmos *best-first*, el criterio básico de exploración es escoger, de entre todos los nodos que se han generado en el árbol de búsqueda y que se mantienen en memoria (pero que todavía no se han expandido), aquel con mayor calidad (o menor coste). Para medir esta calidad, los algoritmos más sencillos tienen en cuenta como criterio de ordenación únicamente la profundidad del nodo en el árbol, otros usan el coste del camino recorrido para llegar al nodo y otros usan información de tipo heurístico (la cual se definirá en la sección 2.5) para estimar su calidad como posibles caminos a expandir. De cualquier modo, los algoritmos precisan mantener una lista ordenada de todos los nodos prometedores por los que se podría continuar explorando para así poder seleccionar el mejor de ellos.

A medida que el proceso de búsqueda va explorando el árbol de búsqueda, el número de nodos que se generan puede crecer de modo exponencial con la profundidad. Si el número medio de sucesores para un nodo (factor de ramificación o *branching factor*) es elevado, este crecimiento será muy acusado. Como resultado la complejidad espacial del algoritmo en el peor caso será también exponencial. Por ejemplo, para un espacio de estados en forma de árbol con profundidad d y factor de ramificación b, el orden de la complejidad espacial en el peor caso es $O(b^d)$. Esto afecta también al coste temporal ya que, por ejemplo, cada vez que se expande un nodo sus sucesores deben incluirse en la lista ordenada de nodos pendientes de expansión, conllevando esta inserción ordenada un tiempo de procesamiento creciente.

En contraposición a los algoritmos *best-first*, los algoritmos de tipo *depth-first* solo necesitan mantener en memoria el camino del árbol que están explorando en cada momento. Cuando finaliza dicha exploración, el algoritmo realiza un retroceso o *backtracking* al nodo anterior y continúa expandiendo una rama diferente. Para un espacio de estados en forma de árbol con profundidad d y factor de ramificación b, la complejidad espacial en el peor caso es $O(b \times d)$. Es por ello que a este tipo de algoritmos también se los denomina *linear-space* o de consumo lineal de memoria. Además el algoritmo no precisa mantener una lista ordenada de nodos pendientes de expansión. Sin embargo, un enfoque *depth-first* puro, suponiendo que devuelva la primera solución localizada, es bastante improbable que encuentre una solución óptima o incluso de calidad, al realizar generalmente la expansión en un orden no relacionado con el coste de los caminos explorados. Asimismo es posible que, aún habiendo una solución, explore antes una rama infinita o cíclica, con lo cual el algoritmo nunca finalizaría.

2.5. La heurística aplicada a la búsqueda

Una de las principales contribuciones de la IA en el campo de los algoritmos de búsqueda en grafos ha sido la introducción de la información heurística para dirigir, junto con la función de coste, la exploración del árbol de búsqueda. La heurística es una técnica que permite *estimar* numéricamente la calidad de los caminos abiertos en el proceso de búsqueda. Está codificada mediante una función denominada heurístico (h), la cual, al aplicarla a un nodo n, nos devuelve una estimación del coste para alcanzar desde dicho nodo un estado final. Dicho de otro modo, el heurístico $h(n)$ evalúa lo cerca que está el estado n de un objetivo final $\gamma \in \Gamma$.

Un proceso de búsqueda dirigido exclusivamente por el heurístico expande primero aquellos caminos que parecen más prometedores. Dado que el heurístico devuelve una estimación, una búsqueda heurística pura no puede asegurar ni la localización de una solución ni el hecho de que si la localiza, ésta sea óptima. En muchos casos, analizando la naturaleza del problema específico a resolver, es posible diseñar heurísticos *ad hoc* para el espacio de estados. Para los casos donde no es posible usar estos heurísticos específicos, a veces se pueden emplear heurísticos genéricos, tales como la distancia euclídea, que pueden servir de estimación para cualquier problema en el que el coste

venga determinado, por ejemplo, por un cálculo de distancias.

Mientras que una búsqueda *best-first* dirigida por el coste del camino recorrido, evoluciona lentamente, explorando exhaustivamente todos los caminos de coste c antes que los de coste c', con $c < c'$, la búsqueda *best-first* dirigida por heurístico es más osada, expandiendo antes los caminos que parecen estar más cerca de un estado final. Esa confianza en la función heurística puede plantear problemas de rendimiento cuando el heurístico no tiene una buena calidad. Por este motivo se han desarrollado algoritmos que guían el proceso de búsqueda en base a una combinación lineal de estos dos tipos de información.

La siguiente combinación fue propuesta para el algoritmo A^* (Hart et al., 1968) del que hablaremos más adelante, y adoptada posteriormente por otros algoritmos de tipo *depth-first*. Un nodo n_k alcanzado a través de un camino $P = (s, n_1, n_2, \ldots, n_k)$, donde s es el estado inicial del problema, es evaluado con la función:

$$f(n_k) = g(n_k) + h(n_k) \tag{2.4}$$

donde:

- $g(n_k)$ es el coste asociado al camino P
- $h(n_k)$ es el valor del heurístico para el nodo n_k y estima el coste de un camino desde n_k hasta un nodo meta

Nótese que para un mismo nodo n puede haber varios valores de $g(n)$ si el nodo es alcanzable desde el estado inicial a través de diferentes caminos, con lo cual si en un momento dado un algoritmo dispone de varios caminos hasta el mismo nodo, deberá disponer de un mecanismo para seleccionar uno de ellos. En los algoritmos basados en el principio de optimalidad de Bellman, la elección recae sobre el de menor coste. Del mismo modo, dado que puede haber diferentes estados meta, un nodo n podría tener un conjunto de valores heurísticos que estimen lo cerca que está el nodo de cada uno de esos estados meta. En consecuencia, en cualquier algoritmo escalar, se seleccionará como valor heurístico el menor de todos ellos. Por convenio la función heurística toma valor cero en los nodos finales.

Una búsqueda guiada por el coste del camino recorrido que expanda siempre el nodo con menor coste garantiza que la solución localizada será óptima. Si introducimos en los cálculos la estimación proporcionada por el heurístico, es posible que la optimalidad del algoritmo se vea comprometida. En la sección 2.6 veremos sin embargo como el uso de cierto tipo de heurísticos puede garantizar la optimalidad.

Sea n un nodo del espacio de estados. Denotemos $h^*(n)$ como el estimador perfecto para el nodo n, es decir, una estimación que es el coste exacto del mejor camino desde el nodo n hasta un nodo final. Se considera que un heurístico h es optimista (o admisible) si proporciona una cota inferior del coste óptimo, es decir:

$$h(n) \leq h^*(n), \ \forall n \in N \tag{2.5}$$

Suponiendo que los costes de todos los arcos son siempre mayores o iguales a cero, algo por otro lado habitual en muchos problemas reales, nótese que el heurístico $h(n) = 0, \forall n$ es admisible. Sin embargo, dado que este heurístico estima la calidad de todos los nodos con el mismo valor, no aporta información alguna que permita discriminar a un nodo en favor de otros. Definimos a continuación la condición de monotonicidad de un heurístico.

Definición 2.5.1. *Supongamos un heurístico h definido sobre un problema representado por un grafo G, con un conjunto de nodos N y un conjunto de arcos A. Se verifica que:*

$$h \text{ monótono} \Leftrightarrow \forall n_i, n_j \in N \quad (n_i, n_j) \in A, h(n_i) \leq h(n_j) + coste(n_i, n_j) \tag{2.6}$$

Las propiedades de admisibilidad y monotonicidad son importantes dado que permiten garantizar la optimalidad de muchos algoritmos de búsqueda heurística. En (Pearl, 1984) puede consultarse información adicional relativa a las propiedades y diseño de heurísticos.

2.6. Algoritmos de búsqueda heurística *best-first* con un único objetivo

El algoritmo *best-first* más conocido para problemas con un único objetivo es A^* (Hart et al., 1968). Este algoritmo realiza una exploración del árbol de búsqueda basada en una lista ordenada de nodos generados, pero pendientes de expansión. Esta lista está ordenada en función del coste del nodo, computado dicho coste como camino recorrido ($g(n)$) más estimación heurística ($h(n)$). Se expande siempre aquel nodo con menor valor de la fórmula 2.4. En caso de que un nodo se pueda alcanzar a través de diferentes caminos, se descartan todos excepto el de menor coste.

Uno de los principales resultados aplicables a este algoritmo es que si el heurístico empleado es monótono (ver fórmula 2.5), cualquier nodo de la lista que se seleccione para expansión ha sido alcanzado a través de un camino óptimo. En tal caso A^* devuelve la solución óptima del problema (siempre y cuando exista alguna). Es posible demostrar que todo heurístico monótono es también admisible.

Otro algoritmo *best-first*, la búsqueda Frontera (*Frontier Search*) (Korf et al., 2005) se diseñó para tratar de reducir los elevados requisitos de memoria de A^*. El término *Frontera* hace referencia a los límites exteriores de la región explorada del espacio de estados. La búsqueda Frontera únicamente almacena este conjunto, que representa los nodos pendientes de expansión, olvidando el conjunto de nodos ya explorados y expandidos. Este conjunto de nodos ya expandidos es útil para, por un lado detectar estados repetidos que puedan aparecer durante la expansión del árbol de búsqueda y, por otro lado, para regenerar el camino al nodo final encontrado.

El primer problema, la aparición de estados repetidos, se soluciona añadiendo en cada nodo explorado un vector de bits con información acerca de los operadores usados

en dicho nodo, inhabilitando aquellos operadores a través de los cuales se ha localizado el nodo actual. Por otro lado, la reconstrucción del camino al nodo final se realiza almacenando en cada nodo frontera información de su predecesor p en la zona media estimada del espacio de estados. Una vez localizada una solución, se realizan sendas búsquedas Frontera desde el estado inicial al nodo p y desde el nodo p al nodo final. Este procedimiento se aplica de modo recursivo hasta reconstruir totalmente el camino solución.

2.7. Algoritmos de búsqueda heurística *depth-first* con un único objetivo

Los algoritmos *best-first* se caracterizan por presentar una complejidad espacial exponencial en el peor de los casos. Esto se da incluso en el caso de la búsqueda Frontera, ya que tiene que almacenar en memoria el último nivel completo del árbol de búsqueda. Esta característica de los algoritmos *best-first* ha motivado un gran interés en otro tipo de algoritmos, la familia *depth-first*, que presenta un consumo lineal de memoria al mantener en la misma únicamente la rama explorada en cada instante. En esta sección presentaremos tres de los algoritmos *depth-first* más conocidos y empleados: *DF-BnB*, IDA^* y $RBFS$.

2.7.1. *DF-BnB* con un objetivo

Uno de los principales representantes de los algoritmos *depth-first* es el algoritmo *DF-BnB* (Lawler y Wood, 1966), denominado también *ramificación y acotación* o *ramificación y poda*. Este algoritmo busca una solución al problema con una búsqueda en profundidad pura, agotando una rama antes de explorar la siguiente. Cuando localiza una solución, su coste le sirve de cota superior para la solución óptima, lo que le permite podar todas las ramas que se generen a continuación y que excedan de dicha cota. Si en una rama se excede el coste de la mejor solución localizada hasta el momento, el algoritmo puede realizar *backtracking*, ya que cualquier solución alcanzable por dicha rama será subóptima (suponiendo, como ya hemos mencionado anteriormente, costes positivos y aditivos para cada arco del grafo).

En la tabla 2.2 se muestra el pseudocódigo del algoritmo *DF-BnB*, el cual devuelve el coste de la solución óptima encontrada. La llamada inicial a este algoritmo sería *DF-BnB* (s, ∞). El principal problema de este algoritmo radica en el tratamiento de espacios de estados infinitos. En una búsqueda *DF-BnB* pura como la realizada por el algoritmo de la tabla 2.2, el algoritmo explora una rama de modo exhaustivo antes de explorar la siguiente. Si *DF-BnB* no ha localizado todavía ninguna solución (óptima o subóptima) y está explorando una rama infinita, dado que puede seguir profundizando indefinidamente en ese camino, no hará *backtracking* y no finalizará, aún cuando pueda haber soluciones en otras ramas del árbol de búsqueda que no hayan sido exploradas todavía.

```
DF-BnB()
     cota ← ∞
     return (BnB-base(s, cota))
BnB-base (n, cota)
     IF (f(n) < cota) THEN
          IF (n ∈ Γ) THEN
               best-sol ← (n, f(n))
               cota ← f(n)
          ELSE
               FOREACH n_i ∈ sucesores(n)
                    cota ← BnB-base (n_i, cota)
     return (cota)
```

Tabla 2.2: Algoritmo Branch & Bound

Es posible fijar una cota superior, ya sea por profundidad o por coste del camino, superada la cual el algoritmo no sigue explorando la rama actual y realiza *backtracking*. Esta técnica evita que el algoritmo se quede atrapado en una rama infinita, pero la cota debe calcularse adecuadamente, pues si las soluciones al problema tienen una profundidad o un coste superior al de la cota, el algoritmo no las encontraría, y por lo tanto no sería completo. Por el contrario, si la cota es muy superior a la profundidad o coste de las soluciones, el esfuerzo de búsqueda podría ser muy grande. En (Vempaty et al., 1991) se planteaba la complementariedad de *DF-BnB* con otros algoritmos como IDA^*, que describiremos más adelante, proponiendo un algoritmo híbrido, DFS^*, el cual localiza una primera solución mediante, por ejemplo, profundización iterativa. Una vez localizada esta solución, el algoritmo conmuta al algoritmo *DF-BnB*, usando esta primera solución como cota y obteniendo el resto. Además, en DFS^* se plantea un uso más *laxo* de la profundización iterativa, incrementando las cotas de modo más agresivo, minimizando la reexpansión de nodos, a costa de poder obtener una primera solución subóptima.

En cuanto al tratamiento de nodos repetidos, cuando se producen ciclos en la rama actual del árbol de búsqueda *DF-BnB* los puede detectar al tener la rama almacenada en memoria. Sin embargo, cuando los nodos se repiten en diferentes ramas (es decir, se puede llegar a un mismo estado a través de diferentes caminos), no es posible para *DF-BnB* detectar esta situación[2], lo cual generará un problema de rendimiento al reexpandir el subárbol de búsqueda del nodo repetido. Esto se debe a que los algoritmos *depth-first*, para mantener una complejidad espacial baja, no guardan una lista de estados visitados.

Otro problema de la búsqueda *DF-BnB* es que durante su exploración puede expandir nodos cuyo coste es superior al de la solución óptima. En la figura 2.2 se presenta un ejemplo de este comportamiento. Al expandir el nodo raíz s se generan los sucesores

[2]Una alternativa para paliar parcialmente esta situación en algoritmos *depth-first* es el uso de tablas de trasposición (Reinefeld y Marsland, 1994).

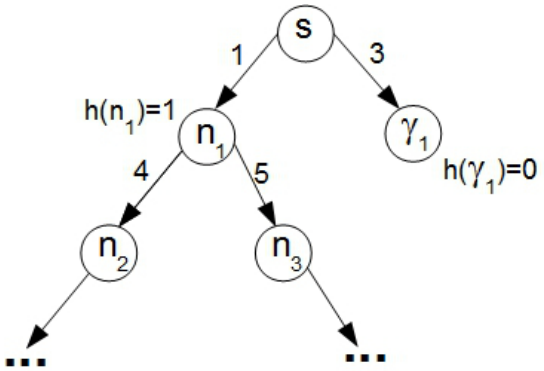


Figura 2.2: Expansión de nodos subóptimos en una búsqueda *depth-first*

n_1 y γ_1. Suponiendo que el orden de expansión de las ramas viene dado por la función f definida anteriormente, como $f(n_1) = 2$ y $f(\gamma_1) = 3$, se expande primero el subárbol bajo n_1, donde todos los nodos tendrán peor coste que γ_1. Con un orden de expansión de izquierda a derecha, en este ejemplo *DF-BnB* también exploraría nodos de coste subóptimo.

2.7.2. IDA^*: Profundización iterativa para un objetivo

El procesamiento de espacios de estados infinitos, no soportados por *DF-BnB*, puede solventarse con el algoritmo IDA^* (*Iterative Deepening A**) (Korf, 1985b). Este algoritmo aplica a la búsqueda en grafos la denominada técnica de profundización progresiva o iterativa, postulada por primera vez por (de Groot, 1965). Esta técnica fue usada por primera vez con gran éxito en el programa de ajedrez *Chess 4.5* desarrollado por Larry Atkin y David Slate (Slate y Atkin, 1979). El algoritmo IDA^* establece una cota[3] de expansión inicial que viene dada por $f(s)$, donde s es el estado inicial del problema. A continuación realiza una búsqueda en profundidad del árbol de búsqueda, discontinuando la exploración de una rama si su estimación de coste $f(n) = g(n)+h(n)$ (donde n es el último nodo generado en la rama explorada, y por tanto el más profundo de dicha rama) excede la cota establecida. Si durante la búsqueda el algoritmo localiza una solución dentro de la cota, entonces la devuelve, finalizando el procesamiento. En caso contrario, se establece una nueva cota de expansión que viene dada por el menor coste $f(n)$ de todos los nodos n en los cuales se ha discontinuado la búsqueda. Este proceso se repite, realizando diferentes iteraciones hasta localizar un nodo final del problema. Nótese que esta cota superior de expansión es asimismo una cota inferior del coste de la solución óptima.

Se puede ver claramente que el algoritmo es completo, ya que la cota crece de modo estricto y en alguna iteración la cota correspondiente llegará a igualar o superar el coste de la solución óptima, localizándola y finalizando el algoritmo. Asimismo, si el

[3] A lo largo de esta Tesis emplearemos indistintamente los términos *cota*, *umbral* y *threshold*.

```
IDA* (s)
     cota ← f(s); siguiente-cota = ∞
     solucion ← ∅; coste-optimo = ∞
     WHILE solucion == ∅
          DFS(s)
          cota ← siguiente-cota
          siguiente-cota ← ∞
DFS(n)
     FOREACH n_i ∈ sucesores(n)
          IF (n_i ∈ Γ) ∧ (f(n_i) ≤ cota) THEN
               solucion ← n_i
               coste-optimo ← f(n_i)
               EXIT
          IF (f(n_i) ≤ cota) THEN
               DFS(n_i)
          ELSE IF (f(n_i) < siguiente-cota) THEN
               siguiente-cota ← f(n_i)
```

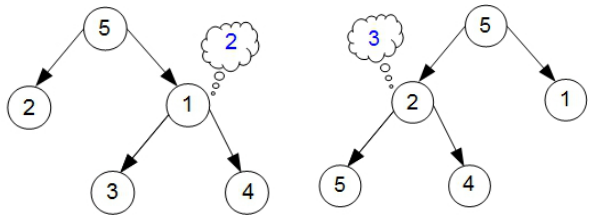
Tabla 2.3: Algoritmo IDA*

heurístico es optimista, no se llegará a expandir ningún nodo cuyo coste sea superior al de la solución óptima, garantizando la optimalidad de la solución obtenida. En (Korf, 1993) se demuestra que si la función de coste es monótona, un algoritmo de profundización iterativa expandirá los nodos por primera vez en orden *best-first*.

Por último, IDA^* no presenta problemas de terminación con ramas infinitas ni con ciclos, siempre y cuando los costes de los arcos sean positivos, al estar la búsqueda en profundidad acotada en todo momento. En la tabla 2.3 se muestra el pseudocódigo del algoritmo IDA^*, donde s es el estado inicial y Γ es el conjunto de estados finales del problema. El principal inconveniente de los algoritmos de profundización iterativa es que en cada una de las iteraciones reexpanden todos los nodos de la iteración anterior[4]. Estas reexpansiones pueden conllevar un coste temporal elevado. El peor caso para la profundización iterativa sería un espacio de estados en el cual todos los nodos tuvieran costes diferentes. Esto provocaría que con cada actualización de la cota, el algoritmo sólo expandiría un nodo nuevo.

Los algoritmos de profundización iterativa, al igual que el algoritmo *DF-BnB* visto en la sección 2.7.1, tampoco detectan en principio la existencia de diferentes caminos que llevan a un mismo nodo, con lo cual pueden reexpandir varias veces un mismo subárbol de búsqueda.

[4]Salvo en la última, donde esto ocurrirá tan solo en el peor caso.



(a) Paso 1 (b) Paso 2

Figura 2.3: Ejemplo de expansión de nodos con *RBFS*

2.7.3. El algoritmo *RBFS*

En (Korf, 1993) y (Korf, 1992) se propone *RBFS* (*Recursive Best-First Search*) un algoritmo que podemos clasificar, a pesar de su nombre, como *depth-first* al tener consumo lineal de memoria. Este algoritmo se presenta como un intento de realizar una expansión del árbol basada en la función de coste, es decir, el algoritmo expande siempre el siguiente nodo nuevo con menor coste ($f(n)$) (de ahí la referencia a *best-first* en su nombre). Con un heurístico monótono, *RBFS* expande los nodos del grafo de búsqueda en el mismo orden que un algoritmo de tipo *best-first*.

Para mantener un consumo lineal de memoria *RBFS*, al igual que el resto de algoritmos *depth-first*, mantiene en memoria únicamente la rama que está explorando en cada momento. Es decir, mientras expande una rama, elimina de memoria el resto de ramas visitadas, pero recordando el mejor coste c alcanzado por cualquiera de esas ramas *olvidadas*. Si en la rama explorada se supera dicho coste c, se reconsidera la rama *olvidada*, expandiéndola de nuevo, eliminando de memoria la actual y recalculando el mejor coste *olvidado* de la misma. En todo momento en memoria sólo se mantiene el camino al mejor nodo más los hermanos de todos los nodos en dicho camino.

En la figura 2.3 podemos ver un pequeño árbol de búsqueda binario expandido por *RBFS* (ejemplo tomado de (Korf, 1993)). Cada nodo está etiquetado con su función de coste. Inicialmente el algoritmo expande el nodo raíz, generando dos sucesores con costes 1 y 2. Como podemos apreciar en la figura 2.3a, *RBFS* expande en primer lugar el nodo con mejor coste (1), eliminando de memoria el resto de ramas. Además, en el nodo a expandir *RBFS* apunta el coste de la mejor rama olvidada (en nuesto caso, coste 2). Este valor servirá de cota superior para la expansión, de tal modo que si en la expansión de la rama todos los caminos son peores que el mejor de los olvidados, *RBFS* realiza *backtracking* para reexplorar ese camino olvidado.

En nuestro ejemplo, los sucesores del nodo con coste 1 tienen costes 3 y 4 respectivamente, cuando según lo anterior, hay un camino con coste 2 que se discontinuó, el cual debe ser reconsiderado. Tal y como vemos en la figura 2.3b, *RBFS* realiza *backtracking* y reexpande el nodo con coste 2, eliminando la rama derecha de memoria y anotando que por dicha rama el mejor camino tenía coste 3. Al generar los sucesores, con costes 4 y 5, el algoritmo reconsiderará de nuevo el camino con coste 3.

```
RBFS (n, v, cota_s)
     IF (f(n) > cota_s) THEN
          return f(n)
     IF (n ∈ Γ) THEN
          EXIT algorithm
     IF (sucesores(n) = ∅) THEN
          return ∞
     FOREACH n_i ∈ sucesores(n)
          IF (f(n) < v ∧ f(n_i) < v) THEN
               F[i] ← v
          ELSE
               F[i] ← f(n_i)
     Ordenar de modo creciente n_i y F[i] por valor de F[i]
     IF |sucesores(n)| = 1 THEN
          F[2] ← ∞
     WHILE (F[1] ≤ cota_s) DO
          F[1] ← RBFS(n_1, F[1], mín(cota_s, F[2])
          Insertar n_1 y F[1] en orden creciente
     return F[1]
```

Tabla 2.4: Algoritmo *RBFS* (tomado de (Korf, 1992))

En la tabla 2.4 se muestra el pseucódigo del algoritmo *RBFS*. La llamada inicial a este algoritmo se realiza con los parámetros $(s, f(s), \infty)$, donde s es el estado inicial del problema. El algoritmo recibe un nodo n a expandir y una cota superior de expansión $cota_s$ (el parámetro v se comentará más adelante). Si el coste $f(n)$ del nodo supera dicha cota, se devuelve dicho coste como valor del mejor camino disponible por esa rama. Si el nodo n es un nodo meta, entonces el algoritmo finaliza. Es decir, *RBFS* devuelve la primera solución localizada. Con un heurístico admisible (ver fórmula 2.5) y una función de coste aditiva y con valores positivos, *RBFS* garantiza que esta solución será óptima.

Si por el contrario, n no es un nodo meta, el algoritmo ordena sus sucesores por orden creciente de coste y expande el mejor de ellos, mientras su coste no exceda el establecido por la cota. A medida que se expanden los nodos, de sus correspondientes subárboles de búsqueda se informa del mejor camino encontrado en dichos subárboles.

Como podemos apreciar en el ejemplo, *RBFS* presenta un comportamiento parecido a los algoritmos de profundización iterativa, con reexpansiones extra de nodos, las cuales empeoran el rendimiento en términos de tiempo. Una expansión guiada por el procedimiento anterior es similar a una profundización progresiva, ya que un subárbol se expande mientras haya caminos cuyo coste no exceda la cota marcada por el coste del mejor camino discontinuado. Este funcionamiento puede ser muy ineficiente en casos como el mostrado en la figura 2.4, tomada de (Korf, 1992).

En este ejemplo suponemos que el coste de cada arco es unitario y no se emplea heurístico (es decir, $h(n) = 0 \quad \forall n$). Tras una serie de profundizaciones, el algoritmo

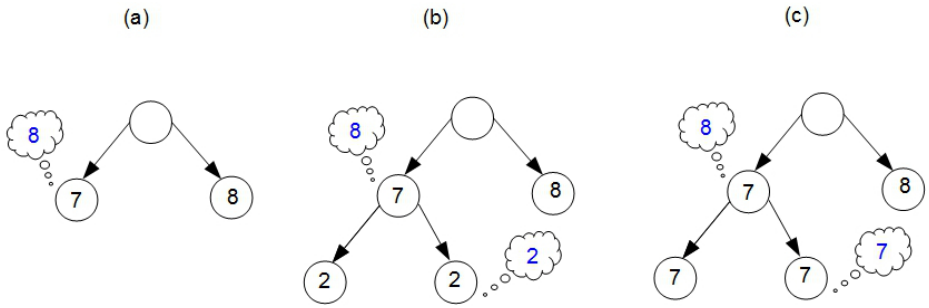


Figura 2.4: Ineficiencia de *RBFS*

alcanza la situación de la subfigura (a), dónde hay un camino de coste 7 a la izquierda del árbol y otro de coste 8 a la derecha. Al expandir el nodo de coste 7, sus sucesores quedan etiquetados tal y como se indica en la subfigura (b). Ambos nodos tienen el mismo coste, 2, con lo cual se expande uno cualquiera de ellos, usando el coste del otro como cota. Esto provocará que hasta profundidad 7, en cada nivel del árbol se vaya alternando de la rama derecha a la izquierda y viceversa, generando un número de reexpansiones muy elevado. Sin embargo, en el nodo padre, la etiqueta informa de que en el subárbol hay por lo menos un camino con coste 7 que merece ser explorado. Es más eficiente explorar las ramas en modo *depth-first* hasta localizar el nodo con coste 7 y a partir de ahí proceder con la expansión *normal* de *RBFS* que hemos comentado.

Para conseguir esto, en *RBFS* se propaga la etiqueta de un nodo por su subárbol de búsqueda si hay constancia de que este nodo ya fue expandido en una etapa previa. Esta situación se da cuando su coste es inferior al valor de su etiqueta. Recordamos que esta etiqueta indica el coste del mejor camino localizado en su expansión y toma valor ∞ para el nodo raíz. Este comportamiento se muestra en la subfigura (c), donde apreciamos que los nodos de nivel 2 heredan la etiqueta de su nodo padre. En el algoritmo, esta etiqueta transmitida por el nodo padre se refleja en el parámetro v de la función *RBFS* (tabla 2.4).

Recientemente, en (Hatem et al., 2015) se propone $RBFS_{CR}$, una variante de *RBFS* que reduce el número de reexpansiones de nodos propagando por el árbol costes actualizados basados en la frecuencia de los costes bajo un nodo cualquiera. Por cada nodo se almacena un histograma de los valores de f en los nodos que han sido podados durante la exploración de una rama. Este histograma se emplea para calcular el valor actualizado de coste del nodo de un modo menos conservador, lo que provoca expansiones más profundas del subárbol bajo el nodo actual, a costa de la optimalidad del algoritmo. Este algoritmo está basado en una variante similar realizada para IDA^* (Sarkar et al., 1991).

2.8. Rendimiento de los algoritmos de búsqueda heurística con un único objetivo

Según lo visto en este Capítulo, claramente los algoritmos de tipo *depth-first* tienen mejor rendimiento que los *best-first* en lo que respecta a complejidad espacial en el peor caso, al mantener únicamente en memoria la rama explorada. Sin embargo, en lo que respecta al tiempo de procesamiento, la ventaja de unos algoritmos frente a otros no está tan clara.

Analizando la cantidad de nodos que procesa cada algoritmo, podemos afirmar que una búsqueda *depth-first* expande en el peor caso como mínimo el mismo número de nodos que una búsqueda *best-first*. Un algoritmo de tipo *best-first* como A^* (Hart et al., 1968), equipado con un heurístico monótono, expande cada nodo únicamente una vez, mientras que en los algoritmos de profundización iterativa se pueden producir numerosas reexpansiones de subárboles de búsqueda completos. En el caso del algoritmo IDA^* aplicado a árboles, si el factor de ramificación efectivo del problema es mayor que uno, entonces IDA^* genera asintóticamente el mismo número de nodos que A^*. Sin embargo hay situaciones donde el comportamiento de la profundización iterativa se degrada mucho con respecto al de una búsqueda *best-first*. Por ejemplo, si todos los nodos del grafo tienen un coste $f(n)$ diferente, entonces un algoritmo ID genera un único nodo nuevo en cada profundización. Por el contrario un enfoque *best-first* solo expandiría N nodos. Hay algoritmos *depth-first* que no reexpanden nodos, como es el caso de *DF-BnB*. Sin embargo, *DF-BnB* normalmente expande nodos cuyo coste es subóptimo, pues el orden de expansión del árbol no viene fijado por la función de coste.

En el análisis de algoritmos para un objetivo (por ejemplo, (Zhang, 1999)) es frecuente realizar comparativas basadas en la cantidad de nodos expandidos, equiparando el tiempo de procesamiento al número de nodos. Esto conlleva suponer que el tiempo de procesamiento por nodo es constante e igual para todos los algoritmos. Sin embargo, el tiempo necesitado para expandir un nodo en un algoritmo *best-first* es bastante mayor que en un algoritmo *depth-first* (Vempaty et al., 1991). Esto se debe a que un algoritmo *best-first* mantiene una lista de nodos a expandir ordenada en función de su coste. Cada vez que se expande un nodo, éste debe eliminarse de la lista y se deben añadir sus sucesores, siendo el tiempo de acceso a esta lista logarítmico en el número total de nodos en la misma. Esto incrementa significativamente el tiempo de expansión de un nodo. Cuanto mayor es el factor de ramificación de un problema, mayor será este tiempo de procesamiento.

En (Vempaty et al., 1991) se realiza un análisis del tipo de problemas más adecuados para cada tipo de algoritmos, en función de la cantidad de nodos finales y del factor de ramificación. Los problemas resueltos en los experimentos son el *Puzle-15*, el problema del viajante (Lawler et al., 1985) y la búsqueda en laberintos y los algoritmos empleados son A^*, IDA^* y *DF-BnB*. Los resultados obtenidos permiten llegar a la conclusión de que con una densidad de nodos finales y un factor de ramificación altos, *DF-BnB* es el algoritmo preferible. Con una densidad de nodos finales baja y un factor de ramificación

alto, la opción a escoger es IDA^*. El algoritmo A^* es útil solo cuando tanto la densidad de nodos finales como el factor de ramificación son bajos.

En (Zhang y Korf, 1993), (Zhang y Korf, 1995) y (Zhang, 1999) se realizan unas comparativas similares que incluyen también el algoritmo *RBFS*. Los problemas empleados son árboles aleatorios finitos, *Puzle-n* y problema del viajante. En el caso de los árboles aleatorios se analizan tanto el número de nodos expandidos como el tiempo de ejecución. En lo que respecta a cantidad de nodos, el análisis concluye que para árboles acotados y de gran tamaño, un enfoque *DF-BnB* se muestra como el más adecuando, mientras que *RBFS* es más eficiente con problemas sencillos o no acotados en profundidad. Si bien A^* es óptimo en cuanto a número de nodos expandidos con un heurístico monótono, sus tiempos de procesamiento son mucho mayores debido al coste temporal asociado al procesamiento de la lista ordenada de nodos pendientes de expansión. En el caso de espacios de estados con forma de grafos cíclicos, *DF-BnB* no es viable, ya que puede caer en un ciclo y no finalizar. En cuanto a IDA^* y *RBFS*, el primero presenta mejor rendimiento.

Estos trabajos concluyen de modo general que los algoritmos más adecuados son *DF-BnB* para espacios de estados con forma de árbol acotado en profundidad y *RBFS* para espacios de estados con forma de árbol no acotado. Para espacios de estados en los cuales pueda haber ciclos, la mejor opción es un algoritmo *best-first*, al no contemplar los algoritmos *depth-first* la posibilidad de detección de estados repetidos.

El uso de buenos heurísticos palía en cierto modo la complejidad temporal de los algoritmos, tanto *best-first* como *depth-first*, al dirigir la búsqueda hacia aquellos nodos que parecen más prometedores. En lo que respecta a los algoritmos *depth-first*, se han diseñado diferentes técnicas con el objetivo de reducir sus tiempos de procesamiento. Por ejemplo, en (Björnsson et al., 2005) se presenta la búsqueda *Fringe*, que realiza profundización iterativa pero emplea un conjunto *Frontera* que mantiene una lista no ordenada de los nodos que se discontinúan durante la iteración actual al haber superado la cota correspondiente. Estos nodos son los que se utilizan como punto de partida para la búsqueda iterativa de la siguiente iteración, lo cual ahorra la reexpansión del árbol que va desde el nodo raíz a este conjunto *Frontera*. Además, el hecho de mantener este conjunto no ordenado evita los altos costes de inserción en el mismo. Estos costes, en el caso de A^*, que sí ordena la lista de nodos, pueden ser logarítmicos en el tamaño de la lista.

En (Reinefeld y Marsland, 1994) se proponen técnicas tales como ordenación de nodos o uso de tablas de transposición para dirigir de modo más eficiente la búsqueda y el tratamiento de nodos repetidos, evitando la reexpansión innecesaria de subárboles de búsqueda.

2.9. Resumen

En este Capítulo hemos definido y formalizado los problemas de búsqueda del camino más corto. Asimismo se ha presentado una clasificación de los algoritmos de

búsqueda para el caso de un objetivo, la cual está basada en la cantidad de memoria requerida para su procesamiento. Para cada clase de algoritmos (*best-first* y *depth-first*) se han presentado los algoritmos más relevantes, junto con un pequeño resumen de su rendimiento y posible ámbito de aplicación. Trabajos previos, en especial (Zhang, 1999), muestran que, de modo general, los algoritmos de tipo *best-first* son más adecuados para problemas con espacios de estados con ciclos. Asimismo los algoritmos *depth-first* presentan mejores resultados con espacios de estados en forma de árbol, siendo *DF-BnB* y *RBFS* las mejores opciones para árboles acotados y no acotados respectivamente.

En el Capítulo 3 generalizaremos este análisis al caso multicriterio, formalizando los problemas multiobjetivo, clasificando los algoritmos de búsqueda también en base a su consumo de memoria y presentando algunas de las generalizaciones del caso con un objetivo al caso multiobjetivo.

Capítulo 3

Búsqueda Multiobjetivo

En el Capítulo 2 se han presentado diferentes algoritmos orientados a la resolución de problemas de camino mínimo (*Shortest Path*) en los cuales el coste de un camino se representa mediante un valor escalar. En los últimos años ha surgido, junto a este paradigma de toma de decisiones basado en la optimización de un coste escalar, uno basado en la Teoría de la Decisión Multicriterio (Yu et al., 1985) (Chankong y Haimes, 1983), la cual permite el análisis y resolución de situaciones más realistas en las cuales el coste de un camino no viene dado exclusivamente por un valor, sino que involucra a más de un objetivo a minimizar (o maximizar), teniendo cada uno de estos objetivos su propia función de coste (o beneficio).

La creciente importancia de este tipo de problemas en el mundo real ha tenido su reflejo también en el problema de la búsqueda en grafos, lo que ha propiciado la aparición de diversos algoritmos de búsqueda que tratan de lidiar con la complejidad inherente al tratamiento de múltiples costes.

En este capítulo presentaremos la formalización de los problemas de búsqueda multiobjetivo en grafos, así como los algoritmos propuestos para su resolución. En primer lugar, en la sección 3.1 presentaremos los conceptos básicos de la Teoría de Decisión Multicriterio y, más concretamente, la subclase de problemas que nos atañen en este trabajo, la búsqueda multiobjetivo. En la sección 3.2 se enumeran algunos problemas de búsqueda multiobjetivo reales. La sección 3.3 define las nociones básicas de búsqueda heurística para varios objetivos. Presentaremos nuevos conceptos como la frontera de Pareto o las relaciones de dominancia, ambos claves para la redefinición de aspectos como la optimalidad de una solución, cuyo significado varía sustancialmente respecto a lo visto en el Capítulo 2 para los algoritmos de un único objetivo. De modo similar a cómo se trataron los algoritmos de búsqueda para un único objetivo en el Capítulo 2, en la sección 3.4 se muestra una clasificación de los algoritmos basada en su uso de la memoria, describiendo los principales trabajos previos realizados en este campo. Las secciones 3.5 y 3.6 presentan los algoritmos multiobjetivo más comunes de tipo *best-first* y *depth-first* respectivamente. La presente Tesis se centra en la generalización de las técnicas *depth-first* al caso multiobjetivo, por lo que en esta sección también se

explicarán con detalle dos algoritmos de este tipo: $IDMOA^*$ (Harikumar y Kumar, 1996) y $MOMA^*0$ (Dasgupta et al., 1999), que corresponden a posibles generalizaciones de algoritmos de optimización escalar (IDA^* y $RBFS$), comentando asimismo ciertos aspectos de la búsqueda *DF-BnB* en el ámbito multiobjetivo. La sección 3.7 incluye una clasificación de los algoritmos multiobjetivo exactos de tipo *depth-first* (tanto los presentados en este capítulo como los aportados en este trabajo de investigación), en base a la estrategia de expansión y a la naturaleza de la cota superior empleada en dicha expansión. Esta clasificación sirve como base para enmarcar los trabajos descritos en la segunda parte de esta tesis. Por último, en la sección 3.8 se indicarán algunas consideraciones generales sobre el rendimiento de estos algoritmos.

3.1. Búsqueda heurística multiobjetivo

La Teoría de Decisión Multicriterio da cabida a problemas en los cuales un centro decisor toma sus decisiones en función de los valores de diferentes objetivos, a diferencia de lo visto en el Capítulo 2. Esta teoría ha sido ampliamente estudiada en las últimas décadas (Bell et al., 1977) (Chankong y Haimes, 1983) (Cohon, 2004) (Goicoechea et al., 1982) (Haimes y Chankong, 1985) (Hwang y Masud, 1979) (Oppenheimer, 1977) (Spronk y Zionts, 1984) (Steuer, 1986) (Zeleny, 1982). Un ejemplo sencillo para ilustrar este paradigma multicriterio es el cálculo de una ruta entre dos puntos en una red de carreteras. Según el paradigma escalar podría abordarse el cálculo de la ruta óptima usando el tiempo o la distancia como función de coste. Sin embargo es razonable pensar en otros criterios a minimizar, como el coste económico del viaje en términos de combustible, peajes de autopista, desgaste del vehículo o riesgo medioambiental (en el transporte de materiales peligrosos). La Teoría de Decisión Multicriterio permite analizar la selección de la mejor ruta considerando todos estos criterios simultáneamente.

Romero (Romero, 1993) destaca los siguientes conceptos en el contexto de la Teoría de Decisión Multicriterio:

- *Atributo.* Son valores relacionados con una propiedad mensurable y que contempla el centro decisor como parámetros a valorar en una solución.
- *Objetivo.* Representa la dirección de mejora de los atributos: minimización o maximización.
- *Nivel de aspiración.* Es un nivel aceptable de consecución para un determinado atributo.
- *Meta.* Es la combinación de un atributo y un nivel de aspiración.

En nuestro ejemplo de la red de carreteras, los atributos pueden ser la distancia y el coste económico. Los correspondientes objetivos son la minimización de ambos atributos. Una meta puede ser, por ejemplo, *realizar un gasto inferior a* 120€.

El paradigma multicriterio involucra habitualmente problemas con soluciones en las cuales no es posible mejorar un objetivo sin empeorar al menos otro. Estos puntos de equilibrio son soluciones óptimas, también denominadas óptimos de Pareto. Si fuera posible mejorar un objetivo sin perjudicar a otro, entonces la solución no sería óptima, al haber margen de mejora. La posible existencia de múltiples soluciones con distinto coste para los diferentes objetivos, pero a la vez todas ellas óptimas, es uno de los factores clave a tener en cuenta a la hora de resolver problemas multicriterio.

Existen distintos tipos de análisis multicriterio. Por un lado nos encontramos el análisis multiobjetivo, cuya finalidad es la obtención de todas las soluciones óptimas de un problema. Estos análisis permiten revelar todos los compromisos existentes entre las distintas soluciones. Este tipo de análisis no tiene en cuenta *a priori* las preferencias del centro decisor, las cuales se usarán para tomar una decisión *a posteriori* entre las soluciones localizadas.

Por otro lado, encontramos aquellos enfoques que pretenden alcanzar una solución mediante una serie de preferencias del centro decisor proporcionados *a priori*, o bien a través de un proceso interactivo. Podemos destacar por un lado los modelos de la Teoría de la Utilidad Multiatributo, que agregan todos los atributos a optimizar en una única función (Bell, 1979) (Chankong y Haimes, 1983) (Charnetski et al., 1977) (Dyer y Sarin, 1979) (Goicoechea et al., 1982) (Keeney y Raiffa, 1993) (Schoemaker y Waid, 1982) (Weber, 1985) (Zeleny, 1982). Esto permite ponderar los atributos en función de la importancia que les otorgue el centro decisor. Nuevamente, el caso de la agregación lineal de atributos es uno de los más sencillos y estudiados, aunque también se emplean métodos que optimizan los criterios de uno en uno, estableciendo restricciones para los restantes (Eschenauer et al., 1990). Estos problemas son susceptibles de ser resueltos por algoritmos de búsqueda para un único objetivo, como los vistos en el Capítulo 2. No obstante, una limitación formal de esta técnica es que no permite encontrar cualquier solución que sea óptimo de Pareto. Las soluciones que son alcanzables optimizando una combinación lineal de atributos reciben el nombre de soluciones soportadas.

Por último podemos destacar también los enfoques basados en la satisfacción de metas, que se apartan, al menos formalmente, del concepto de optimización para encontrar una solución satisfactoria. La programación por metas es idónea para problemas en los cuales el centro decisor tiene que tomar una decisión en un contexto de metas múltiples.

Esta tesis se centra exclusivamente en el análisis multiobjetivo comentado en primer lugar, estudiando algoritmos que devuelven el conjunto de todas las soluciones óptimas de Pareto para un problema. Estas soluciones óptimas involucran diferentes costes asociados a cada uno de los objetivos contemplados en el problema. Una vez obtenidas todas las alternativas óptimas, existen multitud de técnicas para seleccionar la alternativa adecuada en base a preferencias individuales (Bogetoft, 1986) (Kok, 1986) (Korhonen, 1986) (Korhonen et al., 1986) (Mond y Rosinger, 1985) (Vansnick, 1986).

Existen problemas multiobjetivo en los cuales los objetivos considerados presentan la misma tendencia. Por ejemplo, si en el diseño de circuitos contemplamos el coste

económico del circuito y su consumo, interesa minimizar ambos objetivos. Además, si reducimos el coste incluyendo menos unidades funcionales en el circuito, esto repercute proporcionalmente en el consumo del mismo. En estos casos el problema multiobjetivo real se puede considerar, en la práctica, un problema con un objetivo único. Minimizando el coste económico, por ejemplo, se minimiza el consumo. En muchos casos estos problemas se resuelven empleando algoritmos de búsqueda para un objetivo (ver Capítulo 2).

Sin embargo, en la mayoría de situaciones los objetivos son contrapuestos. En estos casos no existe normalmente una solución o coste óptimo único, y es preciso alcanzar algún tipo de compromiso entre los distintos objetivos considerados.

Uno de los ejemplos clásicos de análisis multiobjetivo es el ya mencionado de búsqueda de una ruta en una red de carreteras, donde dados dos caminos diferentes entre un origen y un destino, probablemente el camino más rápido incluirá circular por autopistas y a más velocidad que por otras rutas, con lo cual el coste económico se eleva. De modo análogo, rebajar el coste económico del viaje puede conllevar el no circular por autopistas para evitar los peajes. Esto se traducirá probablemente en un mayor tiempo de recorrido. En la figura 3.1 se muestran dos rutas diferentes entre dos ciudades obtenidas con criterios de optimización distintos. En la figura 3.1a se observa el resultado de calcular la ruta primando la minimización de la distancia. Se aprecia que la distancia recorrida es de 1.125km, con un coste económico estimado para el viaje, entre combustible y peajes, de 124′55€. En la figura 3.1b, por el contrario, aparece el resultado de calcular la ruta primando la minimización del coste económico. Se aprecia que el coste económico es inferior (100′85€), fundamentalmente por evitar peajes, mientras que la distancia recorrida es superior (1.130km).

Otro ejemplo lo encontramos en el problema del diseño de circuitos electrónicos. Podemos encontrar aquí nuevamente dos objetivos con tendencia distinta: el coste económico del circuito y el retardo máximo del mismo. El introducir más unidades funcionales y/o registros en un circuito permite una paralelización de operaciones que de otra forma deben realizarse en bucle. Esta paralelización disminuye el retardo, pero también aumenta el coste y el área del circuito. Es decir, en el caso de objetivos compensados o de distinta tendencia, al tratar de mejorar uno de los objetivos, el resto se verán probablemente empeorados, con lo cual es necesario buscar una solución de compromiso para los valores involucrados en el coste final.

Como consecuencia de lo anterior, en los problemas multiobjetivo ya no disponemos en general de un único coste escalar óptimo, sino que puede haber un conjunto de soluciones con costes diferentes pero todas igual de válidas u óptimas, siendo la elección entre una u otra de carácter subjetivo. Supongamos que en el ejemplo de la búsqueda de rutas por carretera cada camino P tiene asociado un coste definido por (r_p, c_p), donde r_p es la cantidad de kilómetros del camino y c_p es el coste monetario derivado de la circulación por dicho camino. Un camino P_1 con coste (1.125km,124'55€) no es mejor ni peor que otro camino P_2 con coste (1.130km,100'85€). Por lo tanto los algoritmos exactos y óptimos que se emplean para la búsqueda del camino más corto,

(a) Itinerario 1 (b) Itinerario 2

Figura 3.1: Cálculo de rutas óptimas en un mapa de carreteras atendiendo a diferentes criterios: distancia (a) y coste económico (b)

los cuales finalizan cuando localizan la mejor solución, no son en general aplicables directamente al caso multiobjetivo, en el cual puede haber varias soluciones óptimas.

3.2. Aplicaciones de la búsqueda multiobjetivo

Una gran cantidad de problemas que pueden modelarse como búsqueda en grafos precisan del tratamiento simultáneo de diferentes objetivos, los cuales con frecuencia presentan algún tipo de compromiso entre ellos. Algunos ejemplos citados en la literatura son:

- *Particionado de circuitos* (Harikumar y Kumar, 1997). Consiste en la división de un circuito en partes que puedan fabricarse como componentes separadas. El tamaño de las componentes viene limitado por la cantidad de puertas disponibles en las FPGA (*Field Programmable Gate Arrays*) comerciales, de tal modo que cada subdivisión del circuito debe poder incluirse en una FPGA. A mayor cantidad de componentes, hay una menor necesidad de reutilización de FPGA, minimizando los retardos. Sin embargo, esto implica una mayor cantidad de interconexiones, las cuales están físicamente limitadas por la cantidad de pines de I/O de las FPGA.

- *Planificación de operadores* (Dasgupta et al., 1999) (McFarland et al., 1988). Consiste en la realización de diseños RTL (*Register Transfer Level*) a partir de

especificaciones del comportamiento de un circuito. Un diseño RTL está formado por unidades funcionales y registros de almacenamiento, así como las interconexiones entre los mismos. Por un lado se busca un diseño lo más pequeño posible (con bajo número de unidades funcionales y registros), lo cual influye directamente en el coste del circuito. Por otro lado interesa que el diseño tenga un retardo lo más bajo posible. Este incremento de rapidez se consigue con la paralelización de operaciones incluyendo más unidades funcionales y registros.

- *Subastas combinatorias* (Buer y Pankratz, 2010). En las subastas combinatorias existen un conjunto T de bienes a subastar. Una puja es una tupla $T = (s, \tau, p)$, donde s es el pujador, $\tau \in T$ es el subconjunto de bienes por que el se puja y p es el precio de la puja. Resolver este problema consiste en buscar una solución que maximice el dinero obtenido de las diferentes pujas, pero de tal modo que el número de bienes adjudicado sea también el mayor posible. Existen variantes, como por ejemplo la subasta de contratos de servicio, donde una administración pública paga a proveedores por ofrecer dichos servicios, pero cuidando la calidad con la que se ofrecen. En este caso una puja es una tupla $T = (s, \tau, p, qos)$, donde *qos* es la calidad de servicio que ofrece el pujador (p.ej. el porcentaje de entregas sin retraso para contratos de transporte). En este caso los objetivos son minimizar el coste para la administración por ofrecer esos servicios y maximizar la calidad de servicio ofrecida a los administrados.

- *Gestión de energía en sistemas de almacenamiento híbridos* (Boxnick et al., 2010). Muchos vehículos híbridos combinan dos tipos diferentes de almacenamiento de energía eléctrica (HESS, *Hybrid Energy Storage System*): condensadores con baja capacidad de almacenamiento pero que permiten dar respuesta rápida a una demanda puntual alta de energía y baterías con gran capacidad de almacenamiento pero baja capacidad de reacción a picos inesperados. Los objetivos perseguidos a la hora de establecer una estrategia de carga de los sistemas de almacenamiento son maximizar la cantidad de energía almacenada para poder hacer frente a picos de energía inesperados y minimizar el deterioro de baterías y condensadores debido a las continuas cargas y descargas de los mismos. El deterioro es mayor en las baterías, ya que los condensadores soportan varios miles de veces más ciclos de carga; sin embargo el alimentar el sistema con los condensadores, con baja capacidad de almacenamiento, hace que éstos puedan no tener energía suficiente para hacer frente a un pico de demanda (p.ej. fuerte viento en contra o pendiente elevada).

- *Patrones de corte* (Soeiro Ferreira et al., 1990) Las empresas madereras reciben pedidos de segmentos de tronco de diferentes longitudes. Los cortes en un tronco se realizan posicionando diferentes cuchillas sobre el mismo y realizando todos los cortes simultáneamente. El tiempo de posicionamiento es alto, con lo cual interesa minimizar la cantidad de patrones de corte diferente para ajustar las

cuchillas el menor número de veces, pero al mismo tiempo minimizando también la cantidad de troncos necesarios para servir todos los pedidos recibidos.

Otros ejemplos de problemas multiobjetivo son la planificación independiente de dominio (Refanidis y Vlahavas, 2003), el enrutamiento de canales en circuitos VLSI (*Very-large-scale integration*) (Dasgupta et al., 1999) (Deutsch, 1988) o el enrutamiento de tráfico en redes multimedia (Clímaco et al., 2003),

3.3. Formalismos de la búsqueda multiobjetivo

La principal diferencia existente entre un problema de búsqueda con un objetivo y un problema de búsqueda multiobjetivo es que, en este último, los costes asociados a un operador u acción, en lugar de estar definidos por un valor escalar, vienen dados por un vector de dimensión q igual al número de objetivos considerados ($f(\vec{n}) = \{c_1, \ldots, c_q\}$). Antes de definir un problema de búsqueda multiobjetivo introduciremos la definición de una serie de relaciones entre vectores, necesarias para extender, entre otras, la noción habitual de optimalidad para el caso multiobjetivo.

Definición 3.3.1. *Sean dos vectores $\vec{v}, \vec{v'} \in \mathbb{R}^q$. La relación de orden parcial $\prec$ (denotada como relación de **dominancia** o **dominancia estricta**) se define como sigue:*

$$\vec{v} \prec \vec{v'} \Leftrightarrow \forall i(1 \leq i \leq q), (v_i \leq v'_i) \wedge (\vec{v} \neq \vec{v'}) \tag{3.1}$$

donde v_i denota el i-ésimo elemento del vector $\vec{v}$.

Definición 3.3.2. *Sean dos vectores $\vec{v}, \vec{v'} \in \mathbb{R}^q$. La relación de orden parcial $\preceq$ (denotada como relación de **dominancia débil** o **"domina o iguala"**) se define como sigue:*

$$\vec{v} \preceq \vec{v'} \Leftrightarrow \forall i(1 \leq i \leq q), (v_i \leq v'_i) \tag{3.2}$$

En el caso de dominancia estricta, un vector $\vec{v}$ domina a otro $\vec{v'}$ si todas las componentes de $\vec{v}$ son menores o iguales que las de $\vec{v'}$ y ambos vectores son distintos. Esto conlleva implícitamente que por lo menos una de las componentes de $\vec{v}$ sea estrictamente menor que la correspondiente de $\vec{v'}$.

Dados dos vectores $\vec{v}, \vec{v'} \in \mathbb{R}^q$ con $q > 1$, no siempre podemos establecer una relación de dominancia entre ellos. Por ejemplo, en el caso de vectores bidimensionales, el vector $(2, 3)$ domina al vector $(2, 4)$, pero no domina al vector $(3, 2)$. De modo análogo, el vector $(3, 2)$ tampoco domina al vector $(2, 3)$.

Definición 3.3.3. *Sean dos vectores $\vec{v}, \vec{v'} \in \mathbb{R}^q$. La relación de orden parcial $\sim$ (denotada como relación de indiferencia) se define como sigue:*

$$\vec{v} \sim \vec{v'} \Leftrightarrow (\vec{v} \not\prec \vec{v'}) \wedge (\vec{v'} \not\prec \vec{v}) \tag{3.3}$$

Es decir, $\vec{v} \sim \vec{v'}$ si $\vec{v}$ no domina a $\vec{v'}$ y a su vez no está dominado por $\vec{v'}$. Los vectores $(2, 3)$ y $(3, 2)$ son indiferentes o no dominados entre sí.

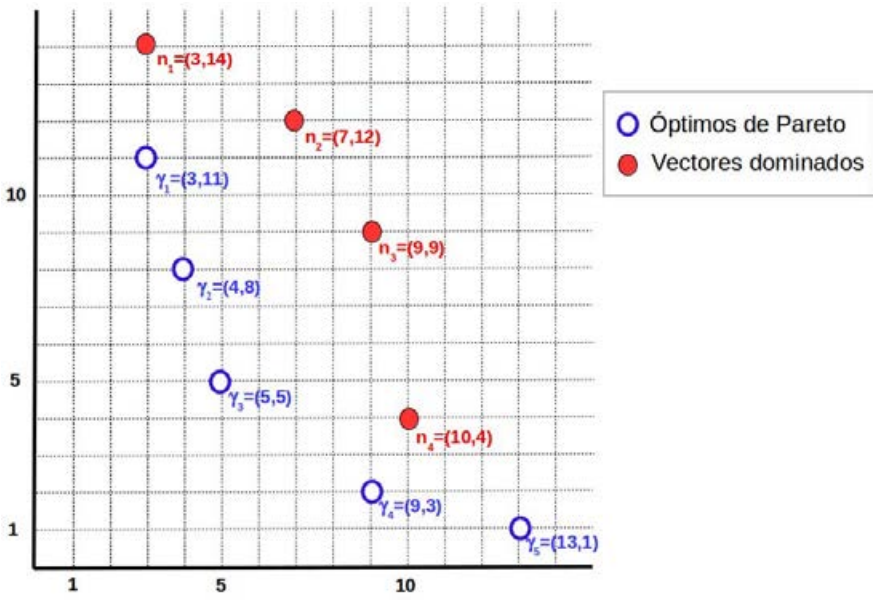


Figura 3.2: Frontera de Pareto de un conjunto de vectores

Definición 3.3.4. *Dado un conjunto de vectores X, se definen los óptimos de Pareto o Frontera de Pareto de dicho conjunto, $nd(X)$, como el conjunto de vectores no dominados de X, es decir:*

$$nd(X) = \{\vec{x} \in X \mid \nexists \vec{y} \in X \quad \vec{y} \prec \vec{x}\} \tag{3.4}$$

En la figura 3.2 se ilustra la Frontera de Pareto de un conjunto de vectores bidimensionales. Las siguientes relaciones, entre otras, excluyen a los vectores dominados de la Frontera de Pareto: $(3,11) \prec (3,14)$, $(4,8) \prec (7,12)$, $(5,5) \prec (9,9)$, $(9,3) \prec (10,4)$.

Vistas las definiciones anteriores, procedemos a definir un problema de búsqueda multiobjetivo. Un Problema de Búsqueda Multiobjetivo se representa por un grafo G finito, dirigido y etiquetado: $G = (N, s, \Gamma, A, \vec{c})$. El grafo tiene $|N|$ nodos y $|A|$ arcos $(n, n')/n, n' \in N$. Cada uno de los arcos (n, n') está etiquetado con un vector q-dimensional $\vec{c}(n, n') = (c_1, \ldots, c_q)$ $(\vec{c}(n, n') \in \mathbb{R}^{q^+})$, donde $c_i, i = 1, \ldots, q$ son valores positivos que representan el coste asociado al i-ésimo objetivo para ese arco. $s \in N$ representa el nodo inicial del problema y el conjunto $\Gamma \subseteq N$ representa el conjunto de nodos finales del problema.

De modo análogo al caso de un objetivo, el coste asociado a un camino $P = (n_0, n_1, \ldots, n_{k-1}, n_k)$, denotado por $\vec{g}(P)$, se calcula como la suma de los vectores de coste de los arcos que componen dicho camino. Obviamente este coste es un vector de dimensión q.

$$\vec{g}(P) = (g_1, \ldots, g_q) \quad g_i = \sum_{j=0}^{k-1} c_i(n_j, n_{j+1}) \quad 1 \leq i \leq q \tag{3.5}$$

Resolver el problema de búsqueda multiobjetivo anterior consiste en encontrar un conjunto $\mathbb{P}^* = \{P_1^*, \ldots, P_k^*\}$ de todos los caminos posibles tales que verifican lo siguiente:

- $P_i^* = (s, n_1, \ldots, \gamma), \gamma \in \Gamma, \forall P_i^* \in \mathbb{P}^*$, es decir, cualquiera de los caminos localizados va desde el nodo inicial hasta un nodo final del problema.
- $\vec{g}(P_i^*) \sim \vec{g}(P_j^*), \forall P_i^*, P_j^* \in \mathbb{P}^*$, es decir, los vectores de coste de todos los caminos localizados son indiferentes entre sí.
- Para cualquier otro camino solución P se verifica que $\vec{g}(P) \nprec \vec{g}(P_i^*)$.

Por convenio, el conjunto de vectores de coste de los caminos de $\mathbb{P}^*$ se denota por C^*. El conjunto $\mathbb{P}^*$ representa el conjunto de los caminos solución del problema con coste óptimo. Por lo tanto en la búsqueda multiobjetivo se redefine el concepto de optimalidad de un camino. Mientras que en la búsqueda con un objetivo la solución o soluciones óptimas eran aquellas con menor coste escalar, en la búsqueda multiobjetivo una solución óptima es aquella cuyo coste no está dominado por el coste de ninguna otra solución.

En la búsqueda multiobjetivo también se contempla el uso de información heurística para dirigir de modo más eficiente al algoritmo. Se define $H(n)$ como el conjunto de vectores heurísticos no dominados del nodo n, los cuales estiman el coste de alcanzar desde dicho nodo n un nodo final. Dado que un problema puede tener varios nodos finales y que cada uno puede ser alcanzado por varios caminos no dominados, es posible tener varias estimaciones heurísticas para cada nodo.

Definición 3.3.5. *Una función heurística multiobjetivo $H(n)$ es admisible (Mandow y Pérez de la Cruz, 2010) cuando para todos los caminos solución no dominados a una solución P^* = (s, n_1, ...,n_i,n_{i+1}, ...,γ_k), $\gamma_k \in \Gamma$ y cada subcamino P_i^* = (s,n_1, ...,n_i) de P^* existe un vector $\vec{h} \in H(n_i)$ tal que $\vec{g}(P_i^*) + \vec{h} \preceq \vec{g}(P^*)$.*

Es decir, se redefine el concepto de admisibilidad en base a la relación de dominancia. Por extensión definimos $F(n)$ como el conjunto de evaluaciones no dominadas del nodo n, calculadas como $\vec{f}(n) = \vec{h}(n) + \vec{g}(n)$, donde $\vec{h}(n) \in H(n)$.

3.4. Clasificación de algoritmos de búsqueda Multiobjetivo

En (Mandow y Pérez de la Cruz, 2003) se propone un *framework* general para algoritmos de búsqueda heurística multicriterio, incluyendo la formalización de la clase de problemas multicriterio y los componentes básicos de un procedimiento general de búsqueda.

Se han propuesto muchos métodos, tanto exactos como metaheurísticos, para resolver problemas de búsqueda multiobjetivo en grafos (Li et al., 2010). Ejemplos de

algoritmos aproximados multiobjetivo se pueden consultar en (Liu et al., 2011), (Perny y Spanjaard, 2008) o (Gonzalez-Rodriguez et al., 2010). En esta sección haremos un resumen de diferentes algoritmos multiobjetivo **exactos** u **óptimos**, es decir, que devuelven el conjunto completo de soluciones no dominadas del problema. El enfoque será similar al empleado en la sección 2.4 para los algoritmos con un único objetivo, mostrando los más representativos de las clases *best-first* y *depth-first*, con especial énfasis en estos últimos.

Muchos de estos algoritmos multiobjetivo son adaptaciones más o menos directas de sus equivalentes para un único objetivo. Es el caso, por ejemplo, del algoritmo MOA^*, versión multiobjetivo de A^*, o $IDMOA^*$, versión multiobjetivo de IDA^*.

Como ya vimos en el Capítulo 2, en el caso de la búsqueda en árboles los algoritmos para resolución de problemas de un único objetivo tienen que enfrentarse a una complejidad exponencial en el peor caso en cuanto al número de nodos examinados en función de la profundidad de la solución. Los algoritmos multiobjetivo son, en general, computacionalmente más exigentes. Para problemas con q objetivos, los costes y las estimaciones vienen dadas por vectores q-dimensionales. Esto hace que cualquier operación sea más costosa que en el caso de un objetivo, donde sólo trabajamos con valores escalares. Es necesario realizar operaciones vectoriales para computar nuevos costes, y especialmente para comprobar si las ramas están dominadas por umbrales en el caso de búsquedas acotadas o para comprobar si las soluciones ya localizadas dominan los caminos explorados.

Además, la naturaleza vectorial de los costes asociados a las soluciones tiene una gran repercusión en el número de soluciones óptimas. La frontera de Pareto crece exponencialmente con la profundidad de la solución en el peor de los casos (Hansen, 1979), incluso con dos objetivos, con lo cual el tamaño del árbol explorado suele ser considerablemente mayor que para el caso de un objetivo, donde es suficiente con devolver una única solución. Es interesante señalar que existen varios tipos de problemas multiobjetivo de interés que no presentan ese comportamiento (Müller-Hannemann y Weihe, 2006) (Mandow y de la Cruz, 2009).

De igual modo, el rango de costes para los diferentes objetivos también tiene un impacto importante en la dificultad del problema. A mayor amplitud del rango, mayor cantidad de posibles óptimos de Pareto. Por ejemplo, para un rango de costes entero [1,3], el conjunto de óptimos de Pareto viene dado por $\{(1,3),(2,2),(3,1)\}$. Sin embargo, para un rango [1,5] este conjunto es $\{(1,5),(2,4),(3,3),(4,2),(5,1)\}$. En caso de tratar con costes no acotados, como ya hemos comentado, la frontera de Pareto puede crecer exponencialmente.

Al igual que en el caso de un solo objetivo, en la búsqueda multiobjetivo exacta se han desarrollado algoritmos tanto con un enfoque *best-first* como *depth-first*. En cualquier caso la problemática de los algoritmos es similar a la de los ya vistos en la sección 2.8, pero con el *agravante* de tener que tratar con costes multidimensionales, lo cual requiere mayor coste computacional. De modo general, los enfoques *best-first* presentan un coste temporal menor trabajando sobre grafos, expandiendo exclusiva-

mente los nodos necesarios, pero a costa de unos requisitos de memoria muy elevados. Estos algoritmos *best-first* tienen un sobrecoste temporal derivado de la ordenación de la lista de nodos pendientes de expansión. Por otro lado, las aproximaciones *depth-first* consiguen un ahorro importante de memoria, manteniendo un consumo lineal para la exploración al guardar en memoria únicamente la rama actual. Sin embargo, este coste espacial puede ser exponencial si así lo fuera el tamaño del conjunto C^*, al tener que almacenarlo en memoria.

3.5. Búsqueda multiobjetivo *best-first*

Una de las primeras aproximaciones multiobjetivo de tipo *best-first* con uso de información heurística (cotas inferiores) fue el algoritmo MOA^* (*MultiObjective* A^*) (Stewart y White, 1991), el cual es una adaptación del algoritmo A^* (Hart et al., 1968) descrito en la sección 2.6. Con un heurístico monótono, A^* garantiza que cualquier nodo expandido en el árbol de búsqueda lo será a través de su camino óptimo. La situación en un problema multiobjetivo varía radicalmente, puesto que a un mismo nodo pueden llegar varios caminos óptimos, los cuales tendrán vectores de coste no dominados entre sí. Para tratar esta situación, MOA^* mantiene un grafo acíclico que recoge todos los subcaminos óptimos encontrados hasta el momento hasta cada nodo generado. El nodo seleccionado para expansión debe poseer un vector de evaluación no dominado por ningún otro del resto de nodos pendientes. Cuando MOA^* selecciona un nodo para expansión, se expanden simultáneamente todos los caminos no dominados incluídos en su grafo acíclico asociado hasta ese nodo. Un trabajo reciente (Pérez-de-la-Cruz et al., 2013) ha demostrado como MOA^*, en algunas clases de problemas, degrada considerablemente su rendimiento cuanto más informado era el heurístico empleado.

(Navinchandra, 1991) presenta una extensión del algoritmo A^* para espacios de estados con forma de árbol, aunque su repercusión ha sido menor al ser MOA^* más generalista y poder tratar con cualquier tipo de grafo. En (Sykes y White, 1991) y (Navinchandra, 1991) se presentan aplicaciones prácticas de estos algoritmos. (Tung y Chew, 1992) presenta otra variante *best-first* heurística que generaliza A^* y expande caminos en lugar de nodos, proponiendo asimismo dos heurísticos genéricos precalculados antes de realizar el proceso de búsqueda.

El trabajo de Mandow y Pérez de la Cruz (Mandow y Pérez de la Cruz, 2005) presenta $NAMOA^*$ (*New Approach MultiObjective A**), una alternativa a MOA^* cuya principal característica es que, frente a la expansión de nodos realizada por éste último, $NAMOA^*$, al igual que el algoritmo presentado en (Tung y Chew, 1992), expande caminos, pero mejorando el proceso de filtrado y poda. $NAMOA^*$ usa una lista ($OPEN$) de nodos pendientes de expansión, similar a la utilizada por A^*. Cada elemento de esta lista está formado por una tupla de la forma $(n, \vec{g}, F(n, \vec{g}))$ donde n es un nodo susceptible de ser expandido, $\vec{g}$ es un vector de coste de un camino no dominado desde el nodo raíz al nodo n y $F(n, \vec{g})$ está formado por un conjunto de vectores de coste resultantes de sumar a $\vec{g}$ las estimaciones heurísticas asociadas al nodo n ($H(n)$).

Normalmente $H(n)$ está formado por un único vector, y por lo tanto también lo estará $F(n, \vec{g})$, aunque en el caso general ambos pueden ser conjuntos de vectores. Es decir, por cada nodo puede haber varias tuplas en la lista $OPEN$ que reflejan cada uno de los caminos no dominados a dicho nodo que han sido generados durante el proceso de búsqueda.

Cuando varios caminos llegan a un mismo nodo, aquellos que están dominados se descartan (operación de **poda**). Asimismo, los vectores de $F(n, \vec{g})$ (generalmente 1) dominados por el vector de coste de alguna solución se eliminan (operación de **filtrado**).

Cuando la lista $OPEN$ se queda vacía, el algoritmo finaliza y pueden recuperarse los caminos a soluciones no dominadas a partir de un grafo que $NAMOA^*$ mantiene durante todo el proceso de búsqueda. Este grafo tiene como raíz al nodo inicial del problema y contiene el conjunto de todos los nodos visitados por el algoritmo junto con los caminos no dominados a esos nodos que se han encontrado.

Se ha demostrado que $NAMOA^*$ es óptimo en la clase de algoritmos admisibles (es decir, que ningún algoritmo de esa clase puede dejar de considerar ninguna etiqueta (n, g) considerada por $NAMOA^*$), y que $NAMOA^*$ es más eficiente que MOA^* porque sólo expande cada etiqueta una vez y nunca expande etiquetas que correspondan a caminos dominados. Las pruebas realizadas muestran que $NAMOA^*$ realiza un consumo de memoria considerablemente menor que MOA^*. Además, $NAMOA^*$ puede ser usado como algoritmo *anytime*, deteniendo su ejecución en cualquier momento y devolviendo el subconjunto de óptimos de Pareto encontrados hasta entonces. $NAMOA^*$ no presenta la patología de MOA^* presentada en (Pérez-de-la-Cruz et al., 2013).

Algunos autores han estudiado algoritmos específicos que buscan una solución no dominada que responda a una preferencia enunciada a priori, en lugar de intentar encontrar toda la frontera de Pareto. Un ejemplo es el de la búsqueda compromiso. Una búsqueda compromiso multiobjetivo (Galand y Spanjaard, 2012) trata de minimizar una función de agregación aplicada a los vectores de coste. Esta función suele aplicar pesos a los diferentes objetivos considerados. La función de agregación s debe cumplir que cualquier solución no dominada puede ser óptima con la parametrización adecuada de s, y que cualquier solución óptima de s debe ser no dominada. Este tratamiento escalar permite reducir los requisitos de memoria e incrementar la velocidad de resolución. Garland y Perny (Galand y Perny, 2006) proponen dos nuevos algoritmos, BCA^* y kA^*, el primero de ellos una adaptación directa de la búsqueda compromiso y el segundo un algoritmo que trabaja sobre una escalarización del problema. Garland y Spanjaard (Galand y Spanjaard, 2012) proponen una búsqueda compromiso con una asignación dinámica de pesos a los objetivos basada en su coste individual (*Ordered Weighted Average*), asignando mayor peso a aquellos objetivos con mayor coste dentro del vector. (Machuca et al., 2013) evalúa el algoritmo kA^* (Galand y Perny, 2006) y una variante de $NAMOA^*$ que determina la solución de mejor compromiso una vez obtenida la frontera de Pareto. Esta segunda opción se comporta de modo más predecible y efectivo a medida que aumenta la dificultad del problema resuelto.

En (Pulido et al., 2012) y (Galand et al., 2013) se presentan sendas variantes de la búsqueda bidireccional al caso multiobjetivo. En (Pulido et al., 2014) se propone un algoritmo de búsqueda multiobjetivo con preferencias basadas en metas lexicográficas. Dado un conjunto de metas (Romero, 1993) (como ya vimos en la sección 3.1, combinación de atributo y nivel de aspiración), el algoritmo localiza los óptimos de Pareto satisfactorios, es decir, que satisfacen todo el conjunto de metas. En caso de no haber soluciones satisfactorias, busca los óptimos de Pareto que minimizan la desviación de los objetivos. Las metas se agrupan por nivel de prioridad, midiendo la desviación por cada uno de estos niveles.

3.6. Búsqueda multiobjetivo *depth-first*

Al igual que en el caso de un único objetivo, la complejidad espacial es el principal problema al que se enfrentan los algoritmos *best-first* multiobjetivo. Los algoritmos *depth-first* multiobjetivo que veremos en esta sección ofrecen, al igual que sus equivalentes para un objetivo, un consumo lineal de memoria en la exploración del árbol de búsqueda, a costa de reexpansión de nodos o expansión de nodos subóptimos. Sin embargo, este consumo puede ser también exponencial si así lo fuera el conjunto de soluciones óptimas del problema, puesto que dicho conjunto debe ser almacenado en memoria. Siguiendo un paralelismo con los algoritmos vistos en el Capítulo 2, los algoritmos exactos analizados en esta sección son *MO-DF-BnB*, $IDMOA^*$ y $MOMA^*0$, unas generalizaciones más o menos fieles de los algoritmos *DF-BnB*, IDA^* y $RBFS$. Todos estos algoritmos son exactos, ya que devuelven el conjunto completo de óptimos de Pareto del problema.

3.6.1. *DF-BnB* multiobjetivo

DF-BnB para un único objetivo ha resultado ser un algoritmo muy eficiente en la exploración de espacios de estados en forma de árboles acotados (Zhang y Korf, 1995). Es por ello que la generalización de *DF-BnB* es una aproximación bastante natural al caso multiobjetivo. En el capítulo 6 se presentará de modo detallado una variante de *DF-BnB* multiobjetivo. De todos modos, el funcionamiento básico de un algoritmo *DF-BnB* multiobjetivo es prácticamente idéntico al de su equivalente para un objetivo.

Las ramas del árbol de búsqueda se exploran en modo *depth-first*, tomando como cota superior los costes de las soluciones que se vayan localizando durante la expansión. La diferencia con el caso de un objetivo es que la cota no será un valor escalar, sino un conjunto de vectores de coste no dominados asociados a los costes de los caminos solución óptimos que se vayan localizando. Cuando en el árbol de búsqueda se localiza un nodo final, su vector de coste se añade a la cota actual, siempre y cuando no esté dominado por algún vector de dicha cota (lo cual indica que sería un nodo final dominado, es decir, subóptimo). Asimismo, cada vez que se localiza un nodo final, su vector de coste elimina de la cota los vectores dominados. Cuando el algoritmo finaliza

al no haber más ramas que explorar, la cota contiene el conjunto C^*.

Dado que inicialmente *DF-BnB* explora el árbol de búsqueda sin cota, al no disponer de información relativa a ninguna solución, es posible que localice soluciones subóptimas, lo cual obliga a realizar tests de dominancia cuando encuentra nuevas soluciones para descartar las dominadas. Otro problema de este algoritmo es que esta situación sin cota transitoria (mientras no encuentra una solución) se puede convertir en permanente si *DF-BnB* cae en una rama infinita, donde cada nodo de la rama siempre tiene sucesores y ninguno de ellos es un nodo meta. En este caso el algoritmo no finalizaría. Estos problemas se tratarán con detalle en el capítulo 6, solventando el último mediante una búsqueda de dos fases.

Han sido numeros los análisis y propuestas sobre algoritmos de tipo *DF-BnB* multiobjetivo, entre los cuales caben destacar (Sourd y Spanjaard, 2008), (Rollón y Larrosa, 2009) ó (Delort y Spanjaard, 2013). (Sourd y Spanjaard, 2008) implementan una versión del algoritmo *Branch & Bound* con una cota superior empleada de modo similar a como lo hace *DF-BnB*. Esta cota superior o *Upper Bound* (UB), contiene el conjunto de costes no dominados de las soluciones que se van encontrando. El algoritmo incluye una fase de inicialización para construir una aproximación del Frente de Pareto, el cual se usará como cota superior inicial.

Para mejorar el rendimiento de este enfoque *DF-BnB* multiobjetivo, antes de expandir un nodo n se calcula una cota inferior de los costes de los nodos finales alcanzables a través de dicho nodo n. Si esta cota inferior está dominada por los vectores del actual conjunto UB, entonces todas las soluciones que incluyan a n en su camino son subóptimas y por lo tanto puede descartarse la exploración del nodo n.

En la figura 3.3 se muestra un ejemplo de este procedimiento para un problema con dos objetivos. El conjunto UB está formado por la cota superior habitual de los algoritmos de tipo *DF-BnB*. La región $UB^{\prec}$ representa el área del espacio de búsqueda dentro de la cual se encuentra los óptimos de Pareto. Denominamos $X(n)$ el conjunto de nodos meta alcanzables a través del nodo n, denotando por $F(X(n))$ los vectores de coste de dichas soluciones. Si es posible construir una función $s(v_1, v_2)$ tal que que separe $UB^{\prec}$ de $F(X(n))$, entonces podemos afirmar que el conjunto $X(n)$ está formado por soluciones subóptimas y no tiene interés expandir el nodo n. Esta función s verifica:

$$s(\vec{v}) < 0, \forall \vec{v} \in UB^{\prec} \tag{3.6}$$

$$s(\vec{w}) \geq 0, \forall \vec{w} \in F(X(n)) \tag{3.7}$$

Las principales aportaciones de (Sourd y Spanjaard, 2008) son la aproximación del conjunto $F(X(n))$ con una cota inferior (LB, *Lower Bound*) y la construcción de la función s. El algoritmo ha sido aplicado al problema de *Spanning Tree* multiobjetivo, de tipo *NP-difícil*, el cual tiene un árbol de búsqueda claramente acotado en profundidad. En la línea de este trabajo, (Delort y Spanjaard, 2013) proponen un algoritmo de dos fases basado en programación dinámica, en el cual una fase calcula un subconjunto de los óptimos de Pareto. En una segunda fase, a partir de estos óptimos se definen una

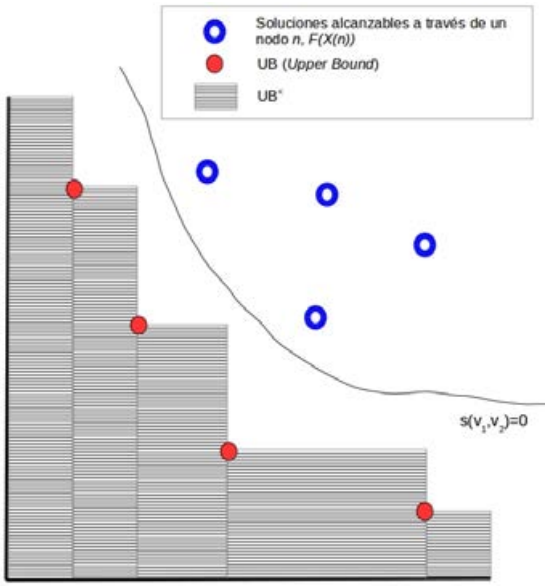


Figura 3.3: Ejemplo cotas inferior y superior para *DF-BnB* multiobjetivo

serie de áreas en las cuales se encuentran el resto de soluciones óptimas. La búsqueda en estas áreas se realiza mediante *DF-BnB*.

(Rollón y Larrosa, 2009) propone el uso de una cota inferior similar para las soluciones alcanzables a través de un nodo, pero en este caso calculada con el algoritmo $MO-MBE$ (*MultiObjective Mini-bucket Elimination*) (Rollón y Larrosa, 2006). En cualquiera de los dos casos, los esfuerzos están centrados en minimizar la cantidad de nodos subóptimos expandidos por *MO-DF-BnB*, realizando las podas antes de llegar a los nodos que excedan las cotas establecidas.

Otros trabajos de interés pueden consultarse en (White, 1982), (White, 1984), (Bausch, 1992), (Galand et al., 2008), (Galand et al., 2010) y (Buer y Pankratz, 2010).

Las variantes multiobjetivo de *DF-BnB* mencionadas en esta sección han sido probadas con éxito en problemas reales con forma de árbol acotado. Sin embargo, su aplicación en problemas con espacios de estados infinitos sólo es viable si se proporciona al algoritmo una cota superior inicial, debido a que si el algoritmo explora una rama infinita antes de haber localizado la primera solución, entonces no finalizará. En el Capítulo 6 se analizará un algoritmo *MO-DF-BnB* con cota superior inicial.

Los algoritmos de tipo *DF-BnB*, como ya se comentaba en la sección 2.7.1, presentan problemas de ineficiencia en los espacios de estados en los cuales a un mismo nodo se pueda llegar a través de diferentes caminos. Estos problemas de ineficiencia también se mantienen en sus equivalentes multiobjetivo.

3.6.2. Profundización iterativa multiobjetivo: algoritmo $IDMOA^*$

Tal y como ya se comentó en el Capítulo 2, dentro de la búsqueda *depth-first* los algoritmos que realizan profundización iterativa (ID, *Iterative Deepening*) permiten trabajar con espacios infinitos o con bucles conservando un consumo lineal de memoria. Sin embargo hasta la fecha sólo se ha explorado superficialmente la adaptación de la estrategia ID al caso multiobjetivo. Hasta donde alcanza nuestro conocimiento, el único algoritmo destacable de este tipo es $IDMOA^*$ (*Iterative Deepening Multiobjective A**) (Harikumar y Kumar, 1996), el cual es una extensión más o menos directa del algoritmo IDA^* (Korf, 1985a).

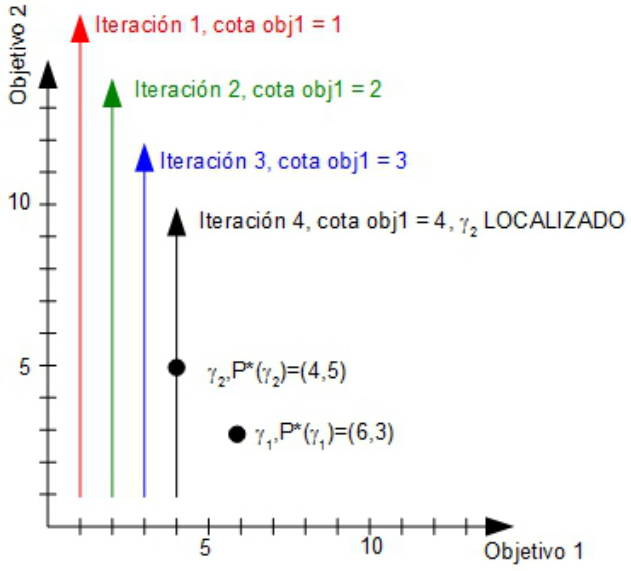
Una de las principales diferencias entre ambos algoritmos es que si bien IDA^*, con un heurístico admisible, puede detener la búsqueda en cuanto localiza la primera solución, garantizando que ésta es óptima, $IDMOA^*$ debe seguir buscando hasta encontrar todos los óptimos de Pareto. Para ello emplea una estrategia en la cual procesará cada uno de los objetivos involucrados en la búsqueda de modo independiente y consecutivo, es decir, realiza una secuencia de búsquedas con profundización iterativa, una para cada objetivo considerado. Es conveniente resaltar que, por este motivo, $IDMOA^*$ no es realmente una generalización directa de la profundización iterativa al caso multiobjetivo, sino un algoritmo que aplica esta técnica por fases, de modo secuencial a cada uno de los objetivos considerados, introduciendo controles adicionales para el descarte de soluciones dominadas.

A continuación explicaremos el funcionamiento básico de $IDMOA^*$ apoyándonos en la figura 3.4, la cual presenta el espacio de costes[1] para un problema considerando dos objetivos y una frontera de Pareto formada por los vectores $\{(4,5),(6,3)\}$. Siendo s el nodo inicial, suponemos que $\vec{h}(s) = (1,1)$ y $\vec{h}(n) = (0,0)\ \forall n \neq s$. Suponemos además costes enteros.

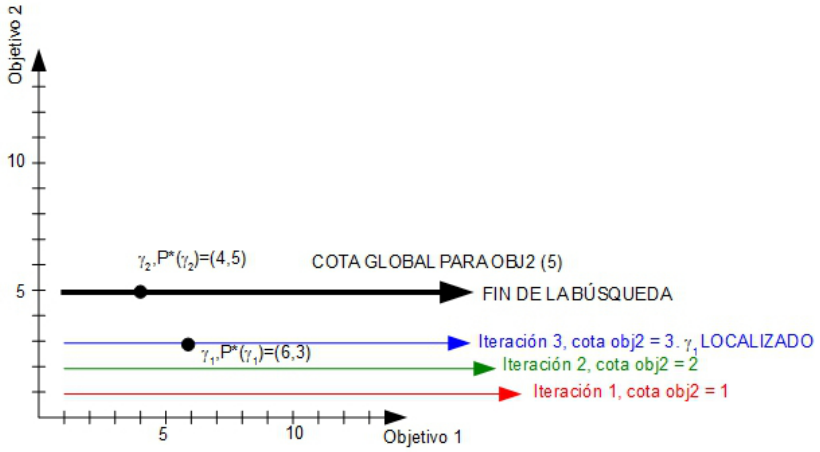
Las búsquedas asociadas al primer objetivo se realizan de modo similar a como lo haría el algoritmo IDA^*, con la idea de localizar todas aquellas soluciones al problema cuyo coste asociado al primer objetivo sea óptimo. Al igual que en IDA^*, la cota inicial de expansión para la búsqueda viene dada por el heurístico del nodo raíz, en este caso, el valor de este heurístico para el primer objetivo, $h_1(s)$. En el ejemplo, esta cota inicial vale 1. El hecho de que se mantenga una cota de naturaleza escalar tendrá connotaciones positivas en el rendimiento de $IDMOA^*$.

A continuación se expande el árbol de búsqueda del mismo modo en que lo haría IDA^*, hasta que el algoritmo encuentra la primera solución. En la figura 3.4a podemos observar como para la primera búsqueda *depth-first*, con cota 1 para el primer objetivo, no se localiza ninguna solución (la región entre el eje Y y la flecha roja que representa la cota actual representa el área de búsqueda de la iteración actual en el espacio de costes). Por lo tanto, se procede a actualizar la cota de modo similar a como lo haría IDA^*, escogiendo el menor valor $f_1(n)$ de entre todos los nodos n en los cuales la

[1] El espacio de costes representa el conjunto completo de valores posibles para los vectores de coste de un problema.



(a) Primera fase de $IDMOA^*$ (consideración del objetivo 1)



(b) Segunda fase de $IDMOA^*$ (consideración del objetivo 2)

Figura 3.4: Procesamiento de objetivos en $IDMOA^*$

búsqueda haya sido discontinuada.

Se procede de modo análogo en las siguientes iteraciones suponiendo que las cotas sucesivas toman los valores 2, 3 y 4. En esta última iteración se localiza la solución γ_2 con coste $(4, 5)$. En este punto se produce la primera diferencia de $IDMOA^*$ con IDA^*, ya que $IDMOA^*$ no se detiene al localizar la primera solución que no sobrepase la cota de la iteración actual, sino que sigue explorando el resto de ramas pendientes buscando otras soluciones con coste idéntico para el primer objetivo, pero que puedan mejorar los restantes, con la finalidad de devolver soluciones no dominadas y óptimas para dicho primer objetivo.

Supongamos que en el ejemplo anterior hubiera otra solución γ_3 con vector de coste $\vec{f}(\gamma_3) = (4, 4)$. Esta solución dominaría a γ_2, pero no tenemos asegurado que γ_3 se localizará antes que γ_2, pues ello dependerá del orden de exploración del árbol. Es por ello que $IDMOA^*$ continúa con la exploración. Si localizara alguna otra solución con coste idéntico para el primer objetivo, realizaría los correspondientes tests de dominancia para quedarse con el óptimo de Pareto.

Por lo tanto, como resultado de la profundización iterativa para el primer objetivo obtenemos un óptimo de Pareto el cual tiene el valor mínimo para el primer objetivo. A continuación $IDMOA^*$ realiza una búsqueda con profundización iterativa para el segundo objetivo, pero, a diferencia del primer objetivo, esta búsqueda incluirá una cota superior adicional proporcionada por los resultados de la búsqueda anterior.

Si al finalizar el procesamiento del primer objetivo $IDMOA^*$ ha localizado una solución con coste (c_1, c_2), sabemos que esta solución es el óptimo de Pareto con el valor mínimo para el primer objetivo, con lo cual cualquier solución que se localice durante el procesamiento de los restantes objetivos tendrá valor igual o superior para este primer objetivo, y por ello no dominará a esa primera solución. Por lo tanto cualquier otro óptimo de Pareto debe tener un valor superior a c_1 para el primer objetivo e inferior a c_2 para el segundo objetivo. Esto nos permite fijar una cota superior para este segundo objetivo, el valor 5, de tal modo que si la profundización iterativa asociada a ese objetivo la supera, tenemos garantizado que en exploraciones más profundas del árbol de búsqueda no habrá óptimos de Pareto para el segundo objetivo.

En el ejemplo anterior, al procesar el segundo objetivo, se establece como cota inicial el valor $h_2(s)$ y se procede de modo similar, realizando iteraciones y actualizaciones de cota. Sin embargo existe una cota superior complementaria que viene dada por el máximo valor para el segundo objetivo de entre todas las soluciones localizadas durante el procesamiento del primero ($\text{máx}(\vec{sol}[2]) \quad \forall \vec{sol} \in Solution$), tal y como se refleja en la figura 3.4b. Se realizan búsquedas en profundidad con cotas crecientes hasta alcanzar la cota global para el segundo objetivo. De modo análogo se procedería con los restantes objetivos.

Complementariamente a los retrocesos realizados cuando en una rama se excede la cota fijada, $IDMOA^*$ también discontinúa ramas cuando éstas están dominadas por alguna solución que ya haya sido localizada. (Harikumar y Kumar, 1996) demuestran que el algoritmo devuelve todos los óptimos de Pareto bajo los supuestos de costes no

negativos para todos los objetivos y un heurístico optimista.

En la tabla 3.1 se muestra el pseudocódigo de $IDMOA^*$. La función **DFS** es la encargada de realizar la búsqueda en profundidad para un objetivo y una cota determinada. Si se discontinúa una rama por haber superado su coste la cota, se recalcula la cota de la siguiente iteración. Si la rama explorada alberga una solución que no supere la cota actual, se añade al conjunto de soluciones, purgando el mismo con los correspondientes tests de dominancia. Si el nodo procesado no es solución ni supera la cota actual, entonces se expande, realizando llamadas recursivas a la función **DFS**.

La función **IDMOA*** es la encargada de procesar secuencialmente los objetivos realizando profundizaciones iterativas, calculando para cada uno de ellos la cota superior absoluta y las cotas consecutivas para las diferentes iteraciones.

3.6.2.1. Ejemplo de $IDMOA^*$

En esta sección presentaremos un ejemplo detallado del funcionamiento de $IDMOA^*$ para un árbol de búsqueda binario biobjetivo, con una frontera de Pareto $C^* = \{(6, 10), (10, 6)\}$. En la figura 3.5a se representa el espacio de estados así como el espacio de costes, con los valores del objetivo 1 en el eje X y los del objetivo 2 en el eje Y. Los nodos γ_1 y γ_2 son nodos finales del problema, y n_3 y n_4 son nodos hoja pero no nodos finales.

Cada nodo está etiquetado con su estimación heurística (en este ejemplo, un único vector) y cada arco está etiquetado con su vector de coste correspondiente. Supondremos una expansión de izquierda a derecha de las ramas del árbol.

$IDMOA^*$ procesa inicialmente toda la información relativa al objetivo 1. Se establece una cota inicial igual a $h(s)[1] = 5$, la componente del heurístico de s asociada al primer objetivo. En la parte derecha de la figura 3.5b se representa sombreada el área del espacio de costes que explorará esta primera iteración para el objetivo 1. Nótese que este área se prolonga indefinidamente sobre el eje Y, pues la información asociada al objetivo 2 no se procesa durante esta iteración, a excepción de los tests de dominancia cuando se localizan soluciones.

Tras fijar la cota del primer objetivo se realiza una búsqueda en profundidad, discontinuando la búsqueda en todos aquellos nodos cuya componente del vector de coste asociada al objetivo 1 exceda el umbral marcado. El primer nodo visitado, n_1, es discontinuado al ser $\vec{f}(n_1)[1] = 6 > 5$, mientras que el nodo n_2 sí es expandido al no superar la cota ($\vec{f}(n_2)[1] = 5 \leq 5$). Su sucesor n_4 también es procesado ($\vec{f}(n_4)[1] = 5 \leq 5$), pero al no tener sucesores, la búsqueda no puede continuar por dicha rama. Finalmente el nodo γ_2 también es discontinuado al ser $\vec{f}(\gamma_2)[1] = 6 > 5$.

Dado que $IDMOA^*$ no ha localizado ninguna solución, computa una nueva cota, la cual viene dada por el mínimo valor para el objetivo 1 de los vectores de coste de los nodos que han sido discontinuados por haber excedido la cota previa. En nuestro caso, estos nodos son n_1 y γ_2. El vector de coste de n_4 no se tiene en cuenta, ya que la

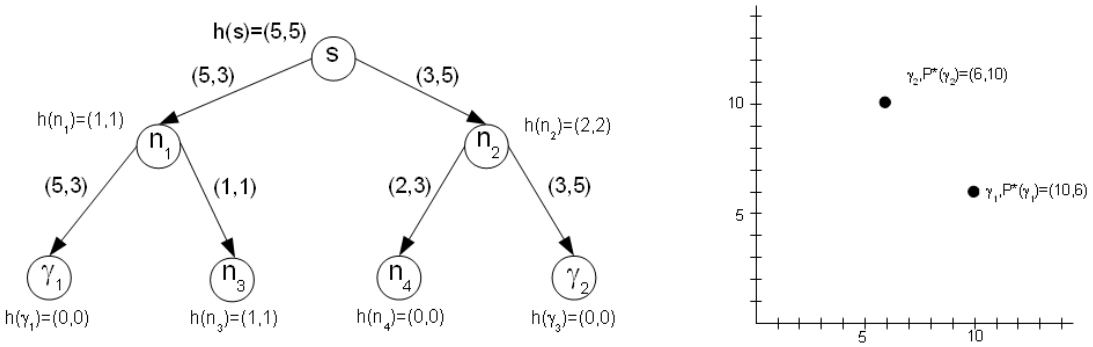
IDMOA* (G, s)

```
Solution ← ∅
objIndex ← 1
threshold ← h⃗(s)[objIndex]
WHILE (Solution <> ∅)
     next-threshold ← ∞
     DFSEARCH(s)
     threshold ← next-threshold
FOR objIndex = 2 TO ♯objectives
     MaxThreshold ← máx_{sol⃗∈Solution}({sol⃗[objIndex]})
     threshold ← h⃗(s)[objIndex]
     WHILE (threshold ≤ MaxThreshold)
          next-threshold ← ∞
          DFSEARCH(s)
          threshold ← next-threshold
return Solution
```

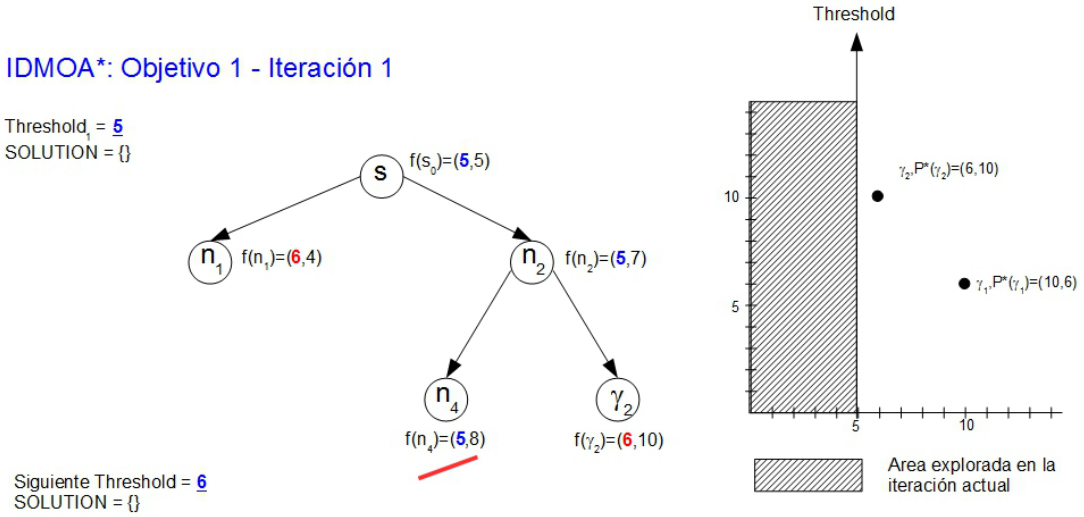
DFSEARCH(n)

```
IF (f⃗(n)[objIndex] > threshold) THEN
     next-threshold ← min(next-threshold, f⃗(n)[objIndex])
     RETURN
IF ((n ∈ Γ) ∧ (f⃗(n)[objIndex] ≤ threshold)) THEN
     Solution ← nd(Solution ∪ {f⃗(n)})
     RETURN
FOREACH n_i ∈ sucesores(n)
     IF ((f⃗(n_i)[objIndex] ≤ threshold) ∧ (∄c⃗ ∈ Solution | c⃗ ≺ f⃗(n_i))) THEN
          DFSEARCH(n_i)
RETURN
```

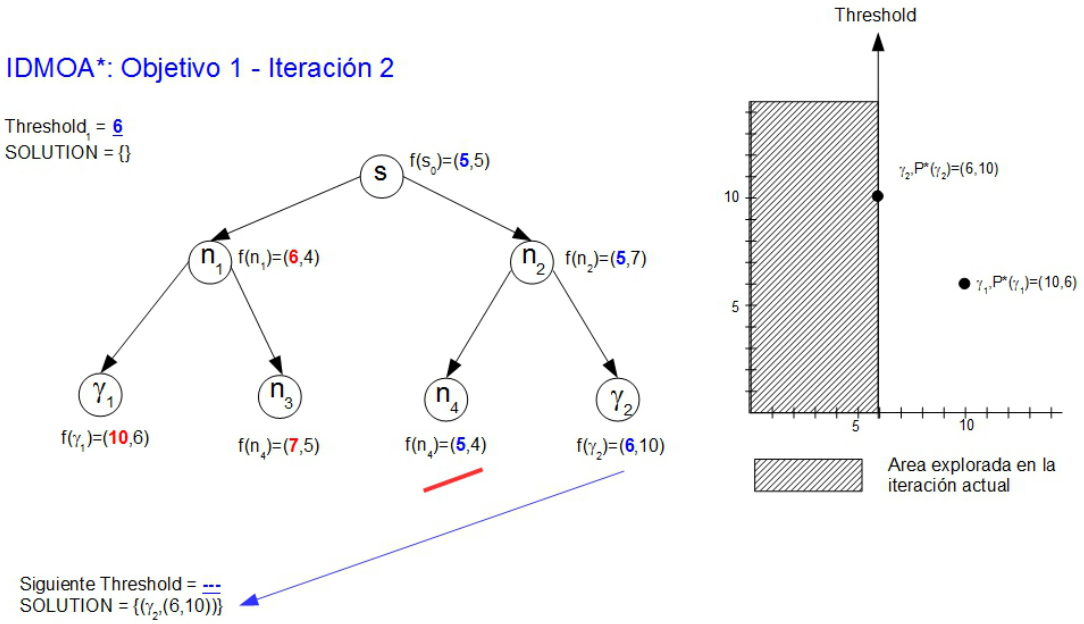
Tabla 3.1: Algoritmo IDMOA* (tomado de (Harikumar y Kumar, 1996))



(a) Problema Multiobjetivo y espacio de costes



(b) Iteración 1 $IDMOA^*$



(c) Iteración 2 $IDMOA^*$

Figura 3.5: Ejemplo de $IDMOA^*$: Problema e iteraciones 1-2

búsqueda no prosiguió en esa rama al no tener n_4 ningún sucesor. Es decir, la nueva cota se calcula como mín$\{\vec{f}(n_1)[1], \vec{f}(\gamma_2)[1]\}$ = mín$\{6, 6\} = 6$.

Una vez fijada la nueva cota, se procede a realizar otra búsqueda en profundidad, la cual se muestra en la figura 3.5c. Podemos apreciar que el área sombreada contiene el espacio de costes explorado en la iteración anterior, algo intrínseco a la profundización iterativa. Dado que la cota crece de modo estricto, todos los nodos expandidos en una iteración i se expandirán en la siguiente iteración $i + 1$.

Con la nueva cota, el nodo n_1 se expande, pero la búsqueda se discontinúa en sus dos sucesores, γ_1 y n_3, ya que ambos exceden la cota para el primer objetivo ($\vec{f}(\gamma_1)[1] = 10 > 6$ y $\vec{f}(n_3)[1] = 7 > 6$). En la rama derecha del nodo raíz el procesamiento es idéntico al de la iteración anterior, excepto en lo que respecta a γ_2, el cual no excede la cota ($\vec{f}(\gamma_2)[1] = 6 \leq 6$). Al ser un nodo final, se añade al conjunto *tentativo* de soluciones finales.

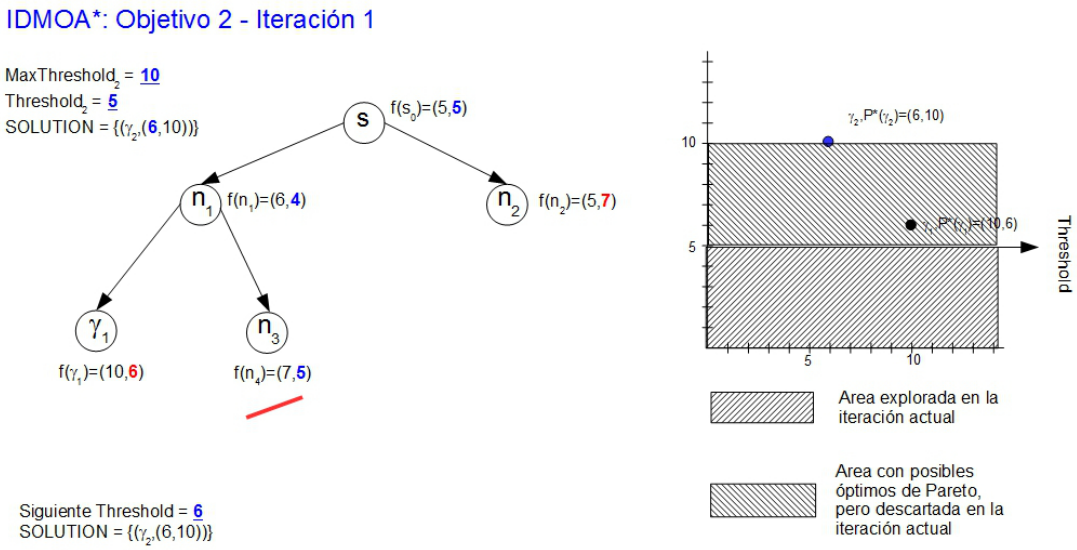
Tras finalizar la expansión del árbol de búsqueda, como el conjunto de soluciones no está vacío, $IDMOA^*$ asegura que hemos localizado un óptimo de Pareto con el mínimo valor para el primer objetivo ($SOLUTION = \{(6, 10)\}$). Cualquier otra búsqueda en profundidad con una cota superior daría soluciones con peor coste para este primer objetivo.

Realizado ya el procesamiento del primer objetivo, $IDMOA^*$ procede con el siguiente, fijando previamente una cota superior absoluta para los umbrales que marcarán la profundización iterativa del segundo objetivo. En nuestro ejemplo, esta cota superior toma el valor 10 (máximo valor para las componentes del segundo objetivo de los vectores de coste de las soluciones localizadas): máx$\{\vec{c}[2]\}\forall(\vec{c}, \gamma) \in SOLUTION$.

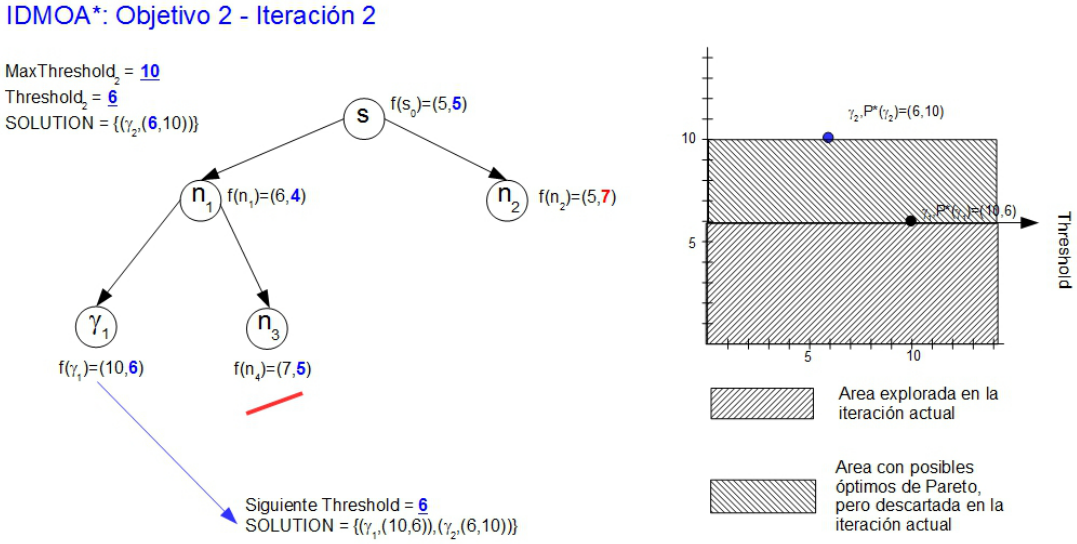
La primera iteración para este segundo objetivo puede verse en la figura 3.6a. Las dos áreas sombreadas representan el espacio de costes donde puede localizarse un óptimo de Pareto que no esté dominado por la solución ya localizada. Sin embargo, en esta primera iteración únicamente se explorará el área inferior, acotada por el valor heurístico del nodo raíz para el segundo objetivo, el cual ejerce de cota inferior inicial para el mismo.

Por lo tanto la cota inferior inicial para el segundo objetivo es $\vec{h}(s)[2] = 5$. En la expansión del árbol de búsqueda se añaden tests adicionales. Ya no se comprueba sólo si el coste escalar relativo al segundo objetivo del nodo excede la cota, sino también si su coste vectorial está dominado por alguna solución ya encontrada. En nuestro ejemplo el nodo n_1 puede expandirse ($\vec{f}(n_1)[2] = 4 \leq 5$), pero no así su sucesor γ_1, ya que excede el umbral actual ($\vec{f}(\gamma_1)[2] = 6 > 5$). El nodo n_3 sí se selecciona para expansión ($\vec{f}(n_3)[2] = 5 \leq 5$), pero al no tener sucesores se produce un *backtracking* (su vector de coste no se tendrá en cuenta por lo tanto para el cálculo del próximo umbral).

En la rama derecha del nodo raíz, el valor $\vec{f}(n_2)$ del nodo n_2 para el segundo objetivo excede la cota ($\vec{f}(n_2)[2] = 7 > 5$). Al haberse explorado ya todas las ramas hasta donde era posible, se calcula una nueva cota como el mínimo valor para la segunda componente (objetivo 2) de los vectores de coste de los nodos discontinuados en la



(a) Iteración 3 $IDMOA^*$



(b) Iteración 4 $IDMOA^*$

Figura 3.6: Ejemplo de $IDMOA^*$: Iteraciones 3-4

iteración anterior, γ_1 y n_2. Por lo tanto la nueva cota será mín$\{\vec{f}(\gamma_1)[2], \vec{f}(n_2)[2]\}$ = mín$\{6, 7\} = 6$. El espacio de búsqueda se expande, tal y como se refleja en la figura 3.6b. En esta búsqueda en profundidad sí se expande γ_1 ($\vec{f}(\gamma_1)[2] = 6 \leq 6$) y se identifica como solución del problema, con lo cual su vector de coste $(10, 6)$ se añade al conjunto $SOLUTION$. Dado que no domina a la solución previamente localizada γ_2, de coste $(6, 1)$, ambas se mantienen. La búsqueda se discontinúa en el nodo n_2 al superar la cota fijada ($\vec{f}(n_2)[2] = 7 > 6$).

Aunque no se incluyan las figuras correspondientes, se realizarían más iteraciones, ya que hay nodos (en nuestro caso n_2) discontinuados por superar la cota, con lo cual se fijaría un nuevo umbral con valor 7 para la iteración posterior, y así sucesivamente hasta alcanzar el umbral máximo *MaxThreshold* para el objetivo 2. Estas iteraciones no encuentran nuevas soluciones para este ejemplo concreto, pero son necesarias de modo general para garantizar la admisibilidad del algoritmo, i.e., que encuentre todas las soluciones no dominadas.

Supongamos un espacio de estados con una frontera de Pareto formada por los vectores de coste $\{(5, 8), (6, 6), (8, 5)\}$. En el procesamiento del primer objetivo se localizaría el camino con coste $(5, 8)$, fijando una cota absoluta superior *MaxThreshold* para el objetivo 2 igual a 8. Al procesar el segundo objetivo, se localiza en primer lugar la solución con coste $(8, 5)$. Sin embargo el algoritmo no puede detenerse en esta iteración, porque dejaría de encontrar el óptimo de Pareto $(6, 6)$. Por ello el proceso de búsqueda debe seguir avanzando la cota hasta llegar al valor *MaxThreshold*.

Dado que en el ejemplo anterior todas las soluciones del problema eran óptimos de Pareto, los tests de dominancia no excluyen nodos del conjunto $SOLUTION$. Sin embargo sí es posible que $IDMOA^*$ localice una solución dominada y la agregue a este conjunto, aunque solo sea de modo temporal. En la figura 3.7 se incluye un caso sencillo de un árbol con un nodo raíz cuyos dos sucesores son nodos finales del problema, uno con coste óptimo (γ_2) y otra subóptimo (γ_1). Dado que la componente de coste para el primer objetivo es la misma en ambas soluciones, se encontrarán en la misma iteración. El orden en el que se localizan viene dado por el orden de expansión. Si el árbol se expande de izquierda a derecha, se localiza primero γ_1, que no es un óptimo de Pareto. Por este motivo $IDMOA^*$ precisa realizar tests de dominancia sobre el conjunto $SOLUTION$ cada vez que localiza un nuevo nodo final. Al localizar γ_2, estos tests de dominancia purgarán $\vec{f}(\gamma_1)$ del conjunto.

3.6.3. $MOMA^*0$: $RBFS$ multiobjetivo

El algoritmo $MOMA^*0$ (Dasgupta et al., 1999) se presenta como una adaptación de *RBFS* al caso multiobjetivo. Cuando *RBFS* explora una rama del árbol de búsqueda almacena el coste del mejor de los caminos descartados, para retomar dicho camino en el momento en que el coste del camino actual sobrepase este valor *recordado*. En el caso multiobjetivo, entre todas las ramas discontinuadas puede haber varias que tengan vectores de coste no dominados entre sí, con lo cual sería necesario almacenarlos todos

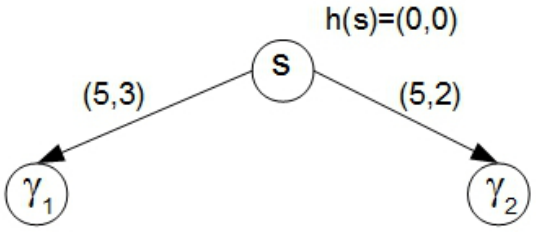


Figura 3.7: Ejemplo de inclusión temporal de soluciones dominadas en IDMOA*

en memoria y establecer un criterio para seleccionar el camino a expandir en caso de *backtracking*.

Dado que, a medida que se profundiza en la exploración del árbol, el número de vectores de coste no dominados crece, es posible que su almacenamiento presente problemas de espacio y procesamiento. Es por ello que el algoritmo $MOMA^*0$, tras inducir un orden total en estos vectores (orden lexicográfico), almacena el menor de todos ellos (menor lexicográfico). El óptimo lexicográfico de un conjunto de vectores, es también un óptimo de Pareto. En la figura 3.8 puede verse un sencillo ejemplo del cálculo que realiza $MOMA^*0$ para la cota superior de cada nodo.

Definición 3.6.1. *Sean dos vectores $\vec{x}, \vec{y} \in \mathbb{R}^q$. La relación de orden total $<_{lex}$ (denotada como relación de orden lexicográfico) se define como sigue:*

$$\vec{x} <_{lex} \vec{y} \Leftrightarrow \exists j, 1 \leq j \leq q \mid x_j < y_j \wedge \forall i, i < j, x_i = y_i \tag{3.8}$$

En este ejemplo, el nodo n_1, seleccionado para expansión, tiene una cota superior, $(4, 4)$, que refleja información de uno de los caminos olvidados, camino disponible bien a través de algún hermano de n_1, bien a través de algún ascendiente de n_1 en el árbol de búsqueda. Una vez que n_1 se selecciona para expansión, sus sucesores $(n_{11}, \ldots, n_{14})$ se ordenan lexicográficamente. Se escoge el mejor lexicográfico (en este caso n_{11}) y se calcula su cota superior como el mejor lexicográfico obtenido de los vectores de coste de sus hermanos y la cota superior del padre. En el ejemplo, esta cota superior es $(3, 5)$. El subárbol de búsqueda de n_{11} se explorará, discontinuando una rama cuando el vector de coste de asociado a dicha rama sea peor lexicográfico que la cota superior establecida.

El *backup* de la información asociada a ramas discontinuadas no tiene por qué restringirse a un único vector, tal y como lo hace $MOMA^*0$. Si hay suficiente memoria disponible, es posible modificar el algoritmo para que almacene más de un vector, lo cual puede reducir la cantidad de reexpansiones que se realizan. Existe una relación inversa entre el número de vectores almacenados y el número de reexpansiones, pero, tal y como se comentó anteriormente, a mayor número de vectores más memoria y tiempo de procesamiento requiere el tratamiento de esta cota multivectorial.

El algoritmo $MOMA^*0$ también hace uso, a través de su función *Minf*, de la técnica de *pathmax* (Chakrabarti et al., 1989) (Dechter y Pearl, 1985) (László Mérõ, 1984). Esta

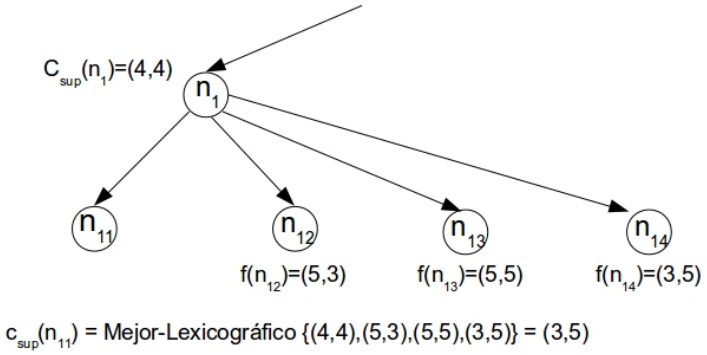


Figura 3.8: Cálculo de la cota superior en $MOMA^*0$

técnica se usa fundamentalmente en presencia de heurísticos admisibles no monótonos y permite propagar información de coste de padres a hijos dentro del árbol de búsqueda, con el fin de reducir la cantidad de reexpansiones.

Para mejorar el rendimiento del algoritmo también se obvian las comparaciones contra el primer objetivo en los tests de dominancia. Dado que la expansión viene guiada por un orden lexicográfico, es trivial que, una vez localizada la primera solución, la primera componente del vector de coste de cualquier nodo a expandir será mayor o igual que la del vector de coste de dicha solución, con lo cual esta componente se descarta de los tests de dominancia.

Aunque en (Dasgupta et al., 1999) se hace una descripción más o menos detallada del algoritmo $MOMA^*0$, algunas erratas detectadas en el algoritmo, así como la falta de claridad en algunos aspectos del mismo no hacen posible determinar su funcionamiento exacto y su codificación. Dado que en el Capítulo 5 se incluyen dos nuevas extensiones de *RBFS* al caso multiobjetivo, a efectos de comparativa se ha considerado conveniente incluír una versión simplificada de $MOMA^*0$. Esta versión no incluye, por ejemplo, el uso de *pathmax*, aunque esto no distorsiona los resultados de las pruebas descritas en el Capítulo 8, al usarse únicamente heurísticos admisibles y monótonos. Además únicamente contempla un vector heurístico por nodo (y por lo tanto una única estimación de coste por nodo en el caso de árboles), y está limitada a dos objetivos. En la tabla 3.2 se muestra el código de esta versión simplificada.

3.6.3.1. Ejemplo de $MOMA^*0$

En esta sección presentaremos un ejemplo detallado del funcionamiento de $MOMA^*0$. El problema resuelto será el mismo que se ha empleado para ilustrar el funcionamiento de $IDMOA^*$ (ver figura 3.5a). La llamada inicial proporciona como cota inferior para el nodo raíz s su vector heurístico, mientras que como cota superior tenemos el vector (∞, ∞), para permitir la exploración de todos los nodos hijos. Recordemos que la cota inferior de un nodo representa información de expansiones previas del mismo, permitiendo actualizar de modo más eficiente los vectores de coste de sus hijos. Asi-

```
Initial call: MOMA*0 ( s, h⃗(s), ∞⃗, ∅)
MOMA*0 (n, c⃗_inf, c_sup, SOLUTION )

IF (f⃗(n) ⊀_lex c⃗_sup)
     return (f⃗(n), SOLUTION)
IF (∃(γ, f⃗(γ)) ∈ SOLUTION | f⃗(γ) ≺ f⃗(n))
     return (∞⃗, SOLUTION)
IF (n ∈ Γ)
     return (∞⃗, SOLUTION ∪ (n, f⃗(n)))
IF (Successors(n) = ∅)
     return (∞⃗, SOLUTION)
For each n_i ∈ Successors(n)
     SucSet(i) ← Minf (f⃗(n_i), c⃗_inf)
Relocate SucSet nodes in lexicographic order
IF (SucSet(2) = ∅)
     SucSet(2) ← ∞⃗
LOOP
     IF (c⃗_sup <_lex SucSet(1))
          return (SucSet(1), SOLUTION)
     IF (SucSet(1) = ∞⃗)
          return (∞⃗, SOLUTION)
     (SucSet(1), SOLUTION) ← MOMA*0 (n_i, SucSet(1), bestLex(c⃗_sup, SucSet(2)))
     Relocate SucSet nodes in lexicographic order
END LOOP


Minf (cost⃗, bound⃗)

FOR i = 1 TO ♯objectives
     IF (bound⃗[i] < cost⃗[i]) return cost⃗
     ELSE cost⃗[i] ← bound⃗[i]
     return cost⃗
```

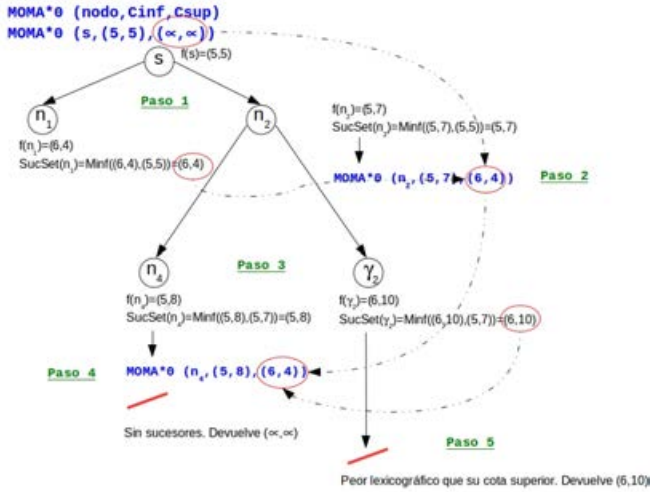
Tabla 3.2: Adaptación del algoritmo $MOMA^*0$ (Dasgupta et al., 1999) considerando un único vector heurístico y dos objetivos.

mismo, la cota superior de un nodo representa información de coste de caminos que han sido abandonados, probablemente de modo temporal, para expandir la rama actual. En caso de empeorar (lexicográficamente) el coste por esta rama, se realizaría el correspondiente retroceso.

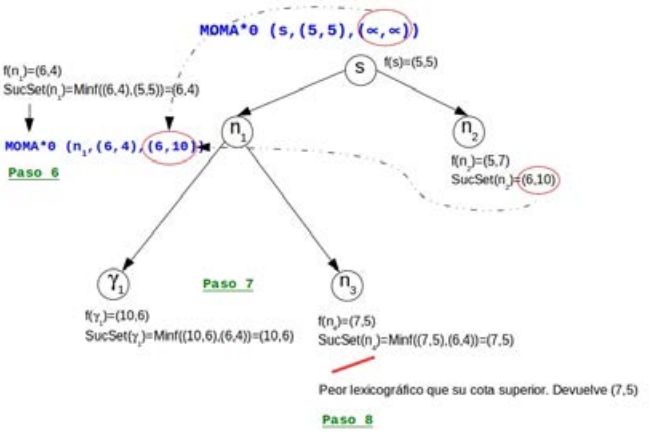
La figura 3.9a muestra las primeras etapas de procesamiento de $MOMA^*0$. Al expandir el nodo s (figura 3.9a, Paso 1) se generan los nodos n_1 y n_2, con sus correspondientes vectores de coste. Dado que no ha habido expansiones previas, la cota inferior del nodo s no modifica dichos vectores de coste. Al ser el vector de coste de n_2 mejor lexicográfico que el de n_1 ($(5,7) <_{lex} (6,4)$) y no ser peor lexicográfico que la cota superior de s ($(\infty,\infty) \not<_{lex} (5,7)$), se procede a explorar el nodo n_2 (figura 3.9a, Paso 2), usando como cota superior el mejor coste lexicográfico de los caminos *olvidados*. En este caso, el camino que va a través de n_1, con coste $(6,4)$. Al expandir n_2 (figura 3.9a, Paso 3) se generan los nodos n_4 y γ_2. Al igual que en la expansión anterior, al no disponer de exploraciones previas, el coste de estos nodos no se actualiza. Se selecciona n_4 para expansión, al ser mejor lexicográfico que γ_2 ($(5,8) <_{lex} (6,10)$) y no ser peor lexicográfico que la cota superior de n_2 ($(6,4) \not<_{lex} (5,8)$). Al expandir n_4 (figura 3.9a, Paso 4) y no tener sucesores, la llamada devuelve el vector (∞,∞), indicando que por dicha rama no hay caminos de interés. El nodo γ_2 no llega a expandirse (figura 3.9a, Paso 5), al ser su vector de coste peor lexicográfico que su cota superior ($(6,4) <_{lex} (6,10)$).

Dado que los costes actualizados de los sucesores de n_2 son peores lexicográficos que la cota superior de dicho nodo, se reconsidera ahora la expansión del nodo n_1 (figura 3.9b), actualizando convenientemente el coste de n_2 con el coste mejor lexicográfico propagado desde sus sucesores, en este caso el vector $(6,10)$. Para la exploración de n_1 (figura 3.9b, Paso 6) se usa como cota superior el coste del mejor camino olvidado ($(6,10)$, a través del nodo n_2). La expansión de n_2 (figura 3.9b, Paso 7) genera los sucesores γ_1 y n_3, con vectores de coste $(10,6)$ y $(7,5)$ respectivamente. Estos vectores no sufren actualización a través de la cota inferior de n_1, al no haber sido estos caminos expandidos previamente. Dado que el coste de n_3 es mejor lexicográfico que el de γ_1 ($(7,5) <_{lex} (10,6)$), se procesa en primer lugar (figura 3.9b, Paso 8). Sin embargo, al ser peor lexicográfico que la cota superior ($(6,10) <_{lex} (7,5)$), la búsqueda se discontinúa.

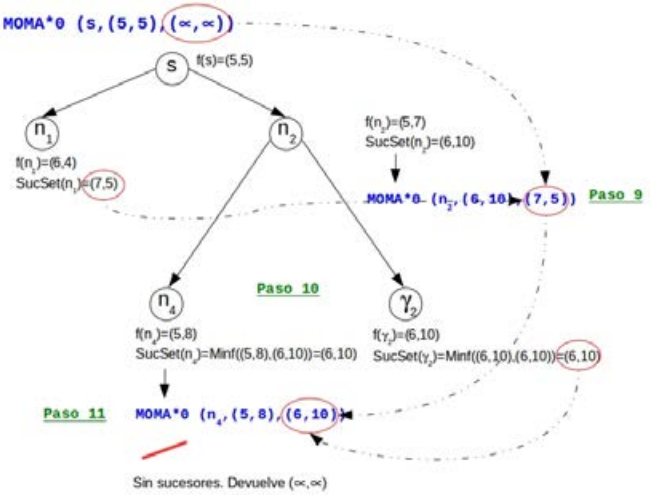
El coste del nodo n_1 se actualiza al mejor coste lexicográfico de sus sucesores (figura 3.9c), en este caso el vector $(7,5)$. En este punto, los nodos n_1 y n_2 ya han sido expandidos y reconsiderados, y sus vectores de coste actualizados en base a dichas expansiones con los valores $(7,5)$ y $(6,10)$ respectivamente. Al ser n_2 mejor lexicográfico ($(6,10) <_{lex} (7,5)$), se procede a su reexpansión (figura 3.9c, Paso 9), usando como cota superior el mejor coste lexicográfico de los caminos olvidados (en nuestro caso, coste $(7,5)$ a través del nodo n_1). La expansión del nodo n_2 (figura 3.9c, Paso 10) genera de nuevo los nodos n_4 y γ_2, los cuales sí ven actualizados sus costes con la cota inferior de n_2, $(6,10)$, fruto de la expansión previa de dicho nodo. En este caso ambos nodos tienen el mismo vector de coste actualizado, $(6,10)$, con lo cual su orden de expansión es diferente. Suponemos que en primer lugar se explora n_4 (figura 3.9c, Paso 11), dado



(a) Resolución con $MOMA^*0$ (1)

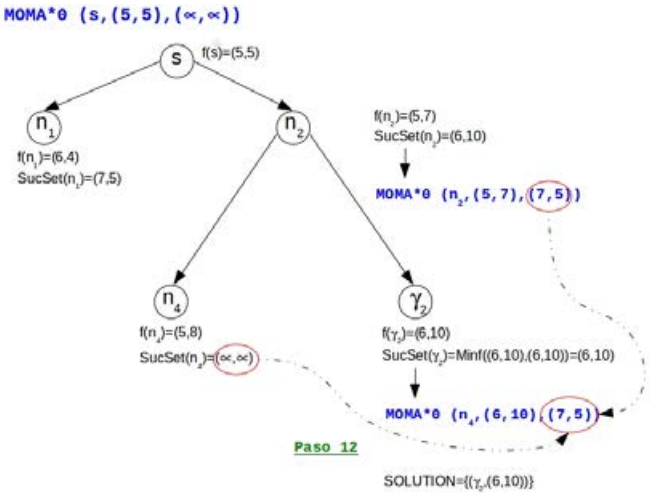


(b) Resolución con $MOMA^*0$ (2)

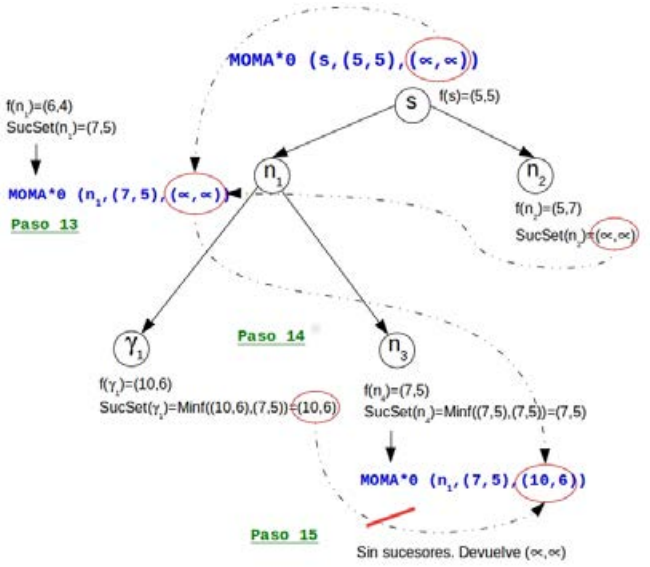


(c) Resolución con $MOMA^*0$ (3)

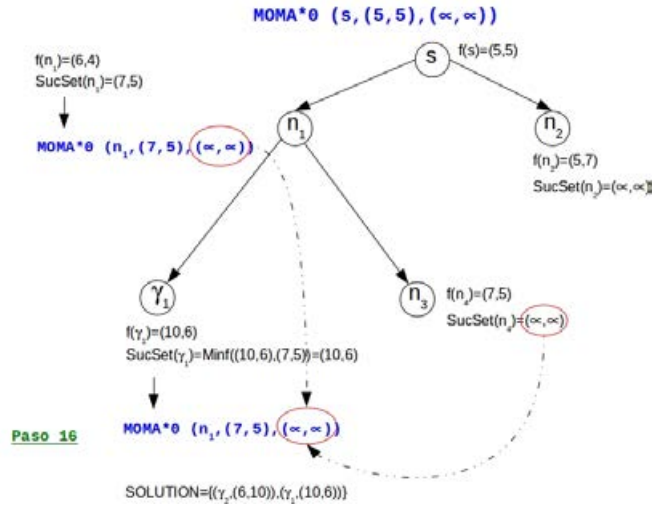
Figura 3.9: Ejemplo de $MOMA^*0$ (a)



(a) Resolución con $MOMA^*0$ (4)



(b) Resolución con $MOMA^*0$ (5)



(c) Resolución con $MOMA^*0$ (6)

Figura 3.10: Ejemplo de $MOMA^*0$ (b)

que su coste no es peor lexicográfico que la cota superior de n_2 ($(7,5) \not<_{lex} (6,10)$). Sin embargo, al no tener sucesores, se realiza el correspondiente retroceso, devolviendo un vector de coste (∞,∞) al igual que en su expansión previa.

El nodo γ_2 también se expande (figura 3.10a, Paso 12), al no ser tampoco peor lexicográfico que la cota superior de n_2 ($(7,5) \not<_{lex} (6,10)$). Al ser un nodo final, se incorpora al conjunto de soluciones óptimas, devolviendo la llamada (∞,∞). En este punto, los dos caminos a través de n_2 se han discontinuado de modo definitivo, uno al no disponer de más nodos para expansión y otro por conducir a una solución óptima.

Se actualiza el coste del nodo n_2 a (∞,∞) (figura 3.10b) y se selecciona para exploración el nodo n_1 (figura 3.10b, Paso 13), al ser mejor lexicográfico que n_2 ($(7,5) <_{lex} (\infty,\infty)$). Al generar los sucesores γ_1 y n_3 (figura 3.10b, Paso 14), sus costes no sufren actualización, ya que la expansión previa fue discontinuada en ambos nodos al ser peores lexicográficos que la cota superior informada por su padre n_1. Se selecciona para expansión n_3 (figura 3.10b, Paso 15), al ser su vector de coste mejor lexicográfico que el de γ_1 ($(7,5) <_{lex} (10,6)$) y no ser peor lexicográfico que la cota superior de n_1 ($(\infty,\infty) \not<_{lex} (7,5)$). Al no tener sucesores, la búsqueda se discontinúa, devolviendo el vector (∞,∞).

Por último se selecciona para expansión el nodo γ_1, (figura 3.10c, Paso 16), al ser su vector de coste mejor lexicográfico que el coste actualizado de n_3 ($(10,6) <_{lex} (\infty,\infty)$) y no ser peor lexicográfico que la cota superior de n_1 ($(\infty,\infty) \not<_{lex} (10,6)$). Al ser γ_1 un nodo meta, se incluye en el conjunto de soluciones óptimas, propagando en el retroceso el vector (∞,∞). Al haber informado todas las ramas del árbol un vector de coste (∞,∞), el algoritmo finaliza, devolviendo la frontera de Pareto.

3.7. Clasificación de algoritmos multiobjetivo exactos *depth-first*

Una vez presentadas las principales técnicas empleadas en la búsqueda *depth-first*, así como los algoritmos más representativos, estableceremos una categorización de este tipo de algoritmos que permitirá incluír tanto los trabajos previos descritos en este Capítulo como las nuevas propuestas que presentaremos en esta tesis, así como la relación entre ambos.

La clasificación se realiza fundamentalmente en base a dos criterios:

- Naturaleza de la cota
- Estrategia de búsqueda

3.7.1. Naturaleza de la cota

La naturaleza de la cota, entendida como cota superior de búsqueda, es un elemento crítico en cualquier búsqueda *depth-first*, pues es la que va a determinar la cantidad de

reexpansiones que se realicen en el árbol de búsqueda. Cualquier nodo cuyo coste exceda la cota será discontinuado, pero no descartado, con lo cual podrá ser expandido en iteraciones posteriores. Una cota multivectorial va a ser mucho más flexible a la hora de permitir que un nodo se expanda durante una iteración. El vector de coste del nodo se comparará contra los vectores de la cota, lo cual es una operación computacionalmente costosa. Para expandir un nodo es suficiente con que su coste no sea dominado por alguno de estos vectores. Esto conlleva que en cada iteración del algoritmo *depth-first* el árbol de búsqueda avance o profundice más. Sin embargo esta cota multivectorial incrementará el número de comparaciones que es necesario realizar con cada nodo del árbol de búsqueda para determinar si debe ser expandido o no.

El caso opuesto viene dado por una cota formada por un único vector. En este caso, el número de comparaciones se reduce al mínimo cuando procesamos un nodo. Sin embargo este tipo de cota provoca avances más lentos en la exploración del árbol de búsqueda, mayor número de iteraciones y, por lo tanto, mayor número de reexpansiones.

Además de la dimensión de la cota, es decir, la cantidad de vectores que la forman, también tiene importancia la calidad de la misma. Como se verá en el Capítulo 4, generar una cota que contenga información de la frontera (nodos discontinuados) del árbol de búsqueda expandido en la iteración anterior hace que la exploración avance más rápidamente. Sin embargo volvemos a encontrarnos con el problema de su procesamiento, dado que la cota crece de tamaño a medida que profundizamos en el árbol.

Existen mecanismos como el orden lexicográfico (ya analizado en $MOMA^*0$) o el *Punto Ideal* (que estudiaremos en el capítulo 4), que nos permitirán, a partir de una cota multivectorial obtener una cota formada por un solo vector. Esto permite simplificar los tests de dominancia, pero a cambio, la cota tendrá en general menor calidad que el conjunto de vectores a partir del cual se ha calculado, lo que se traduce en mayor número de reexpansiones.

3.7.2. Estrategia de búsqueda

El otro aspecto a tener en cuenta en la categorización es la estrategia de búsqueda. De modo similar a las secciones previas, distinguimos dos grandes tipos de enfoques en la búsqueda *depth-first*: la profundización iterativa y la búsqueda recursiva *best-first*.

La profundización iterativa, tal y como ya se ha comentado, fija una cota superior que limita la expansión del árbol de búsqueda durante la iteración actual. Una vez que en todas las ramas ya se ha superado dicha cota, se recalcula la cota para la siguiente iteración en base a la información de los nodos hoja que han sido discontinuados.

La búsqueda recursiva *best-first*, basada en *RBFS*, establece una cota que representa el mejor de los caminos discontinuados. Una rama se expande mientras su coste no sea peor que el de dicho camino, en cuyo caso se reconsidera la rama a explorar.

A efectos de completitud, se ha incluído un tercer grupo de algoritmos que sigue una estrategia secuencial o por fases. En este grupo se incluye, por ejemplo, $IDMOA^*$, que procesa secuencialmente los diferentes objetivos del problema utilizando profun-

Cota \ Búsqueda	*ID*	*RBFS*	Secuencial - Por fases
Orden lexicográfico	$LEXIDMOA^*$	$MOMA^*0$	$IDMOA^*$
Orden de Pareto	$PIDMOA^*$	*Pareto-MO-RBFS*	IDA^*-*MO-BnB*
Punto Ideal	$IPID^*$	*IP-MO-RBFS*	*RBFS-MO-BnB*

Tabla 3.3: Categorización de algoritmos multiobjetivo exactos *depth-first*

dización iterativa. Asimismo se incluyen diferentes variantes de *DF-BnB* que incluyen una fase de localización de cota inicial y luego una fase de refinamiento de las soluciones localizadas.

3.7.3. Clasificación de los algoritmos *depth-first*

Una vez fijados los criterios de clasificación, podemos establecer una categorización de los posibles algoritmos multiobjetivo exactos de tipo *depth-first*, la cual viene reflejada en la tabla 3.3. En dicha tabla se han indicado algunos de los algoritmos vistos en este Capítulo, así como los desarrollados como fruto de este trabajo. Esta tabla nos servirá para presentar de forma ordenada las contribuciones de esta tesis, así como analizar sus resultados. La columna asociada a profundización iterativa incluye los tres algoritmos que se analizarán en el Capítulo 4. La principal diferencia entre ellos es el modo de calcular la cota de expansión.

En la segunda columna de algoritmos, los basados en *RBFS*, se incluye $MOMA^*0$, el cual utiliza una cota de expansión calculada como el mejor lexicográfico de los nodos discontinuados en la iteración previa. Los otros dos algoritmos, *Pareto-MO-RBFS* e *IP-MO-RBFS*, son aportaciones de esta tesis y se analizarán en profundidad en el Capítulo 5.

En la última columna se incluyen, sin referencia alguna al tipo de cota empleada, un grupo de algoritmos *depth-first* en los cuales el cálculo de esta cota no se ajusta a ninguna de las tres opciones ya comentadas (orden lexicográfico, orden de Pareto y Punto Ideal). En el caso de $IDMOA^*$, aunque este algoritmo está basado en profundización iterativa, no es una extensión directa de ésta al caso multiobjetivo. $IDMOA^*$ calcula una cota por cada iteración, pero, tal y como se ha visto en la sección 3.6.2, esta cota es de tipo escalar, al procesar $IDMOA^*$ los objetivos de modo independiente, motivo por el cual en esta clasificación no se ha considerado incluirlo en el primer grupo. Se trata en realidad de una aplicación de q búsquedas IDA^* secuenciales, una por cada objetivo del problema.

El resto de algoritmos incluídos en este tercer grupo son las variantes multiobjetivo de *DF-BnB*, las cuales emplean el conjunto de soluciones ya encontradas, C^*, como cota superior del árbol de búsqueda. Como veremos en el Capítulo 6, *DF-BnB* se combina con una fase inicial de búsqueda de la primera solución, la cual debe realizarse

mediante un algoritmo completo. Una vez obtenida dicha solución, se suministra como cota superior inicial a *DF-BnB* multiobjetivo para que refine el conjunto C^* hasta obtener todos los óptimos de Pareto. Esto permite aprovechar la potencia de *DF-BnB* incluso en grafos infinitos.

La clasificación vista en esta sección servirá asimismo de hilo conductor para el análisis de rendimiento. En el Capítulo 8 se realiza una primera batería de pruebas para localizar el mejor candidato de cada categoría y finalmente, con los mejores algoritmos de cada estrategia de búsqueda se realiza un nuevo conjunto de pruebas que determinará el mejor algoritmo *depth-first* para búsqueda multiobjetivo.

3.8. Consideraciones acerca del rendimiento de los algoritmos multiobjetivo

El rendimiento de los algoritmos multiobjetivo no es comparable bajo ninguna circunstancia con algoritmos que trabajan con un único objetivo. La mayoría de éstos últimos se limitan a devolver la primera solución que encuentran, la cual es óptima en el caso de algoritmos como A^* o IDA^*, es decir, pudiendo haber en el espacio de estados varios nodos finales con el mismo coste óptimo, los algoritmos únicamente devuelven uno de ellos. Por el contrario, los algoritmos multiobjetivo exactos devuelven todos y cada uno de los óptimos de Pareto del problema. En este sentido, los algoritmos multiobjetivo no son generalizaciones directas de los escalares, sino de versiones de éstos modificadas para encontrar todas las soluciones óptimas.

Otro aspecto claramente diferente entre ambos tipos de algoritmos es el esfuerzo computacional requerido para el procesamiento de un nodo. En la sección 2.8 veíamos que este esfuerzo variaba mucho de un algoritmo *best-first*, con tiempos de procesamiento generalmente altos, a un algoritmo *depth-first*, donde este tiempo es prácticamente constante. En el caso multiobjetivo, si atendemos a los resultados de los análisis *best-first*, las diferencias son mayores, debido a que muchas de las operaciones involucradas implican la comparación de un vector con un conjunto de vectores de cardinalidad variable. De este sobrecoste no se librarán tampoco los algoritmos *depth-first* multiobjetivo. En el Capítulo 8 veremos como las comparaciones de dominancia tienen una gran repercusión en el rendimiento de los diferentes algoritmos.

3.9. Resumen

En este Capítulo se han presentado de modo detallado algunos algoritmos multiobjetivo de tipo *depth-first*, los cuales generalizan diferentes paradigmas de la búsqueda escalar. Sin embargo, como ya hemos visto para los algoritmos *best-first*, esta generalización no tiene porque ser única. Por ejemplo, del algoritmo A^* existen por lo menos tres variantes multiobjetivo: MOA^* (en la cual se realiza expansión de nodos, y por tanto de todos los caminos óptimos asociados a cada nodo), el algoritmo presentado

en (Tung y Chew, 1992) (que expande caminos en lugar de nodos) y $NAMOA^*$ (que también expande caminos y mejora las operaciones de poda y filtrado). Es un objetivo y aportación fundamental de esta tesis explorar nuevas generalizaciones en el ámbito depth-first y analizar cuales son las mejores tanto en grafos finitos como infinitos. Hemos de destacar igualmente la falta de una evaluación sistemática del rendimiento de los algoritmos ya presentados en la literatura ($IDMOA^*$ y $MOMA^*0$), así como de una comparativa entre los mismos o con *MO-DF-BnB*.

En los siguientes Capítulos se mostrarán los problemas de diseño que presentan algunos de los algoritmos ya existentes, con repercusiones en su rendimiento, y como se han abordado estos problemas mediante nuevas generalizaciones multiobjetivo. En el Capítulo 4 presentaremos tres nuevas variantes de profundización iterativa: $LEXIDMOA^*$, $PIDMOA^*$ e $IPID^*$; el Capítulo 5 presenta dos nuevas generalizaciones multiobjetivo de *RBFS*: *Pareto-MO-RBFS* e $IP\text{-}MO\text{-}RBFS$; en el Capítulo 6 veremos una versión simplificada de *DF-BnB* multiobjetivo que permite trabajar con espacios de estados infinitos. Las aportaciones se completarán con demostraciones formales de las propiedades de los algoritmos y una evaluación de los mismos en los Capítulos 7 y 8 respectivamente.

Parte II

Diseño de algoritmos, análisis formal y evaluación

En esta segunda parte se incluyen todas las contribuciones originales de esta tesis: diseño, análisis formal y evaluación empírica de diferentes algoritmos exactos multiobjetivo de tipo *depth-first*. Por un lado se presentan nuevos algoritmos multiobjetivo que mejoran el rendimiento de otros ya existentes. Concretamente se definen los algoritmos $LEXIDMOA^*$, $PIDMOA^*$ e $IPID^*$, basados en el concepto de profundización iterativa y los algoritmos *Pareto-MO-RBFS* e *IP-MO-RBFS*, basados en la búsqueda recursiva *best-first*. Asimismo se propone una adaptación del algoritmo *DF-BnB* multiobjetivo con el fin de que sea completo al tratar con espacios de estados infinitos. Para todos los algoritmos nuevos se presentan pruebas formales de completitud y admisibilidad. Por otro lado en este bloque se realiza también una evaluación empírica detallada, fundamentalmente sobre espacios de estados infinitos, del rendimiento de todos los algoritmos presentados.

Más concretamente:

- El capítulo 4 presenta tres nuevos algoritmos ($LEXIDMOA^*$, $PIDMOA^*$ e $IPID^*$) que extienden la profundización iterativa al caso multiobjetivo. Estos algoritmos varían el tamaño y la *calidad* de la cota de expansión para intentar mejorar el rendimiento de otras propuestas previas.

- El capítulo 5 presenta dos nuevos algoritmos (*Pareto-MO-RBFS* e *IP-MO-RBFS*), que son una generalización de *RBFS* al caso multiobjetivo. Al igual que en el capítulo anterior, el factor diferencial de estos algoritmos es el tratamiento que dan a la cota de expansión empleada durante las diferentes iteraciones del proceso de búsqueda.

- En el capítulo 6 se presentan diferentes variantes del algoritmo *DF-BnB* multiobjetivo que permiten su uso en espacios de estados infinitos. Básicamente se

emplean diferentes algoritmos completos (incluídos algunos de los diseñados en capítulos anteriores) para proporcionar una cota inicial de búsqueda, evitando la incompletitud de *DF-BnB*.

- El capítulo 7 incluye un análisis formal de la completitud y optimalidad de los nuevos algoritmos aportados en este trabajo.
- En el capítulo 8 se realiza una evaluación detallada del rendimiento de los diferentes algoritmos multiobjetivo exactos de tipo *depth-first*, tanto previos como aportados en esta tesis. El análisis se realiza teniendo en cuenta los mejores candidatos de cada una de las tres grandes categorías: profundización iterativa, *RBFS* y secuencial. Previamente se han analizado y seleccionado los parámetros de rendimiento a tener en cuenta para dicha evaluación, habida cuenta de la dificultad especial del caso multiobjetivo por la naturaleza vectorial de los costes.

Capítulo 4

Nuevos enfoques multiobjetivo basados en profundización iterativa

En el Capítulo 2 se referencia una de las clasificaciones más generales para los algoritmos de búsqueda, aquella que tiene en cuenta los requisitos de memoria de los algoritmos a la hora de recorrer el árbol de búsqueda. Esta clasificación incluye dos categorías principales: los algoritmos de tipo *best-first* y los algoritmos de tipo *depth-first*. Esta tesis profundiza en el análisis y diseño de algoritmos del segundo grupo generalizados al caso multiobjetivo, siendo su principal ventaja el uso de unos recursos de memoria lineales con la profundidad del problema que se está tratando para el proceso de búsqueda (no así para guardar la solución). Nuestro esfuerzo en este Capítulo se centrará más concretamente en los algoritmos de profundización iterativa.

Una propuesta previa en este sentido es $IDMOA^*$ (Harikumar y Kumar, 1996), una aplicación del paradigma de profundización iterativa para el problema multiobjetivo, pero que emplea cotas o umbrales de expansión escalares. Este algoritmo no es realmente una generalización multiobjetivo de IDA^*, ya que aplica la profundización iterativa sobre información escalar de coste relativa a cada uno de los objetivos del problema, siendo estos objetivos procesados individual y secuencialmente. $IDMOA^*$, pese a ser completo y óptimo, presenta en su diseño algunas deficiencias en cuanto a rendimiento derivadas principalmente de este procesamiento secuencial de los objetivos considerados y de la localización de soluciones subóptimas durante el proceso de búsqueda.

En este capítulo presentaremos diferentes variantes multiobjetivo de la profundización iterativa, las cuales tratan de solventar los problemas anteriormente mencionados.

La estructura del capítulo es la siguiente. En la sección 4.1 analizaremos con detalle algunos aspectos relacionados con el rendimiento de $IDMOA^*$, identificando sus posibles mejoras. La sección 4.2 presenta un nuevo algoritmo, $PIDMOA^*$ (Coego et al., 2009), que representa una generalización directa de IDA^* al caso multiobjetivo,

mediante el uso de umbrales formados por conjuntos de vectores, para tratar de acelerar el proceso de búsqueda. El uso de este umbral multivectorial incrementa el número de tests de dominancia a realizar durante la generación del árbol de búsqueda. Es por ello que las secciones 4.3 y 4.4 presentan los algoritmos $LEXIDMOA^*$ e $IPID^*$ respectivamente, variantes multiobjetivo con umbrales formados por un único vector que tratan de realizar la búsqueda de modo más eficiente que $PIDMOA^*$, reduciendo los tests de dominancia, ya que estos tests, tal y como se verá en el Capítulo 8, son un factor crítico en el rendimiento de la profundización iterativa multiobjetivo.

4.1. Análisis de $IDMOA^*$

La sección 3.6.2 incluye una descripción detallada del algoritmo $IDMOA^*$. Como vimos, este algoritmo extiende en cierto modo el concepto de búsqueda por profundización iterativa al caso multiobjetivo. Procesa secuencialmente cada uno de los objetivos considerados, iterando por cada uno de ellos y buscando en cada una de estas iteraciones las soluciones óptimas para el objetivo procesado. Este procesamiento secuencial propicia que los tests de cota realizados por $IDMOA^*$ sean escalares, ya que se limitan a comprobar los valores del objetivo considerado en la iteración actual del algoritmo. Esto implica que los tests realizados contra la cota se realizan de modo muy eficiente, ya que esta cota no es un vector ó conjunto de vectores, sino un valor escalar. $IDMOA^*$ no es, por tanto, una generalización propiamente dicha de IDA^* al caso multiobjetivo, sino una aplicación secuencial de IDA^* a cada uno de los diferentes objetivos considerados.

Una posible fuente de ineficiencia de $IDMOA^*$ consiste precisamente en este tratamiento secuencial de los objetivos. Durante la profundización progresiva i del algoritmo se procesa únicamente la componente i del vector de coste de cada uno de los nodos procesados. Esta componente de coste se compara contra la cota escalar que mantiene el algoritmo. Esto hace que durante esta profundización se obvien todos los valores asociados al resto de componentes de los vectores de coste. Si el problema tuviera n objetivos, en la iteración i no se procesa la información relativa a las componentes $\{c_{i+1}, c_{i+2}, \ldots, c_n\}$. En otras palabras, en cada profundización iterativa el algoritmo va aprendiendo cómo de bueno o malo es un nodo para un objetivo determinado, prescindiendo de la información asociada a los restantes objetivos, la cual será procesada en las iteraciones correspondientes. Por este motivo, el número de reexpansiones de nodos realizadas por $IDMOA^*$ puede ser muy alto.

Por otro lado, aunque los tests realizados contra la cota son escalares, y por lo tanto muy eficientes, $IDMOA^*$, precisamente debido al procesamiento secuencial de objetivos ya mencionado, es susceptible de incluír, aunque sólo de modo temporal, soluciones subóptimas en el conjunto de soluciones, tal y como se pudo ver en la figura 3.7. Cada vez que $IDMOA^*$ localiza una nueva solución durante el proceso de búsqueda, su vector de coste debe ser comparado con el de los vectores de coste del conjunto de soluciones ya encontradas para determinar si está dominada (en cuyo caso se descarta)

o domina a algunos de los vectores de este conjunto (en cuyo caso estas soluciones dominadas deben purgarse del conjunto solución). El primer caso, localización y descarte de una solución subóptima, es común a los algoritmos de búsqueda multiobjetivo *best-first*, donde recibe el nombre de *filtrado* (Mandow y Pérez de la Cruz, 2005), y a los algoritmos de tipo *depth-first*, donde recibe el nombre de *bounding*; el segundo caso (localización de una solución, óptima o no, que mejora a una solución ya localizada) genera tests de dominancia adicionales de tipo vectorial.

4.2. El algoritmo $PIDMOA^*$

La descripción del algoritmo $IDMOA^*$ (ver sección 3.6.2) y el análisis de la sección 4.1 muestran que aunque $IDMOA^*$ aplica con éxito la técnica de la profundización iterativa a problemas de búsqueda multiobjetivo:

- No es una generalización directa de IDA^* al caso multiobjetivo, ya que lo que propone es una secuencia de profundizaciones iterativas, cada una de las cuales se centra en un objetivo diferente.
- Puede explorar nodos y realizar tests de dominancia innecesarios, lo que sugiere que su rendimiento podría ser susceptible de mejora.

En esta tesis proponemos un algoritmo alternativo que traslade de una forma más pura los principios de IDA^* al caso multiobjetivo. Más concretamente:

- Sustituimos el umbral escalar por un umbral formado por un conjunto de vectores.
- Realizamos una búsqueda que no considerará nunca nodos que no puedan conducir a soluciones óptimas.
- Por tanto, todas las soluciones encontradas serán óptimos de Pareto, lo que permitiría detener el algoritmo en cualquier momento con una garantía sobre las soluciones encontradas.

El algoritmo $PIDMOA^*$ (*Pareto-Front Iterative Deepening Multi-Objective A**) (Coego et al., 2009), a diferencia de $IDMOA^*$, procesa todos los objetivos simultáneamente, prescindiendo del enfoque escalar, lo cual hace que la exploración del árbol sea más eficiente en cuanto a cantidad de nodos expandidos por iteración. Esto se consigue fundamentalmente cambiando el umbral escalar de cada iteración por un conjunto umbral que contiene una lista de vectores no dominados entre sí. En cada iteración se van expandiendo caminos, discontinuando la búsqueda en un nodo de un camino cuando su vector de estimación de coste está dominado por alguna de las soluciones ya encontradas o por alguno de los vectores de coste del conjunto umbral. Mientras $IDMOA^*$ guía la búsqueda en base a la calidad del objetivo considerado en cada

```
PIDMOA* (G, s, Γ )
SOLUTION ← ∅; Threshold = nodomset(H(s))
WHILE Threshold ≠ ∅
     Threshold ← DFS (s, Threshold)
return (SOLUTION)


DFS (n, Threshold_par)
Nodomvectorsf ← {f⃗ ∈ F(n) | (∄(γ, c⃗*) ∈ SOLUTION | c⃗* ⪯ f⃗)}
IF (Nodomvectorsf = ∅) THEN return ∅;
Domvectorsf ← {f⃗ ∈ Nodomvectorsf | (∄t⃗ ∈ Threshold_par | t⃗ ≺ f⃗)}
IF (Domvectorsf =  ∅) THEN return Nodomvectorsf;
IF (n ∈ Γ ) THEN
     SOLUTION ← SOLUTION ∪ {(n, f⃗(n))}
ELSE
     Threshold_dfs ← ∅
     Sucessors ← expand_node (n)
     FOR each suc in Sucessors DO
          Threshold_dfs ← Threshold_dfs ∪ DFS(suc, Threshold_par)
          delete dominated vectors within Threshold_dfs;
     return (Threshold_dfs)
```

Tabla 4.1: Algoritmo $PIDMOA^*$

momento de la búsqueda, $PIDMOA^*$ analiza el vector de coste completo, lo cual le permite expandir la búsqueda por ramas prometedoras para cualquiera de los objetivos del problema. Aunque, como veremos más adelante, esto conlleva la realización de tests de dominancia vectoriales, computacionalmente más costosos que los tests escalares de $IDMOA^*$, el resultado final será el de un número de iteraciones y nodos expandidos considerablemente menor.

4.2.1. Descripción del algoritmo

En la Tabla 4.1 se muestra el pseudocódigo del algoritmo $PIDMOA^*$. El algoritmo recibe como argumentos un grafo G, un nodo inicial s y un conjunto de nodos finales Γ. Asimismo, mantiene dos conjuntos básicos de trabajo: *SOLUTION* y *Threshold*. El conjunto *SOLUTION* contiene pares (nodo, vector de coste) de todas las soluciones encontradas hasta el momento. Dado que se puede probar que $PIDMOA^*$ sólo localiza soluciones óptimas (ver lemas 7.3.4 y 7.3.5 más adelante), este conjunto únicamente contendrá óptimos de Pareto. El conjunto *SOLUTION* se inicializa a $\emptyset$, mientras que *Threshold* se inicializa al conjunto de vectores heurísticos no dominados entre sí del nodo inicial (la función *nodomset* devuelve los vectores no dominados del conjunto recibido como parámetro).

El conjunto *Threshold* marca la cota de coste para las soluciones que debe buscar

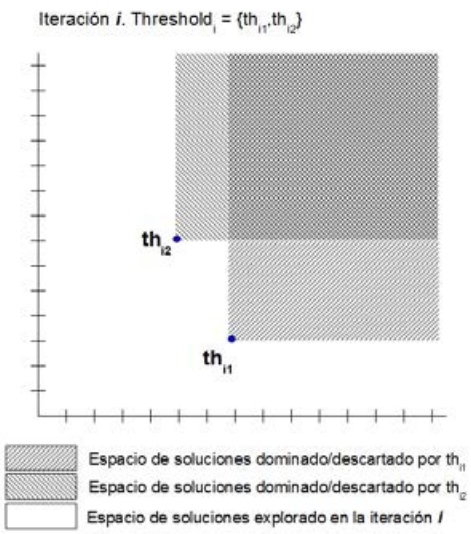


Figura 4.1: Espacio de costes explorado por $PIDMOA^*$ durante una iteración.

el algoritmo en la iteración actual, es decir, acota el espacio de costes en el cual vamos a intentar localizar soluciones en cada una de las iteraciones. Cada iteración se define por una actualización de este conjunto con nuevos vectores de coste asociados a los caminos que todavía son susceptibles de contener soluciones no dominadas.

En la figura 4.1 se representa gráficamente, para un problema con dos objetivos, el espacio de costes que $PIDMOA^*$ descartaría en la búsqueda durante una iteración, así como el espacio de costes que sí exploraría, en base a la información contenida en el conjunto *Threshold*, el cual contiene dos vectores. Cada uno de los dos vectores, $\vec{th}_{i1}$ y $\vec{th}_{i2}$, sirve como cota superior para la búsqueda durante la iteración. Cualquier camino que exceda alguna de estas dos cotas se discontinúa, entendido como exceso el que el vector de coste acumulado en ese camino esté dominado estrictamente por alguno de los vectores umbral. El área que no está dominada por ninguno de los dos vectores es el ámbito de búsqueda de la iteración actual. Nótese que el espacio de soluciones descartado puede ser más amplio si ya se han localizado algunas soluciones, pues también se descartaría en la búsqueda el área dominada por cada una de estas soluciones.

El algoritmo finaliza cuando el conjunto *Threshold* queda vacío. Cada una de las soluciones localizadas por $PIDMOA^*$ es un par $(\gamma, \vec{f}(\gamma))$, donde γ es un nodo final y $\vec{f}(\gamma) \in C^*$ es el coste de un camino solución desde el nodo inicial hasta γ.

Cada iteración del algoritmo usará un conjunto *Threshold* diferente. El conjunto *Threshold* para la iteración $i+1$, $Threshold_{i+1}$, se calcula lanzando una búsqueda

de tipo *depth-first* desde el nodo raíz, estando la búsqueda dirigida por el conjunto *Threshold* de la iteración i, $Threshold_i$. Durante esta iteración i, en cada nodo n explorado del árbol, en función de los vectores de coste de dicho nodo[1], nos podemos encontrar con las siguientes situaciones:

(i) El nodo n es un nodo que está dominado por soluciones encontradas previamente. En este caso, cualquier nodo final alcanzado a través de n va a tener un vector de coste que también estará dominado por otra solución ya encontrada, es decir, sería una solución subóptima. En consecuencia, se discontinúa la búsqueda en n en la iteración actual. Lógicamente, también se discontinuará la búsqueda en cualquier iteración posterior.

(ii) El nodo n tiene un vector de coste $\vec{f}(n)$ que está dominado de modo estricto por algunos de los vectores del conjunto *Threshold*. Esto implica que en esta rama de la búsqueda no hemos localizado ninguna solución con un vector de coste dentro de los márgenes de búqueda definidos por los vectores del *threshold* actual. Por lo tanto se discontinúa la búsqueda. Además se añade el vector de coste $\vec{f}(n)$ al conjunto $Threshold_{i+1}$, con la idea de poder continuar la búsqueda por esta rama en las posteriores profundizaciones.

(iii) El nodo n es un nodo final. En este caso se añade incondicionalmente el par[2] $(n, \vec{f}(n))$ al conjunto *SOLUTION*, ya que $PIDMOA^*$ está diseñado para localizar únicamente soluciones no dominadas (ver lemas 7.3.4 y 7.3.5).

(iv) Si no se cumple ninguna de las condiciones anteriores, la búsqueda continúa en modo *depth-first*.

Las comprobaciones enumeradas en la lista anterior se traducen en diferentes tipos de tests de dominancia vectoriales:

- Cada nodo generado en el árbol de búsqueda debe comparar su vector de coste contra los vectores incluídos en el conjunto *SOLUTION*, para determinar si la rama en la que se encuentra se sigue explorando o se descarta de modo definitivo porque no vaya a generar una solución que domine o sea indiferente con alguna de las ya localizadas. Esta es una operación de filtrado necesaria en todo algoritmo multiobjetivo y que, paradójicamente, será más eficiente cuanto más tarde localice el algoritmo las soluciones.

[1] En esta tesis emplearemos la notación $F(n)$ para denotar a los vectores de coste de un nodo n alcanzados a través de un camino P. Estos vectores de coste se calculan como $F(n) = \{\vec{g}(n) + \vec{h}(n) \mid \vec{h}(n) \in H(n)\}$, donde $\vec{g}(n)$ es el coste para alcanzar el nodo n a través del camino o rama que está siendo explorado actualmente por el algoritmo. Si el algoritmo, durante la exploración de una rama diferente, alcanza de nuevo el nodo n, probablemente su vector $\vec{g}(n)$ difiera, y por lo tanto también será diferente el conjunto $F(n)$.

[2] Al ser n un nodo final, su conjunto de estimaciones heurísticas incluye únicamente el vector cero ($H(n) = \{\vec{0}\}$), con lo cual $F(n)$ contendrá también un solo vector ($F(n) = \{\vec{g}(n)\}$).

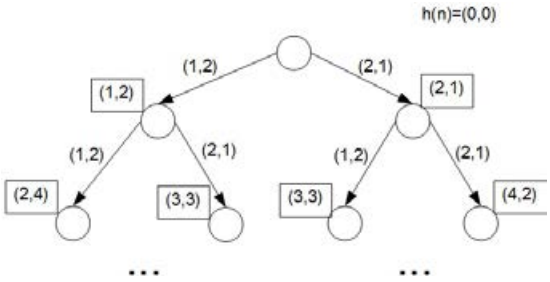


Figura 4.2: Grafo de ejemplo para mostrar la evolución de umbrales en PIDMOA*

- Cada nodo generado en el árbol de búsqueda debe comparar su vector de coste contra los vectores incluídos en el conjunto *Threshold*, para determinar si la rama en la que se encuentra se sigue explorando o por el contrario se discontinúa la búsqueda en la iteración actual, para intentarlo en iteraciones posteriores. Lógicamente, cuanto menor sea la cardinalidad del conjunto, más eficiente será la búsqueda.

- Cuando en el caso anterior se discontinúa una rama, el vector de coste asociado a la misma se añade al conjunto *Threshold* tentativo para la iteración posterior. Sin embargo esta inclusión no es definitiva. Si existen en *Threshold* dos vectores $\vec{th}_1$ y $\vec{th}_2$ tales que $\vec{th}_1 \preceq \vec{th}_2$, comparar durante la búsqueda el vector de coste de un nodo n ($\vec{f(n)}$) contra ambos vectores de *Threshold* es redundante, puesto que si $\vec{f}(n) \preceq \vec{th}_1$, entonces $\vec{f}(n) \preceq \vec{th}_2$ y la búsqueda continúa. En este caso el vector $\vec{th}_2$ podría eliminarse del conjunto *Threshold* sin consecuencias. Para mantener *Threshold* como un conjunto de vectores no dominados se incluyen tests de dominancia cada vez que se amplía este conjunto con algún nuevo vector.

En la figura 4.2 se muestra un grafo de ejemplo con 2 objetivos y costes enteros. El grafo considerado tiene forma de árbol y cada nodo tiene 2 sucesores, siendo los costes de los dos arcos asociados la suma del vector de coste del padre con los vectores (1,2) y (2,1) respectivamente. Cada nodo está etiquetado con el coste del camino que lleva al mismo. Suponemos que el umbral inicial es el vector (0,0) y $\forall n, H(n) = \{(0,0)\}$. Asimismo suponemos que el grafo tiene dos caminos solución, ambos óptimos de Pareto, con costes (4,8) y (9,6).

En la figura 4.3 se puede observar como evoluciona el umbral de $PIDMOA^*$. Los cuadrados sombreados en negro representan el área del espacio de costes descartada por cada uno de los vectores del umbral para la iteración en curso. Se representan también los nodos γ_1 y γ_2, nodos meta de los caminos óptimos mencionados anteriormente ($\vec{f}(\gamma_1) = (4,8)$ y $\vec{f}(\gamma_2) = (9,6)$). En la primera iteración (figura 4.3a) se generan los sucesores (1,2) y (2,1), todos ellos discontinuados al estar descartados por el umbral inicial (0,0). Dado que ambos vectores son no dominados entre sí, ambos pasan a formar parte del umbral para la siguiente iteración (figura 4.3b).

El umbral va avanzando en las diferentes iteraciones hasta que en la iteración reflejada en la figura 4.3e se localiza la solución γ_1 con coste $\vec{f}(\gamma_1) = (4, 8)$. En la última figura (4.3f), el área sombreada en azul representa el área descartada de modo definitivo por esta solución encontrada, pues cualquier vector dentro de ese área estaría dominado por la solución. Las gráficas reflejan claramente como $PIDMOA^*$ promueve la búsqueda en todos los frentes/objetivos, avanzando los umbrales con actualizaciones de valor para todas las componentes de coste.

4.2.2. Ejemplo de $PIDMOA^*$

La figura 4.4a muestra un ejemplo detallado de búsqueda en un árbol binario con dos objetivos. El árbol tiene un nodo raíz s y dos nodos finales (γ_1 y γ_2). Cada uno de las aristas está etiquetada con un único vector de coste con valores para los dos objetivos considerados. Asimismo cada uno de los nodos está etiquetado con un vector heurístico.

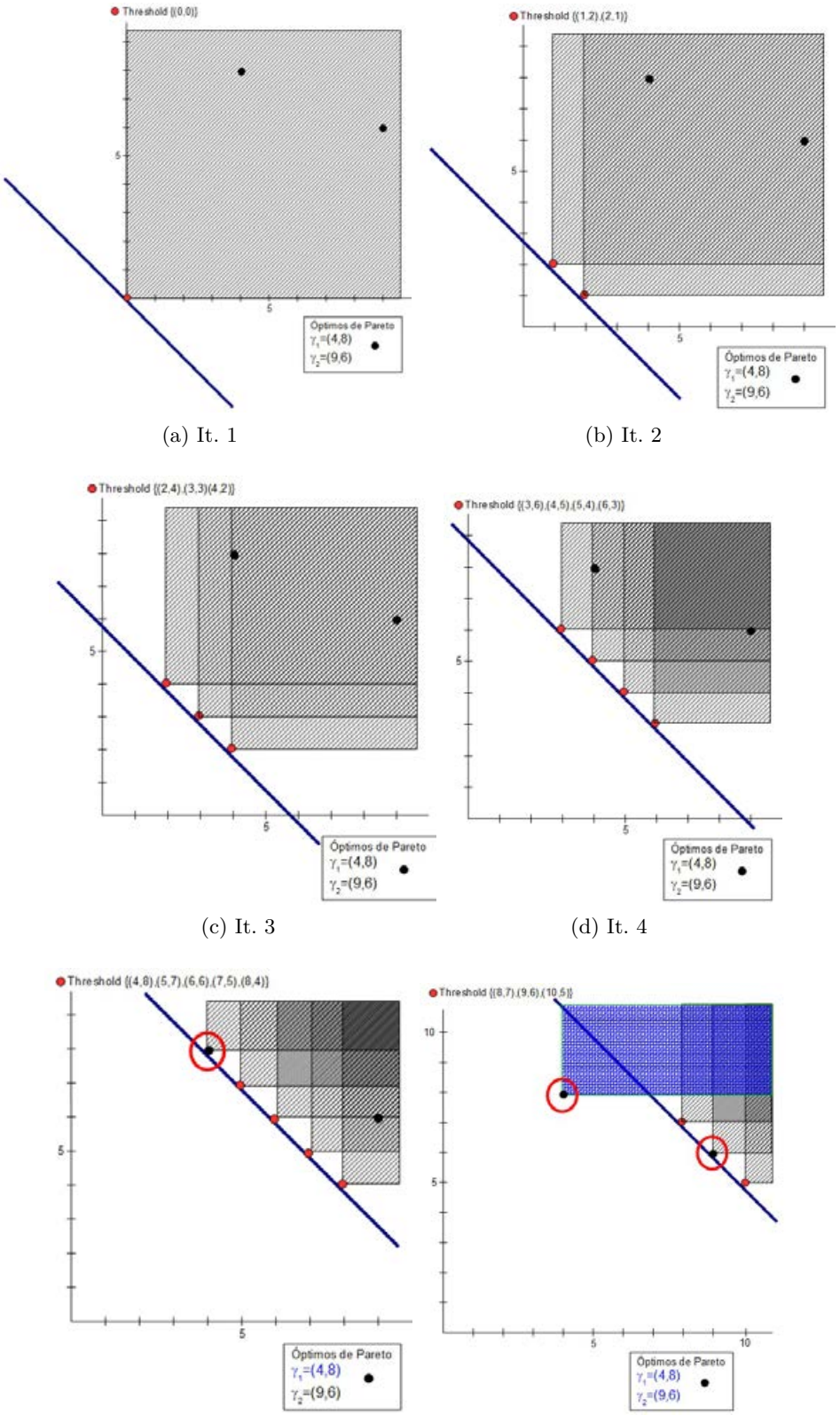
En cada una de las figuras que ilustran este ejemplo se muestra, al lado del árbol explorado en cada iteración, el espacio de costes que se va a explorar en dicha iteración. En el eje X se representa el valor de la primera componente de coste y en el eje Y el valor de la segunda componente. El área sombreada en el espacio de costes se corresponde con vectores de coste que están dominados por el conjunto *Threshold* de la iteración actual, y que por lo tanto serán descartados durante la misma. Dicho de otro modo, en cada iteración $PIDMOA^*$ únicamente explorará el espacio de costes no sombreado.

Aunque en las figuras, para mayor claridad del ejemplo, el árbol se representa completamente expandido, su generación se realiza en modo *depth-first*, con lo cual únicamente la rama de este árbol que está siendo explorada se encontrará almacenada en memoria durante la ejecución del algoritmo.

Inicialmente $PIDMOA^*$ usa como *Threshold* el heurístico del nodo raíz, en este caso un conjunto con un único vector, $H(s) = \{(5, 5)\}$. En la primera iteración $PIDMOA^*$ trata de localizar soluciones no dominadas por este vector de coste. Por lo tanto, todos los nodos cuya estimación de coste $\vec{f}(n)$ no esté dominada por ningún vector del conjunto *Threshold* se expandirán en modo *depth-first*. El conjunto de nodos explorados y el conjunto *Threshold* de esta primera iteración se muestran en la figura 4.4b.

El nodo n_1 se explora, dado que $\vec{f}(n_1) = (6, 4) \sim (5, 5)$ y podría conducir a una solución dentro del espacio que deseamos explorar (p.ej. un posible nodo final con coste (7,4.5)). La expansión del nodo n_1 genera sendos sucesores, γ_1 y n_3, cuyos vectores de estimación de coste son respectivamente $(10, 5)$ y $(7, 5)$. Dado que ambos están estrictamente dominados por el vector $(5, 5)$ de *Threshold*, la búsqueda se discontinúa, añadiendo ambos vectores al conjunto *Threshold* de la siguiente iteración.

Nótese que en este caso, aunque en la exploración del árbol de búsqueda se ha generado un nodo final (γ_1), éste no ha sido expandido, con lo cual el algoritmo no llega a registrarlo como solución del problema. Esto se debe a que el vector $\vec{f}(\gamma_1)$



(a) It. 1 (b) It. 2

(c) It. 3 (d) It. 4

(e) It. 5 (f) It. 6

Figura 4.3: Ejemplo de evolución del umbral de $PIDMOA^*$ sobre el grafo de la figura 4.2

está dominado por el *Threshold* actual. Este mecanismo de detección tardía es el que permite que $PIDMOA^*$ únicamente localice soluciones finales que son óptimos de Pareto.

Tras el correspondiente *backtracking*, se explora la rama derecha del árbol. El nodo n_2 no se expande, debido a que el vector del conjunto *Threshold*, $(5,5)$, domina a $\vec{f}(n_2) = (5,7)$. Por lo tanto el vector $\vec{f}(n_2)$ se incluye en el conjunto *Threshold* de la siguiente iteración, quedando éste como $Threshold = \{(10,5),(7,5),(5,7)\}$. Dado que, para mejorar la eficiencia del algoritmo, el conjunto *Threshold* se mantiene como un conjunto de vectores no dominados entre sí, tras los correspondientes tests de dominancia, y dado que $(7,5) \prec (10,5)$, este último vector se elimina de *Threshold*.

En la segunda iteración (figura 4.4c), *Threshold* se ha actualizado a $\{(7,5),(5,7)\}$, es decir, el conjunto de vectores no dominados de los vectores de estimación de coste de aquellos caminos que han sido discontinuados en la primera iteración.

Todos los nodos generados en la iteración i se generarán también en la iteración $i+1$, y posiblemente alguno más. En nuestro ejemplo, el nodo γ_1, discontinuado en la primera iteración, tampoco se expande en la segunda, ya que sigue dominado por *Threshold* ($(7,5) \prec \vec{f}(\gamma_1) = (10,5)$). En cambio el nodo n_3 sí se explora, ya que no hay ningún vector $\vec{th} \in Threshold$ tal que $\vec{th} \prec \vec{f}(n_3) = (7,5)$. Dado que este nodo n_3 no tiene sucesores, se realiza *backtracking*, devolviendo n_3 un vector de coste $\vec{\infty}$, informando de que a través de este nodo no hay caminos de interés para ser explorados en iteraciones posteriores. A continuación se explora la rama derecha del árbol.

En esta rama, el nodo n_2 sí se explora, a diferencia de lo que ocurría en la primera iteración, puesto que su vector de estimación de coste no está dominado por ningún vector de *Threshold*. Sin embargo sus sucesores, n_4 y γ_2 sí son discontinuados, puesto que $(5,7) \in Threshold$ y $(5,7) \prec \vec{f}(n_4) = (5,8)$ y $(5,7) \prec \vec{f}(\gamma_2) = (5,10)$.

Los nodos que han sido discontinuados al estar dominados por algún vector del *Threshold* son γ_1, n_4 y γ_2, incorporándose al conjunto *Threshold* de la tercera iteración sus respectivos vectores de coste $(10,5)$, $(5,8)$ y $(5,10)$. Análogamente a lo realizado durante la primera iteración, dado que $(5,8) \prec (5,10)$, este último vector del conjunto *Threshold* se elimina para mantener en dicho conjunto únicamente vectores no dominados entre sí. Por lo tanto, para la tercera iteración *Threshold*$= \{(10,5),(5,8)\}$.

Durante esta tercera iteración (figura 4.5a) se explora el nodo γ_1, con coste $(10,5)$, dado que su vector de coste no está dominado de modo estricto por ningún elemento del conjunto *Threshold*. Como γ_1 es un nodo final, el par $(\gamma_1,(10,5))$ se añade al conjunto *SOLUTION*. A partir de este instante, cualquier camino explorado cuyo vector de coste acumulado esté dominado por el coste de la solución encontrada $(10,5)$ será automáticamente descartado, no siendo contemplado tampoco para la generación del siguiente conjunto *Threshold*.

El procesamiento del nodo n_3 es similar al de la iteración anterior: se expande, pero dado que no tiene sucesores, la búsqueda se discontinúa, no teniéndose en cuenta su vector de coste para el próximo conjunto *Threshold*.

De modo análogo, el nodo n_4 sí se explora, pero dado que no tiene sucesores la búsqueda se discontinúa. En cuanto al nodo γ_2, sigue dominado, en este caso por el vector (5,8) de *Threshold*, con lo cual su vector de coste $\vec{f}(\gamma_2) = (5, 10)$ se añade al conjunto *Threshold* de la cuarta iteración.

En esta última iteración (figura 4.5b) el procesamiento del árbol es similar al de la iteración anterior. La principal diferencia es que γ_2 es expandido y se identifica como nodo final, añadiendo el par $(\gamma_2, (5, 10))$ al conjunto *SOLUTION*. Cuando esta iteración finaliza, todas las ramas se han discontinuado bien porque se han localizado nodos finales o bien porque los nodos no tenían sucesores (n_3 y n_4). Aunque en este ejemplo no se haya presentado este caso, también se podría haber discontinuado algún camino porque estuviera dominado por alguna solución ya localizada. Dado que ninguna rama se ha discontinuado por dominancia de un vector de *Threshold* sobre el vector de coste de algún nodo del árbol, el conjunto *Threshold* final calculado está vacío, finalizando por tanto el algoritmo.

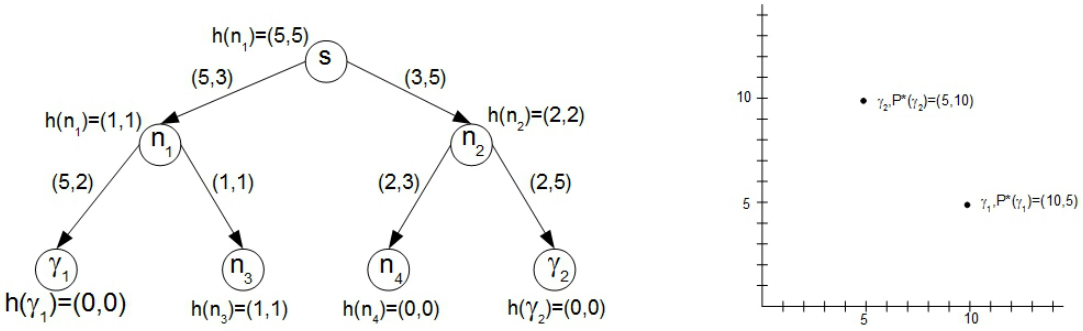
4.3. El algoritmo $LEXIDMOA^*$

El algoritmo $PIDMOA^*$, presentado en la sección anterior, reduce (tal y como se verá en los análisis descritos en el Capítulo 8) el número de reexpansiones de nodos con respecto a $IDMOA^*$. Al mantener un umbral multivectorial, se permite al árbol de búsqueda profundizar simultáneamente en un mayor número de ramas en cada iteración. El inconveniente de este enfoque es que cada nodo procesado debe ser comparado contra este umbral multivectorial, incrementando la cantidad de tests de dominancia realizados en este sentido.

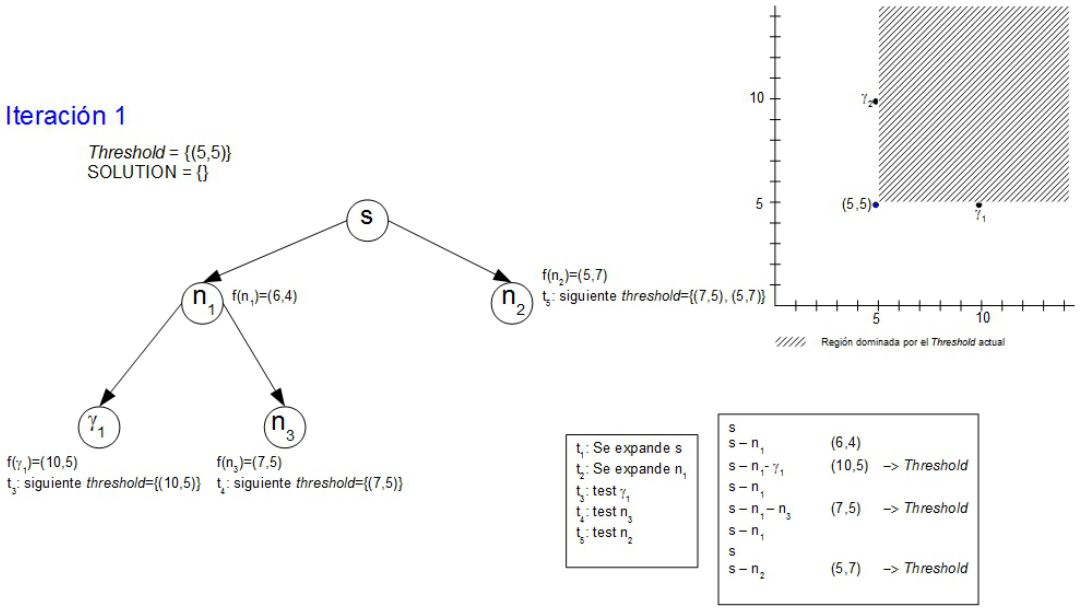
Es por ello que puede resultar de interés el tratar de reducir en la medida de lo posible el tamaño del umbral, incluso a un único vector. Este es el camino adoptado por el algoritmo que se presenta en esta sección, $LEXIDMOA^*$. $LEXIDMOA^*$, al igual que $PIDMOA^*$, discontinúa la búsqueda en todos aquellos nodos dominados por el umbral actual. Sin embargo, a diferencia de $PIDMOA^*$, el umbral asociado a la siguiente iteración se obtiene escogiendo el menor lexicográfico de los vectores de coste de los nodos discontinuados. El orden lexicográfico se presentó en la sección 3.6.3.

Al reducir el umbral a un único vector, $LEXIDMOA^*$ sólo tiene que realizar un test de dominancia contra el vector de coste de cada nodo del árbol de búsqueda para determinar si este nodo será expandido o no en la iteración actual. Obviamente, como contrapartida, esto hará que en cada iteración la profundización del árbol de búsqueda sea menor en comparación con $PIDMOA^*$. En el Capítulo 8 se determinarán los pros y contras de este nuevo enfoque.

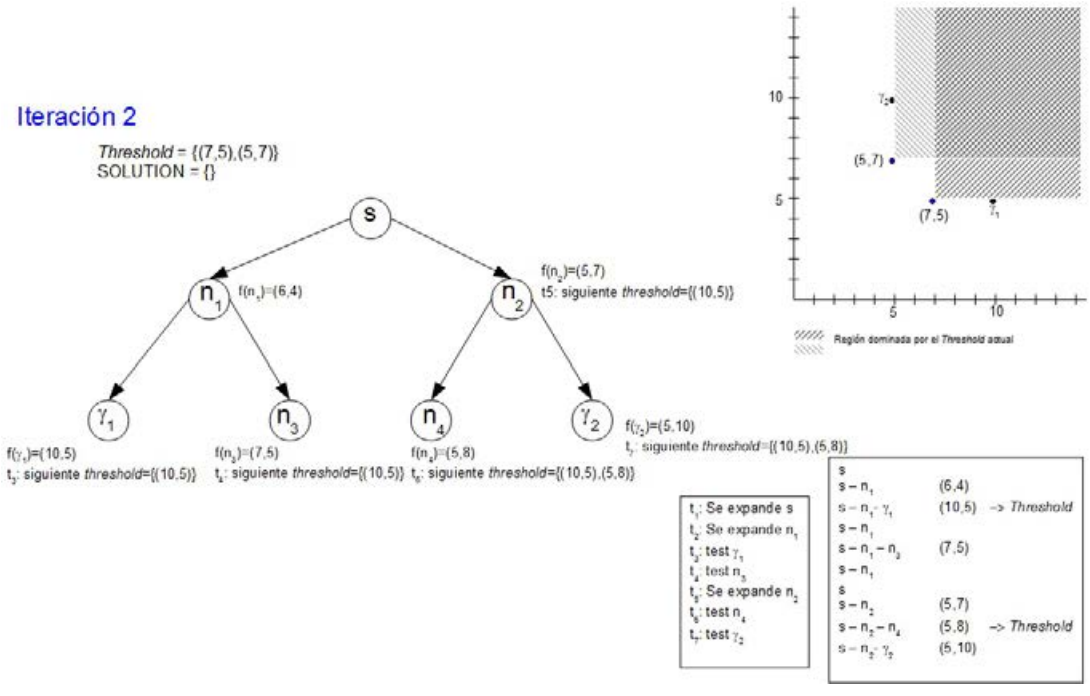
La principal novedad de $LEXIDMOA^*$ frente a algoritmos similares es la inducción de un orden total en el conjunto de vectores no dominados obtenidos a partir de los nodos discontinuados en una iteración del algoritmo. Esta estrategia fue también una de las primeras propuestas en la generalización de algoritmos *best-first* (Martins,



(a) Problema Multiobjetivo

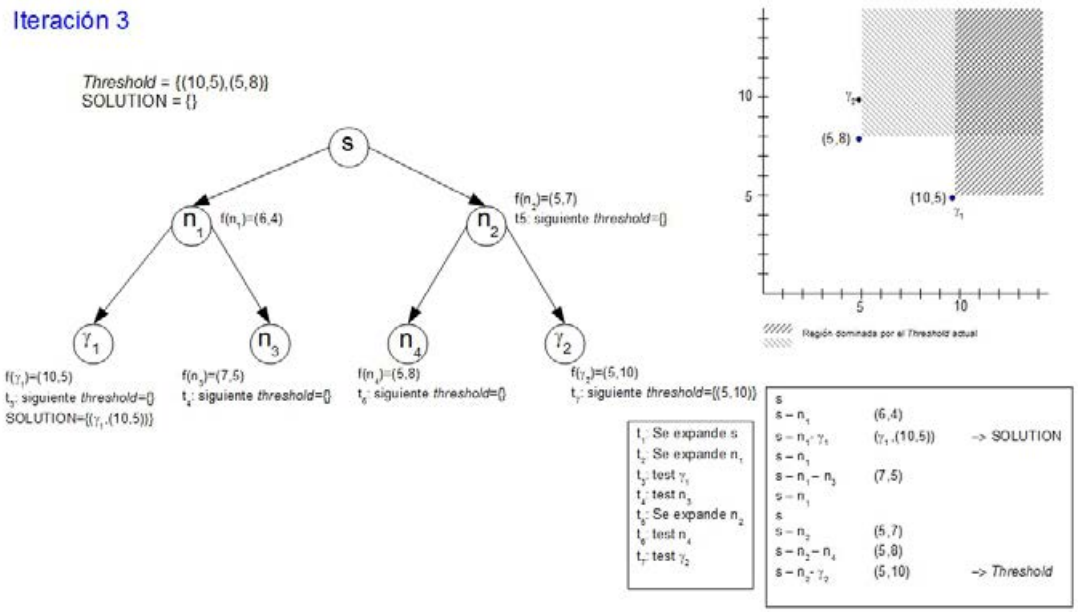


(b) Iteración 1 $PIDMOA^*$

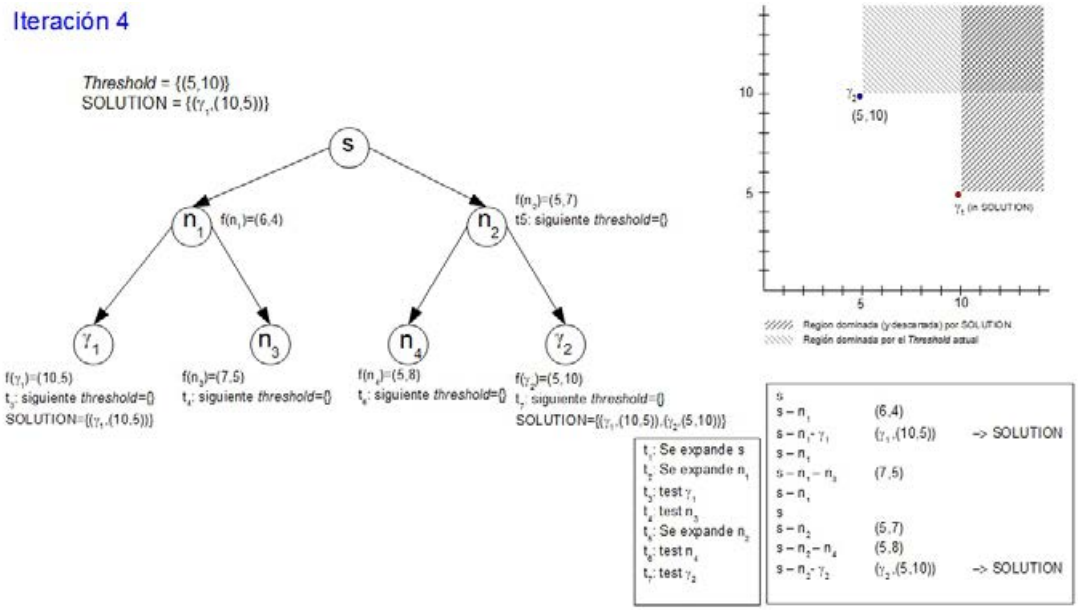


(c) Iteración 2 $PIDMOA^*$

Figura 4.4: Ejemplo de $PIDMOA^*$: Problema e iteraciones 1-2



(a) Iteración 3 $PIDMOA^*$



(b) Iteración 4 $PIDMOA^*$

Figura 4.5: Ejemplo de $PIDMOA^*$: Iteraciones 3-4

1984).

4.3.1. Descripción del algoritmo

El funcionamiento de $LEXIDMOA^*$ es similar al de $PIDMOA^*$, radicando las principales diferencias en el cálculo del umbral y en el criterio de expansión o discontinuación de un nodo.

Al igual que $PIDMOA^*$, cuando $LEXIDMOA^*$ discontinúa un nodo, el vector de coste del mismo se tiene en cuenta para calcular el umbral de la siguiente iteración. Mientras $PIDMOA^*$ acumula estos vectores de coste en un conjunto umbral (eliminando si fuera necesario los vectores dominados), $LEXIDMOA^*$ calcula el menor lexicográfico de dicho conjunto. Este menor lexicográfico se corresponde, obviamente, con el vector de coste de alguno de los caminos discontinuados y es el que servirá de cota superior para la generación del árbol de búsqueda en la siguiente iteración del algoritmo.

En cuanto al criterio de expansión, $PIDMOA^*$ expande un nodo siempre y cuando no esté dominado por ningún vector del conjunto umbral. Sin embargo $LEXIDMOA^*$ expande un nodo cuando su vector de coste es menor lexicográfico que el único vector que conforma el umbral.

Este comportamiento hace que se reduzcan los tests que el algoritmo tiene que realizar para determinar si un nodo será expandido o no. Únicamente es necesario comparar el vector de coste del nodo contra el único vector umbral. Sin embargo este vector umbral es el que va a dirigir la búsqueda en la siguiente iteración, provocando que únicamente se expanda la rama asociada a los nodos con vectores de coste igual a la cota. Por todo ello, la cantidad de nodos nuevos que se expandan de una iteración a otra no será tan numerosa como en el caso de un umbral multivectorial. En el Capítulo 8 se analizará el impacto de este comportamiento sobre el rendimiento de $LEXIDMOA^*$.

Una característica a tener en cuenta de $LEXIDMOA^*$ es que únicamente localiza óptimos de Pareto, nunca soluciones dominadas (al igual que $PIDMOA^*$), no necesitando por ello tests de dominancia adicionales para purgar el conjunto de soluciones que vaya obteniendo durante la resolución del problema.

El pseudocódigo del algoritmo $LEXIDMOA^*$ se muestra en la tabla 4.2.

La función principal $LEXIDMOA^*$ se encarga de explorar los sucesivos árboles acotados, suministrando los correspondientes umbrales de expansión. El umbral inicial es el menor lexicográfico de todos los vectores heurísticos asociados al nodo raíz del árbol. Una vez computado el umbral, se realiza una exploración de tipo *depth-first* sobre el árbol, invocando a la función $DFSLEX$ y usando el umbral anterior como cota superior.

Si el nodo considerado tiene vectores de coste dominados por alguna de las soluciones ya encontradas, la búsqueda en dicho nodo se discontinúa y sus vectores de coste no se tienen en cuenta para calcular el umbral de la siguiente iteración. Esto hace que el nodo no sea expandido en ninguna de las iteraciones posteriores.

```
LEXIDMOA* (G, s, Γ )
SOLUTION ← ∅; threshold ← menorlexicografico(H(s))
WHILE threshold ≠ ∅
     (threshold, SOLUTION) ← DFSLEX (s, threshold, SOLUTION)
return (nodomset(SOLUTION))


DFSLEX (node, currentTh, SOLUTION)
NdomSol ← {f ∈ F(node) | (∄(γ, c*) ∈ SOLUTION
                | c* ⪯ f)}
IF (NdomSol = ∅) THEN return (∅, SOLUTION);
NdomTh ← {f ∈ NdomSol | f ≤lex currentTh}
IF (NdomTh = ∅) THEN return (NdomSol, SOLUTION);
IF (node ∈ Γ ) THEN
     SOLUTION ← SOLUTION ∪ {(node, f(node))}
     return (∅, SOLUTION)
ELSE
     ThresholdDFS ← ∅
     Sucessors ← expand_node (node)
     FOR each suc in Sucessors DO
          (ThresholdRT, SOLUTION) ← DFSLEX(suc, currentTh, SOLUTION)
          ThresholdDFS ← mejorlexicografico (ThresholdDFS ∪ ThresholdRT)
     return (ThresholdDFS, SOLUTION)
```

Tabla 4.2: Algoritmo $LEXIDMOA^*$

Por el contrario, si los costes de un nodo no están dominados por ninguna solución, entonces el camino que lleva a dicho nodo es susceptible de poder generar una ruta óptima a un nodo meta, en cuyo caso debería ser explorado. Ahora la decisión radica en determinar si será explorado en la iteración actual o en una posterior. Para ello se comprueba si el umbral es menor lexicográfico que el coste del nodo a considerar. En caso afirmativo no se produce la expansión del nodo, devolviendo su vector de coste para que así sea tenido en cuenta en el cálculo del próximo umbral.

Si el coste del nodo es mejor lexicográfico que el umbral, entonces el nodo debe procesarse. En primer lugar comprobamos si es un nodo meta, en cuyo caso se añade directamente al conjunto de soluciones óptimas C^*. Si no es un nodo meta, entonces se procede a su expansión. Para ello se generan sus sucesores y se invoca sucesiva y recursivamente a la función $DFSLEX$. Cuando esta expansión finalice, las soluciones óptimas encontradas en el correspondiente subárbol explorado se habrán añadido al conjunto $SOLUTION$ y de los vectores de coste de los caminos discontinuados en dicho subárbol se obtiene el mejor lexicográfico.

En la sección 4.3.2 se muestra un ejemplo del funcionamiento de este algoritmo.

4.3.2. Ejemplo de $LEXIDMOA^*$

En esta sección mostraremos un ejemplo de resolución de un problema multiobjetivo con el algoritmo $LEXIDMOA^*$. En la figura 4.6a se muestra un sencillo problema de búsqueda con 2 objetivos, en el cual cada nodo está etiquetado con sus vectores heurísticos (en este caso, un único vector), y cada arco está etiquetado con un vector de coste. El nodo raíz del problema es s. El conjunto de nodos finales, Γ, incluye los nodos γ_1, γ_2 y γ_3. Los caminos que conducen a γ_2 y γ_3 son ambos soluciones no dominadas, mientras que el camino a γ_1 es subóptimo, ya que su coste $(5, 15)$ está dominado por el coste del camino que lleva a γ_2, $(5, 12)$. Al igual que en la explicación del ejemplo de la sección 4.2.2, el árbol se dibuja completo para cada iteración, aunque en la práctica el árbol explorado se expande de forma *depth-first* de izquierda a derecha, manteniendo una única rama del mismo en memoria de modo simultáneo.

En la primera iteración de $LEXIDMOA^*$ (figura 4.6b), el umbral inicial es el mejor lexicográfico de los vectores heurísticos del nodo raíz, en este caso el vector $(0, 0)$. Se inicia una búsqueda *depth-first* sobre el espacio de estados, discontinuando la búsqueda cuando el umbral actual sea menor lexicográfico que el vector de coste del nodo considerado. En nuestro ejemplo, los vectores de coste de n_1 y n_2 son $(2, 7)$ y $(4, 5)$ respectivamente, ambos *peores lexicográficos* que el umbral actual $(0, 0)$. La búsqueda se discontinúa en ambas ramas y sus respectivos costes se utilizan para computar el umbral de la siguiente generación, buscando el menor lexicográfico de ambos. En este caso el nuevo umbral será el vector $(2, 7)$.

Como ya se comentó en la sección 4.3.1, este umbral va a provocar que la exploración del árbol en la siguiente iteración avance únicamente por las ramas asociadas a nodos con vectores de coste iguales al umbral, progresando la profundización en general muy

lentamente. En la segunda iteración de $LEXIDMOA^*$ (figura 4.6c) el nodo n_1 sí se expande, al no ser el umbral mejor lexicografico que el vector de coste de n_1. Se generan en este caso los nodos meta γ_1 y γ_2, con vectores de coste $(5, 15)$ y $(5, 12)$ respectivamente. Al ser ambos vectores peores lexicográficos que el umbral, la búsqueda se discontinúa. El nodo n_2 se discontinúa por idéntico motivo. Al no quedar ramas pendientes de explorar en la iteración actual, se calcula el siguiente umbral, resultando el vector $(4, 5)$.

En la tercera iteración (figura 4.6d) la búsqueda procede de modo similar, expandiéndose en este caso el nodo n_2 y generándose n_3 y γ_3, siendo el umbral mejor lexicográfico que sus vectores de coste. El nuevo umbral pasa a ser el vector $(5, 12)$.

La cuarta iteración (figura 4.7a) presenta como novedad la expansión del nodo γ_2, con vector de coste $(5, 12)$ igual al umbral. Debido a ello el nodo se procesa y, dado que es un nodo meta, se añade al conjunto de soluciones finales. Nótese que, dado que suponemos que el árbol se explora de izquierda a derecha, el nodo γ_1 ya ha sido revisado, pero, dado que el umbral era mejor lexicográfico que el coste de de γ_1, dicho vector de coste se usa para computar el siguiente umbral. Si la expansión se hubiera realizado de derecha a izquierda, se habría localizado antes el óptimo de Pareto $(5, 12)$ y se hubiera provocado una poda en γ_1.

Por lo que respecta al resto de nodos, n_3 está dominado por la solución γ_2, con lo cual este nodo se descarta incluso para el cálculo del siguiente umbral. Por el contrario, γ_3 sí se tiene en cuenta para dicho cálculo. El siguiente umbral resulta ser el vector $(5, 15)$.

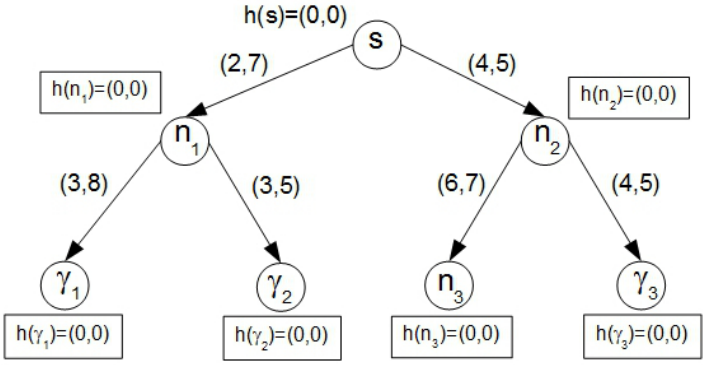
En la quinta iteración (figura 4.7b) sí se procesa el nodo γ_1, pero como este nodo meta está dominado por γ_2, se descarta. El nodo n_3 también se descarta por el mismo motivo que en la iteración anterior. El umbral no es mejor lexicográfico que el coste del nodo γ_3, con lo cual éste se convierte en el único candidato para el cómputo del último umbral, $(8, 10)$.

En la última iteración (figura 4.7c) la única novedad consiste en el procesamiento del nodo γ_3, el cual, al no ser el umbral menor lexicográfico que su vector de coste, se incorpora como solución al conjunto $SOLUTION$. Dado que el resto de ramas son, o bien soluciones óptimas o bien nodos descartados al estar dominados por soluciones ya encontradas, no hay vectores candidatos para cálculo de umbral y el algoritmo finaliza, devolviendo el conjunto de óptimos de Pareto.

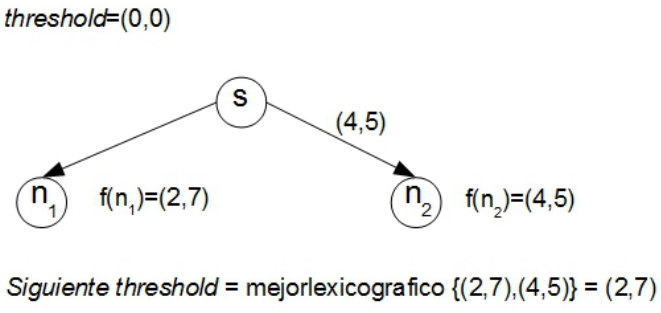
4.4. El algoritmo $IPID^*$

En esta sección presentamos $IPID^*$ (*Ideal Point Iterative Deepening*) (Coego et al., 2012), una variante multiobjetivo de IDA^* que intenta combinar las ventajas de $PIDMOA^*$ y $LEXIDMOA^*$.

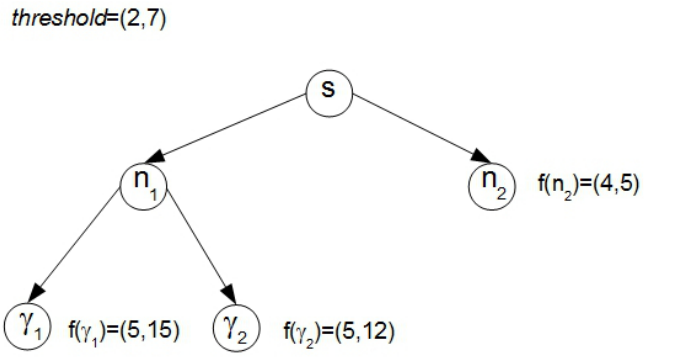
$IPID^*$, al igual que $PIDMOA^*$, mantiene un umbral vectorial. Esta es una condición necesaria para poder procesar información de todos los objetivos de modo si-



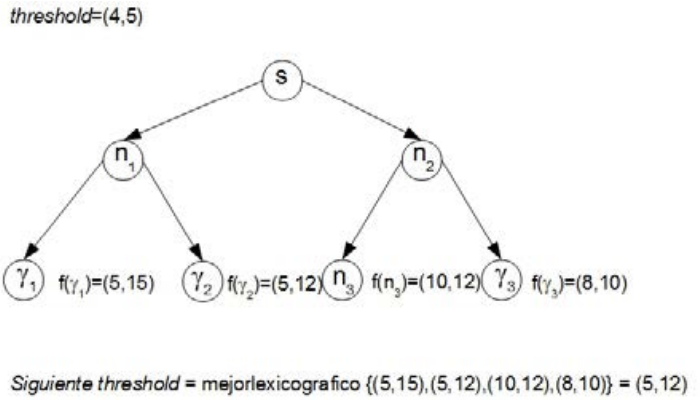
(a) Un problema de búsqueda



(b) Iteración 1 $LEXIDMOA^*$

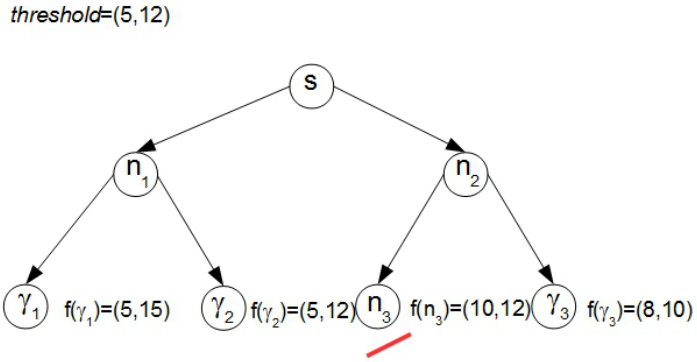


(c) Iteración 2 $LEXIDMOA^*$

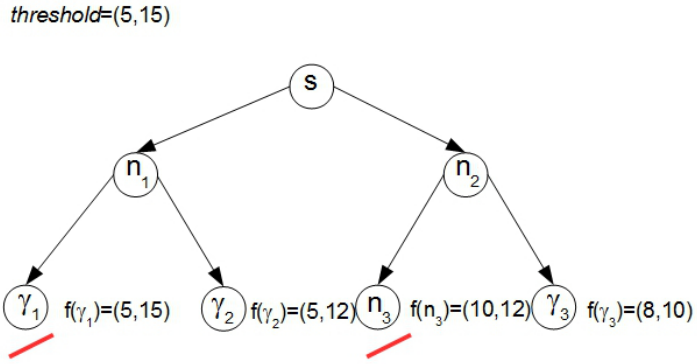


(d) Iteración 3 $LEXIDMOA^*$

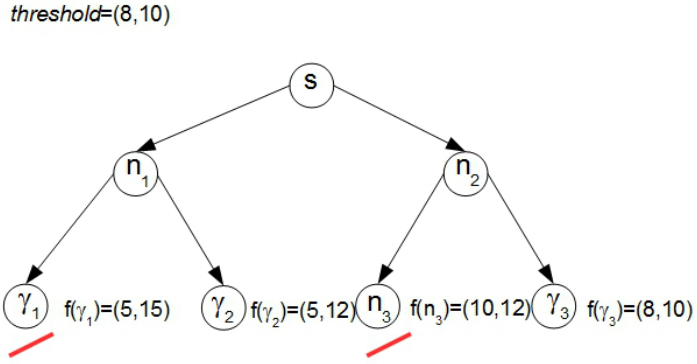
Figura 4.6: Ejemplo de $LEXIDMOA^*$: problema e iteraciones 1-3



(a) Iteración 4 $LEXIDMOA^*$



(b) Iteración 5 $LEXIDMOA^*$



(c) Iteración 6 $LEXIDMOA^*$

Figura 4.7: Ejemplo de $LEXIDMOA^*$: iteraciones 4-6

multáneo. Cada componente del vector de coste está asociado a un objetivo diferente. Por lo tanto, para comparar el coste de un nodo con la cota de búsqueda (establecida por el umbral) o con soluciones ya encontradas, y poder determinar si existe o no dominancia, se debe procesar dicho vector de coste en su totalidad. La diferencia principal con $PIDMOA^*$ es que en $IPID^*$ el umbral está formado por un único vector. De este modo, aunque seguirán siendo necesarios los tests de dominancia para controlar la profundización, sólo será necesario comprobar el vector de estimación de coste de un nodo contra el único vector umbral existente. En este aspecto $IPID^*$ es similar a $LEXIDMOA^*$, aunque la naturaleza de la comparación a realizar difiera: test de dominancia en el caso de $IPID^*$ y test de orden lexicográfico en el caso de $LEXIDMOA^*$.

4.4.1. Descripción del algoritmo

Comenzaremos la descripción de $IPID^*$ partiendo de dos definiciones: el punto ideal y la relación *estrictamente mejor*.

Definición 4.4.1. *Dado un conjunto de vectores T, el punto ideal de T ($i\vec{Point}(T)$) es un vector formado por los valores más pequeños ($i\vec{Point}(T)[j]$) que pueden ser localizados en cualquier vector del conjunto T para cada componente, es decir:*

$$\forall j, 1 \leq j \leq q, i\vec{Point}(T)[j] = \text{mín}\{\vec{v}[j] \mid \vec{v} \in T\} \tag{4.1}$$

Por ejemplo, dado un conjunto $T = \{(1,9),(4,5),(7,2)\}$, el punto ideal de T ($i\vec{Point}(T)$) será un vector con 2 componentes calculadas como sigue:

- $i\vec{Point}(T)[1] = \text{mín}(t_{11}, t_{21}, t_{31}) = \text{mín}(1,4,7) = 1$
- $i\vec{Point}(T)[2] = \text{mín}(t_{12}, t_{22}, t_{32}) = \text{mín}(9,5,2) = 2$

Por lo tanto, $i\vec{Point}(T) = (1,2)$. Se puede observar una representación gráfica de este concepto en la figura 4.8. Por construcción del punto ideal, éste siempre será un vector que domine o iguale a los vectores a partir de los cuales se ha calculado, es decir, $\forall \vec{t} \in T, i\vec{Point}(T) \preceq \vec{t}$.

Definición 4.4.2. *Definimos la relación* estrictamente mejor *($\ll$) como:*

$$\vec{g} \ll \vec{g'} \Leftrightarrow \forall i (1 \leq j \leq q), \vec{g}[j] < \vec{g'}[j]. \tag{4.2}$$

Esta relación se usará posteriormente en el algoritmo $IPID^*$ de modo complementario a las relaciones típicas de dominancia. A continuación describimos el funcionamiento básico de $IPID^*$. Al ser una generalización de la profundización iterativa, $IPID^*$ precisa calcular un umbral que sirva de cota ó *threshold* en cada una de las iteraciones. La cota para la primera iteración se obtiene a partir del conjunto de vectores heurísticos del nodo raíz, calculando el punto ideal de dicho conjunto. Este punto ideal sirve de cota inicial. Para el resto de iteraciones, $IPID^*$ calcula su correspondiente

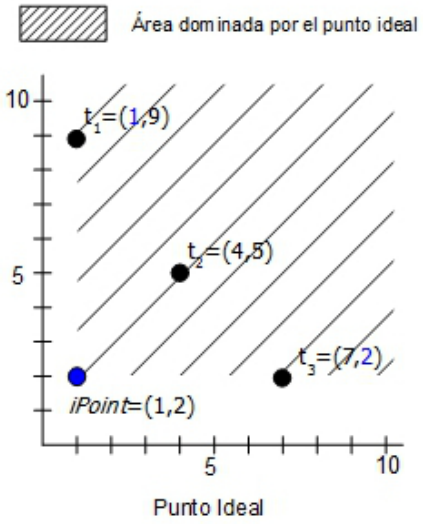


Figura 4.8: Punto Ideal

umbral como el punto ideal de los vectores de coste de aquellos nodos en los cuales se ha discontinuado la búsqueda. Por lo tanto, el umbral de $IPID^*$ está formado por un único vector, lo cual permitirá reducir considerablemente el número de tests de dominancia que se realizan durante la búsqueda, tal y como veremos en el Capítulo 8.

Una vez calculado el vector umbral de una iteración, se realizan las correspondientes búsquedas en modo *depth-first* usando dicho vector como cota. Cuando el camino que se está explorando está dominado por alguna solución que ya haya sido localizada previamente, entonces la búsqueda se discontinúa, descartando este camino tanto en esta iteración como en las sucesivas, ya que en este caso no es posible que el camino conduzca a un nodo final óptimo de Pareto. Como veremos en el Capítulo 7, para garantizar la admisibilidad de $IPID^*$ cada arco debe tener un coste mínimo positivo, haciendo que en la expansión de una rama los costes vayan creciendo estrictamente para cada uno de los objetivos, es decir, a caminos de longitud infinita corresponderán siempre costes infinitos.

Por otro lado, si el camino explorado es estrictamente peor que el umbral (o de modo análogo, el umbral es estrictamente mejor que el coste del camino explorado), entonces el camino también se discontinúa, pero en este caso su vector de estimación de coste se incluye en el conjunto *Threshold* que se usará para calcular el punto ideal que servirá de umbral en la siguiente iteración. El motivo de chequear si el umbral es estrictamente mejor que el coste en lugar de realizar un test de dominancia, se debe a que, en este último caso, $IPID^*$ podría calcular de modo indefinido el mismo vector umbral, no finalizando el algoritmo. La sección 4.4.2 ilustra un ejemplo en el cual se produce esta situación, y como el uso de la relación *estrictamente mejor* permite solventarla.

Si el camino actual no se discontinúa, entonces $IPID^*$ comprueba si el nodo actual

```
IPID (G, s, Γ )
SOLUTION ← ∅; ThresholdSet ← H(s)
WHILE ThresholdSet ≠ ∅
     threshold⃗ ← iPoint (ThresholdSet)
     (ThresholdSet, SOLUTION) ← DFS (s, threshold⃗, SOLUTION)
return (nodomset(SOLUTION))


DFS (n, currentTh⃗, SOLUTION)
NdomSol ← {f⃗ ∈ F(n) | (∄(γ, c⃗) ∈ SOLUTION
                | c⃗ ⪯ f⃗)}
IF (NdomSol = ∅) THEN return (∅, SOLUTION);
NdomTh ← {f⃗ ∈ NdomSol | ¬(currentTh⃗ ≪ f⃗)}
IF (NdomTh = ∅) THEN return (NdomSol, SOLUTION);
IF (n ∈ Γ ) THEN
     SOLUTION ← SOLUTION ∪ {(n, f⃗(n))}
     return (∅, SOLUTION)
ELSE
     ThresholdDFS ← ∅
     sucessors ← expand_node (n)
     FOR each suc in sucessors DO
          (ThresholdRT, SOLUTION) ← DFS(suc, currentTh⃗, SOLUTION)
          ThresholdDFS ← nodomset (ThresholdDFS ∪ ThresholdRT)
     return (ThresholdDFS, SOLUTION)
```

Tabla 4.3: Algoritmo IPID

es un nodo solución. En caso afirmativo, entonces se añade al conjunto de soluciones junto con su vector de coste. $IPID^*$, debido al uso del punto ideal, puede localizar temporalmente soluciones subóptimas. Esta situación se ilustrará en el primer ejemplo mostrado en la sección 4.4.2. Estas soluciones no afectan a la admisibilidad de $IPID^*$, pues serán descartadas en pasos posteriores del algoritmo, pero generan tests de dominancia extras. Como veremos en el Capítulo 8, en contra de lo que se podría esperar, este cómputo extra no afecta de modo significativo al rendimiento del algoritmo.

Cuando en una iteración todos los caminos son discontinuados, bien por ser soluciones, bien por estar dominados por alguna de las soluciones encontradas previamente, entonces el algoritmo finaliza, devolviendo el conjunto completo de soluciones óptimas del problema.

Teniendo en cuenta estas consideraciones, el pseudocódigo completo del algoritmo $IPID^*$ se muestra en la tabla 4.3.

En el algoritmo, la función *IPID* es la encargada de calcular el umbral de las diferentes iteraciones con el cómputo del punto ideal, realizando para cada iteración la correspondiente llamada a la función *DFS*, pasando como parámetros el nodo raíz y el vector umbral. Cuando el umbral está vacío, la función *IPID* finaliza, devolviendo el

conjunto de soluciones. La función *DFS* es la encargada de realizar las profundizaciones en modo *depth-first*, procesando las expansiones y retrocesos en función de los vectores de coste en el camino explorado, el conjunto solución calculado hasta el momento y el vector umbral usado como cota.

A pesar de que $IPID^*$ comparte paradigma con los algoritmos $PIDMOA^*$ y $LEXIDMOA^*$ y tiene un funcionamiento similar, las diferencias entre estos algoritmos son claves en su rendimiento. Estas diferencias vienen dadas fundamentalmente por dos aspectos: el tamaño de los umbrales y su cálculo. En lo relativo al tamaño del umbral, $PIDMOA^*$ emplea como tal un conjunto que incluye todos los vectores de coste no dominados de los caminos que han sido discontinuados en la iteración actual. Esto hace que este conjunto umbral sea habitualmente multivectorial, y, como veremos en el Capítulo 8, de un tamaño tal que degrada significativamente el rendimiento. $LEXIDMOA^*$ e $IPID^*$ también calculan este conjunto, pero no lo usan directamente, sino que, una vez finalizada la iteración, lo reducen a un único vector: el mejor lexicográfico en el caso de $LEXIDMOA^*$ y el punto ideal en el caso de $IPID^*$. Este vector es el que usarán de umbral en la siguiente iteración del algoritmo. La principal ventaja de usar un umbral con un único vector es la considerable reducción del número de tests de dominancia realizados por los algoritmos para determinar si se discontinúa un camino por superar dicho umbral.

La otra diferencia importante atañe al procesamiento realizado sobre el umbral. En el caso de $PIDMOA^*$, este cómputo elimina los vectores dominados del umbral; $LEXIDMOA^*$ calcula el mejor lexicográfico e $IPID^*$ calcula el punto ideal. Si bien este último cálculo es computacionalmente menos costoso, al comparar exclusivamente los valores escalares de cada objetivo, el uso del punto ideal como umbral plantea una situación que no se da en $PIDMOA^*$ o $LEXIDMOA^*$: $IPID^*$ puede incluír de modo temporal (al igual que lo hace $IDMOA^*$) soluciones subóptimas dentro del conjunto solución. Por este motivo se ve obligado a realizar tests de dominancia extras cada vez que se localiza una solución, comprobando si ésta domina a alguna de las ya existentes, en cuyo caso se elimina esta última.

El cálculo del umbral repercute directamente en su *calidad*, en la cantidad de información que proporcionan a iteraciones posteriores. $PIDMOA^*$, al mantener la frontera de vectores de coste no dominados de la iteración anterior, proporciona una información más completa que permite reducir el número de reexpansiones en el árbol de búsqueda. En lo que respecta a $IPID^*$ y $LEXIDMOA^*$, el comportamiento de éste último al fijar como umbral el vector de coste de uno de los caminos discontinuados hace que en la siguiente iteración se expanda la rama asociada a dicho camino (y posiblemente alguna otra más que tuviera un coste idéntico). La expansión de las restantes ramas se realizará en iteraciones posteriores. Esto provoca un lento avance en la exploración del árbol de búsqueda. Por el contrario, $IPID^*$ genera un umbral basado en información obtenida de diferentes vectores de coste, lo cual favorece una mayor expansión de nodos nuevos en las siguientes iteraciones (aunque obviamente no tan grande como la efectuada con el umbral multivectorial de $PIDMOA^*$).

En la siguiente sección se presenta un ejemplo del funcionamiento del algoritmo $IPID^*$.

4.4.2. Ejemplo de $IPID$

En esta sección mostraremos un ejemplo de resolución de un problema con el algoritmo $IPID^*$. Asimismo ilustraremos el motivo por el cual ha sido necesario incluír la relación *estrictamente mejor* para decidir la expansión o no expansión de un nodo, sustituyendo a la relación de dominancia empleada por $PIDMOA^*$.

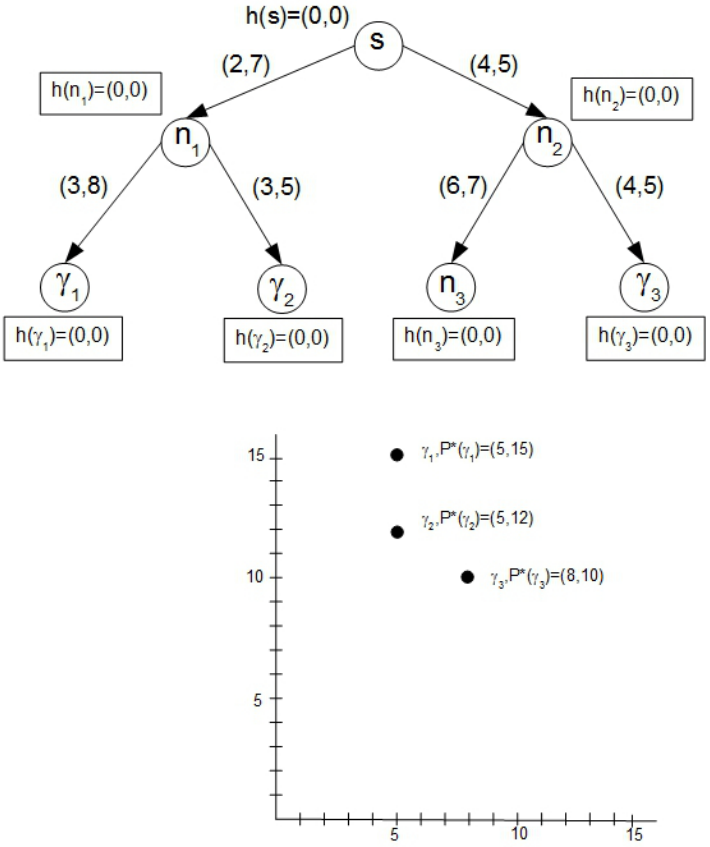
En la figura 4.9a aparece el mismo problema resuelto en la sección 4.3.2 empleando el algoritmo $LEXIDMOA^*$.

En la primera iteración de $IPID^*$ (figura 4.9b) se calcula el umbral inicial como $iPoint(H(s)) = \vec{h}(s) = (0,0)$. A continuación se lanza una búsqueda *depth-first* acotada por este vector umbral. Dado que $\vec{f}(s) = (0,0)$ no es estrictamente peor que el umbral actual $(0,0)$, el nodo s se expande. Una vez generado el sucesor n_1 se comprueba que el umbral actual es estrictamente mejor que el vector de coste del nodo ($\vec{threshold} = (0,0) \ll \vec{f}(n_1) = (2,7)$), con lo cual la búsqueda se discontinúa en esta rama y $\vec{f}(n_1)$ se añade al conjunto usado para calcular el próximo umbral. Lo mismo sucede con el otro sucesor, n_2, ya que $\vec{threshold} = (0,0) \ll \vec{f}(n_2) = (4,5)$.

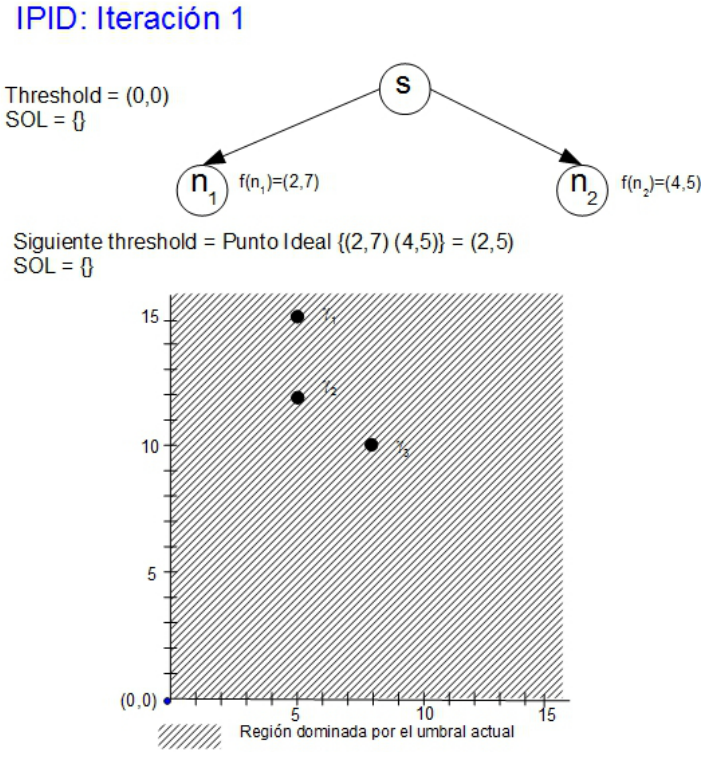
Al no haber más nodos susceptibles de expansión, $IPID^*$ procede a calcular el umbral de la segunda iteración como $\vec{threshold} = iPoint\{(2,7),(4,5)\} = (2,5)$. Con este nuevo umbral arranca la nueva iteración de $IPID^*$ (figura 4.10a). En esta figura, a efectos de comparación, se muestra el área *descartada* a efectos de búsqueda por el umbral usado por $IPID^*$ y el umbral que habría descartado un algoritmo como $PIDMOA^*$ si se hubiera mantenido como umbral el conjunto de vectores $\{(2,7)(4,5)\}$.

Como podemos observar, el umbral definido por $IPID^*$ es más restrictivo que el que definiría $PIDMOA^*$, el espacio de costes que descarta es mayor. Al calcular el punto ideal como el mínimo de los valores para cada componente, el punto ideal *retrocede* frente al umbral original a partir del cual se calcula (el conjunto de vectores discontinuados durante la iteración). Si A($\{\vec{v}_1, \vec{v}_2, \dots\}$) es el área del espacio de costes dominada por los vectores $\vec{v}_1$, $\vec{v}_2$, ..., entonces $A(\{(2,5)\}) - A(\{(2,7),(4,5)\})$ representa un área que un algoritmo como $PIDMOA^*$ sí exploraría en una nueva iteración, pero que un algoritmo como $IPID^*$ descarta. El resultado de este comportamiento es que las cotas que establezca $IPID^*$ para la profundización iterativa avanzarán más despacio que en $PIDMOA^*$, aumentando posiblemente el número de iteraciones y, por tanto, de reexpansiones.

En esta segunda iteración, el nuevo umbral $(2,5)$ es *estrictamente peor* que el umbral previo $(0,0)$. El nodo inicial por lo tanto se expande, generando de nuevo en primer lugar el sucesor n_1. Como $\vec{f}(n_1) = (2,7)$ no es estrictamente peor que el umbral actual $(2,5)$, el nodo n_1 se expande. Primero se genera el sucesor γ_1, con $\vec{threshold} = (2,5) \ll \vec{f}(\gamma_1) = (5,15)$. Al ser γ_1 estrictamente peor que el *threshold*, la búsqueda se discontinúa y se almacena el vector $(5,15)$ para el cálculo del siguiente umbral. Lo

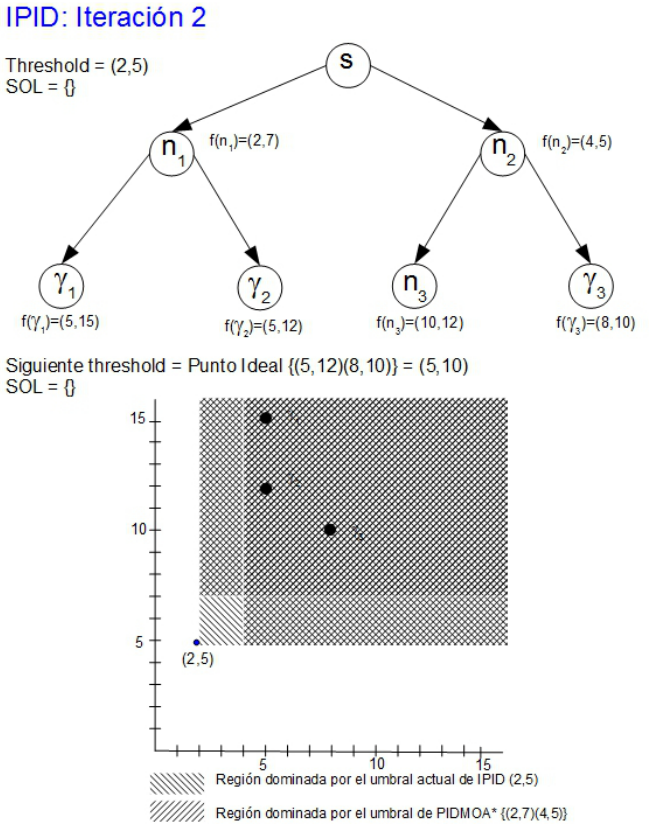


(a) Un problema de búsqueda y sus soluciones en el espacio de costes

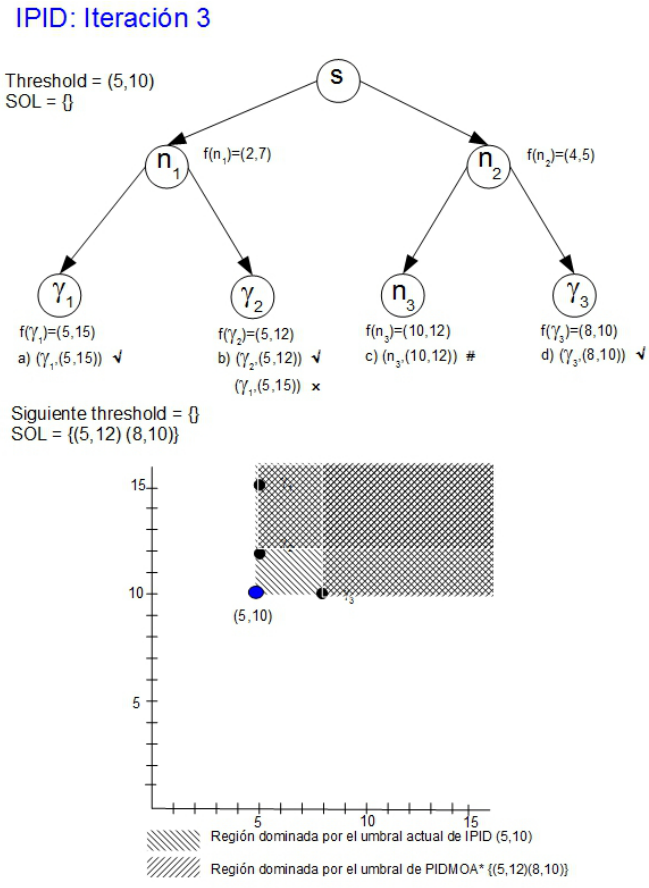


(b) Iteración 1 IPID

Figura 4.9: Ejemplo de IPID: problema e iteración 1



(a) Iteración 2 IPID



(b) Iteración 3 IPID

Figura 4.10: Ejemplo de IPID: Iteraciones 2-3

mismo sucede con el otro sucesor de n_1, γ_2, ya que $\vec{threshold} = (2,5) \ll \vec{f}(\gamma_2) = (5,12)$.

Al procesar la rama derecha del árbol el comportamiento del algoritmo es similar. El otro sucesor de s, n_2 tiene un coste $\vec{f}(n_2) = (4,5)$, el cual no es estrictamente peor que el umbral $(2,5)$. En este caso sí procede la expansión de n_2, y por lo tanto se generan alternativamente sus sucesores, los nodos n_3 y γ_3. En ambos nodos se discontinúa la búsqueda ya que $\vec{threshold} = (2,5) \ll \vec{f}(n_3) = (10,12)$ y $\vec{threshold} = (2,5) \ll \vec{f}(\gamma_3) = (8,10)$.

Estos cuatro vectores de coste de otros tantos nodos discontinuados se usarán para calcular el nuevo umbral. De todos modos, a medida que se van obteniendo los vectores, se van eliminando del conjunto temporal que los aloja (*ThresholdDFS*) aquellos que ya están dominados, al igual que ya se hacía en $PIDMOA^*$. Siendo *nodomset* una función que devuelve, para un conjunto de vectores T, el subconjunto de vectores de T no dominados entre sí, entonces el conjunto *ThresholdDFS* va evolucionando como sigue:

- Inicialmente $ThresholdDFS \leftarrow \varnothing$
- Tras procesar γ_1, $ThresholdDFS \leftarrow nodomset(\{(5,15)\} \cup ThresholdDFS) = nodomset(\{(5,15)\}) = \{(5,15)\}$
- Tras procesar γ_2, $ThresholdDFS \leftarrow nodomset(\{(5,12)\} \cup ThresholdDFS) = nodomset(\{(5,15),(5,12)\}) = \{(5,12)\}$
- Tras procesar n_3, $ThresholdDFS \leftarrow nodomset(\{(10,12)\} \cup ThresholdDFS) = nodomset(\{(10,12),(5,12)\}) = \{(5,12)\}$
- Tras procesar γ_3, $ThresholdDFS \leftarrow nodomset(\{(8,10)\} \cup ThresholdDFS) = nodomset(\{(5,12),(8,10)\}) = \{(5,12),(8,10)\}$

Tras finalizar la iteración, se calcula $\vec{threshold} = iPoint\{(5,12),(8,10)\} = (5,10)$.

La tercera y última iteración de $IPID^*$ se muestra en la figura 4.10b. Dado que el nuevo umbral, $(5,10)$, es estrictamente peor que el umbral anterior, $(2,5)$, tanto el estado inicial s como sus sucesores n_1y n_2 se generan y expanden al igual que lo hicieron en la iteración anterior.

En la rama izquierda, al explorar γ_1, como $\vec{f}(\gamma_1) = (5,15)$ no es estrictamente peor que el umbral $(5,10)$, al ser γ_1 un nodo final se incluye el par $(\gamma_1,(5,15))$ en el conjunto de soluciones. Al explorar el otro sucesor de n_1, γ_2, sucede lo mismo, ya que $\vec{f}(\gamma_2) = (5,12)$ no es estrictamente peor que el umbral $(5,10)$, con lo cual se añade también el par $(\gamma_2,(5,12))$ al conjunto $SOLUTION$.

En este punto del algoritmo hemos localizado dos soluciones, pero una de ellas, $(\gamma_2,(5,12))$, es Pareto óptima, mientras que la otra, $(\gamma_1,(5,15))$ es subóptima, ya que $(5,12) \prec (5,15)$. Es decir, se da la situación que ya comentamos en la sección anterior: el algoritmo $IPID^*$ puede añadir, aunque sólo sea de modo temporal, soluciones dominadas al conjunto de soluciones. Por ese motivo, cuando el algoritmo finaliza es

necesario realizar una comprobación para eliminar mediante tests de dominancia las soluciones dominadas. Esta comprobación puede realizarse tanto en el momento de localizar una nueva solución como al final de algoritmo. En el primer caso se realizan tests de dominancia adicionales cada vez que se localiza un nodo final, pero reduciendo los realizados cuando se expande un nodo para ver si está dominado por una solución ya localizada. En el segundo caso, la inserción de una solución es directa, a costa de incrementar los tests durante la expansión de los nodos. Esta opción es la que se ha seguido en la implementación actual de $IPID^*$.

Tras el *backtracking* correspondiente se procesa la rama derecha del árbol. El nodo n_2, ya expandido en la iteración anterior, se expande también en la actual. Se exploran sus nodos sucesores n_3 y γ_3. En el caso del nodo n_3, dado que su vector de coste $(10, 12)$ está dominado por una solución ya localizada (γ_2, con coste $(5, 12)$), esta rama se discontinúa y el nodo n_3 no se tiene en cuenta para el cálculo del siguiente umbral. Por otro lado, el nodo γ_3 tiene un vector de coste $\vec{f}(\gamma_3) = (8, 10)$, el cual no es estrictamente peor que el umbral actual $((5, 10))$. Por este motivo, al ser γ_3 una solución al problema, se añade el par $(\gamma_3, (8, 10))$ al conjunto de soluciones finales $SOLUTION$.

Al no haber más nodos pendientes de exploración, la iteración finaliza. Dado que no hay vectores susceptibles de ser usados para el cálculo del siguiente umbral, el algoritmo finaliza, devolviendo el conjunto de soluciones localizadas, previo filtrado con los tests de dominancia correspondientes. En el ejemplo que nos ocupa, $SOLUTION = nodomset(\{(5, 15), (5, 12), (8, 10)\}) = \{(5, 12), (8, 10)\}$.

Este sencillo ejemplo es resuelto por $PIDMOA^*$ en el mismo número de iteraciones y con el mismo número de nodos expandidos. Sin embargo, en las iteraciones 2 y 3, el umbral de $PIDMOA^*$ está formado por dos vectores. Esto hace que para cualquier nodo, antes de ser expandido, debe verificarse que no está dominado por ese par de vectores, duplicando el número de tests de dominancia. Como veremos en el Capítulo 8, en espacios de estados más grandes, el número de comparaciones de $PIDMOA^*$ se dispara, principalmente mientras no se localiza ninguna solución, ya que éstas contribuyen a purgar el umbral.

4.4.3. Utilidad de la relación *estrictamente mejor*

A continuación ilustraremos la necesidad de haber sustituido el test de dominancia contra el umbral durante la expansión de un nodo por el test *estrictamente mejor* definido en la sección 4.4.1. Esto se debe a que si en $IPID^*$ se realizara el test de dominancia normal, el algoritmo podría entrar en un bucle infinito, al calcularse una y otra vez el mismo vector umbral para todas las iteraciones.

Como ejemplo de ello, supongamos que usamos una versión de $IPID^*$ que discontinúa un camino si el coste del mismo está dominado por el vector umbral, de modo similar a como lo hace $PIDMOA^*$. Este algoritmo modificado lo aplicamos al problema representado por el grafo de la figura 4.11, en el cual los vectores heurísticos de

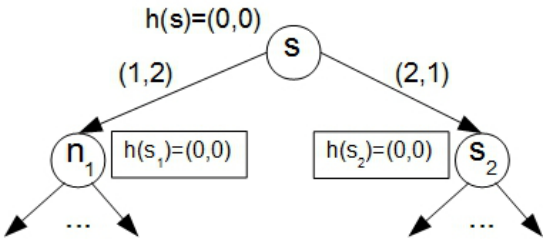


Figura 4.11: Ejemplo de bucle infinito evitado por IPID

todos los nodos son cero ($\forall n\, H(n) = \{\vec{0}\}$) y el nodo raíz s tiene dos nodos sucesores s_1 y s_2, con $\vec{f}(s_1) = (1,2)$ y $\vec{f}(s_2) = (2,1)$.

Inicialmente $IPID^*$ usa como umbral inicial $iPoint(H(s)) = (0,0)$. En la primera iteración, el vector de coste s no está dominado por ninguna solución. Dado que el umbral actual $(0,0)$ no domina al coste del camino actual $(0,0)$, entonces el nodo s se expande en modo *depth-first*. En lo que respecta al primer sucesor, s_1, $\vec{threshold} = (0,0) \prec (1,2) = \vec{f}(s_1)$, con lo cual la búsqueda a través de s_1 se discontinúa y su vector de coste se añade al nuevo conjunto $Threshold$. Lo mismo sucede con el sucesor s_2, ya que *threshold*$= (0,0) \prec (2,1) = \vec{f}(s_2)$.

El umbral de la siguiente iteración será $\vec{threshold} = iPoint\{(1,2),(2,1)\} = (1,1)$. Se realiza una nueva exploración del grafo y, aunque el umbral ha avanzado con respecto a la iteración anterior, el procesamiento de los sucesores de s tiene el mismo resultado. Tanto $\vec{f}(s_1) = (1,2)$ como $\vec{f}(s_2) = (2,1)$ están dominados por el umbral actual, el vector $(1,1)$, con lo cual la búsqueda se discontinúa en ambos nodos, sus vectores de coste se añaden al conjunto $Threshold$ de la siguiente iteración y el cómputo del punto ideal vuelve a dar como resultado el vector $(1,1)$. De este modo el algoritmo se vería atrapado en un bucle infinito.

Al sustituir en $IPID^*$ el chequeo de dominancia por el *estrictamente mejor*, el algoritmo se asegura de que el *threshold* de la iteración $i+1$ siempre será diferente del usado en la iteración i. En nuestro ejemplo anterior, ya que el vector $(1,1)$, umbral de la segunda iteración, no es *estrictamente mejor* que ninguno de los vectores de coste de s_1 y s_2, ambos nodos se expandirían. Si exigimos un coste mínimo positivo para cada objetivo (ver sección 7.1), tendremos asegurado que todas las componentes del vector de coste incrementan su valor, obteniendo a la postre umbrales estrictamente peores que los de iteraciones anteriores, y por lo tanto diferentes, lo cual resulta en una evolución del algoritmo hacia la frontera de Pareto.

4.5. Resumen

En este Capítulo se han mostrado nuevos algoritmos multiobjetivo de tipo *depth-first*: $PIDMOA^*$, $LEXIDMOA^*$ e $IPID^*$, los cuales abordan la generalización del

	Tipo de umbral	Cálculo del umbral	Retroceso
$PIDMOA^*$	Un conjunto de vectores por cada iteración	Conjunto de vectores no dominados de las ramas en las cuales se ha retrocedido	Si el coste del camino está dominado por alguno de los vectores del umbral
$LEXIDMOA^*$	Un vector por cada iteración	Menor lexicográfico de los vectores de coste de las ramas en las cuales se ha retrocedido	Si el coste del camino es peor lexicográfico que el umbral
$IPID^*$	Un vector por cada iteración	Punto Ideal de los vectores de coste de las ramas en las cuales se ha retrocedido	Si el coste del camino es estrictamente peor que el umbral

Tabla 4.4: Resumen de algoritmos multiobjetivo exactos *depth-first* basados en profundización iterativa

paradigma de profundización iterativa escalar de modo diferente al de un enfoque previo, $IDMOA^*$. En la tabla 4.4 se muestra un pequeño resumen de las características de estos algoritmos.

Aunque todos estos algoritmos están basados en IDA^*, $IDMOA^*$ no realiza una generalización del mismo, sino que se limita a realizar tantas profundizaciones iterativas como objetivos haya, es decir, ejecuta IDA^* una vez por cada objetivo contemplado. El hilo entre las sucesivas profundizaciones es la cota superior que éstas marcan para los objetivos no procesados.

Como consecuencia de ello, el rendimiento de $IDMOA^*$ se ve afectado por innecesarias reexpansiones, motivadas por el hecho de no procesar los vectores de coste como un todo, sino fijándose solo en una componente a la vez. Por el contrario, el primer algoritmo presentado en este Capítulo, $PIDMOA^*$, sí es una generalización *pura* de IDA^*, procesando toda la información de los vectores de coste en la misma profundización y manteniendo una cota multivectorial que avanza simultáneamente para todos los objetivos. Por este motivo realiza menos iteraciones y expande menos nodos.

Sin embargo, esta cota multivectorial incrementa el número de tests de dominancia a realizar por el algoritmo, influyendo negativamente en los tiempos de procesamiento. Sendas evoluciones de $PIDMOA^*$ ($LEXIDMOA^*$ e $IPID^*$), reducen el umbral multivectorial de $PIDMOA^*$ a un único vector, disminuyendo la cantidad de tests de dominancia a costa de realizar más reexpansiones. Paradójicamente, la búsqueda del mejor algoritmo multiobjetivo de profundización iterativa no deja de ser a su vez un problema multiobjetivo, en el cual tratamos de minimizar dos objetivos que, tal y como veremos en la evaluación presentada en el Capítulo 8, son antagónicos: el número

de reexpansiones de nodos y el número de tests de dominancia realizados.

En el siguiente Capítulo veremos sendas generalizaciones al caso multiobjetivo de otro algoritmo exacto *depth-first*, *RBFS*.

Capítulo 5

Nuevos enfoques multiobjetivo basados en *RBFS*

En el Capítulo 3 hacíamos referencia a *RBFS*, un algoritmo de búsqueda para un objetivo que expande los nodos en orden *best-first*, pero con un consumo lineal de memoria. Para ello, cuando *RBFS* expande un nodo n, guarda información acerca del mejor de los caminos que han quedado pendientes, bien a través de los hermanos de n, bien a través del resto del árbol explorado. Esta información, que denominamos cota superior, es la que limita la profundización por la rama actual. Si el coste asociado a esta rama excede la cota superior, esto es indicativo de que hay un camino *abandonado* (el asociado a la cota) que interesa recuperar, realizándose el correspondiente *backtracking* para reexpandirlo.

La aplicación de *RBFS* al caso multiobjetivo debe contemplar, obviamente, la naturaleza vectorial de los costes. Ya no tiene por qué haber un único coste asociado al mejor camino olvidado, como sucede en el caso escalar. En un árbol de búsqueda multiobjetivo pueden haberse abandonado caminos con diferentes costes, pero estos costes pueden ser no dominados entre sí, por lo que cualquiera de ellos es susceptible de ser reexpandido en caso de que el coste de la rama actual sea dominado por el de estos caminos olvidados. En la sección 3.6.3 se describía $MOMA^*0$, una aproximación ya existente de *RBFS* al caso multiobjetivo. Los bajos rendimientos obtenidos con una versión simplificada y optimizada de este algoritmo nos han llevado a considerar el desarrollo de nuevos algoritmos basados en *RBFS*.

En este Capítulo presentaremos sendas variantes multiobjetivo de *RBFS*: *Pareto-MO-RBFS* e *IP-MO-RBFS*, que emplean unas aproximaciones similares a las usadas por los algoritmos $PIDMOA^*$ e $IPID^*$ para la profundización progresiva. En la sección 5.1 realizamos una serie de consideraciones previas acerca de los algoritmos *RBFS* multiobjetivo. La sección 5.2 presenta *Pareto-MO-RBFS*, una variante multiobjetivo de *RBFS* que emplea toda la frontera no dominada del actual árbol de búsqueda como cota superior para la rama actualmente explorada. Esta cota multivectorial permitirá un avance rápido de la exploración, a costa de incrementar los tests de dominancia. En

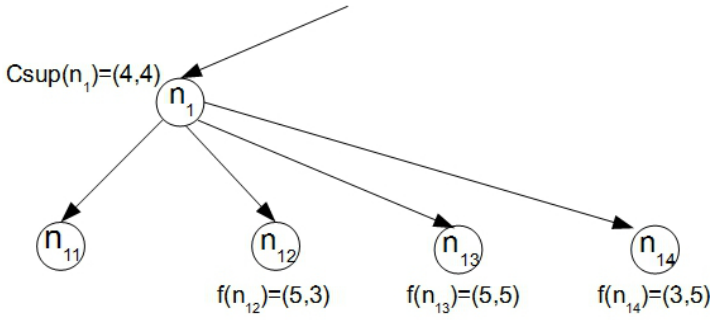


Figura 5.1: Cálculo de la cota superior en *RBFS* multiobjetivo.

la sección 5.3 presentamos *IP-MO-RBFS*, que basa el cómputo de la cota superior del algoritmo en el concepto de *Punto Ideal* visto en el Capítulo 4.

5.1. *RBFS* y la búsqueda multiobjetivo

La extensión de *RBFS* al caso multiobjetivo, tal y como se comentaba en la sección 3.6.3 implica un tratamiento diferente de la cota superior. Tal y como se puede apreciar en la figura 5.1, ya no existe un único coste mínimo de los caminos postergados como sucede en el caso escalar. En este ejemplo, si a través del nodo n_{11} alcanzamos un nodo con coste $(5, 5)$, este camino es susceptible de ser postergado (por lo menos temporalmente), al haber caminos mejores a través de n_{12} y n_{14}, pues sus vectores de coste ($(5, 3)$ y $(3, 5)$ respectivamente) dominan al coste del camino actual. Asimismo el nodo n_1 informa mediante su cota superior que en alguna otra rama del árbol existe también un camino postergado, con coste $(4, 4)$.

De igual modo que el tratamiento de la cota de profundización era el factor diferencial en los algoritmos multiobjetivo de profundización iterativa vistos en el capítulo 4, el tramiento de la cota superior, que informa de los caminos olvidados más prometedores, es el factor clave de los algoritmos que veremos en este capítulo. Mantener una cota superior que almacene información de todos los mejores caminos olvidados favorece el avance más rápido del algoritmo en cuanto a reconsideraciones de caminos a retomar. Sin embargo, el alto número de comprobaciones a realizar contra la cota, para decidir si descartar o no un camino, pueden penalizar el rendimiento temporal. En el caso de $MOMA^*0$ sus autores optaron por mantener la cota lo más sencilla posible, un único vector calculado como el mejor lexicográfico de todos los caminos discontinuados.

En este Capítulo consideramos dos variantes con un tratamiento opuesto de la cota superior de búsqueda: *Pareto-MO-RBFS*, que mantiene una cota multivectorial, e *IP-MO-RBFS*, que mantiene una cota formada por un único vector calculada como el punto ideal de los caminos olvidados.

Otra consideración a tener en cuenta en el rendimiento de los algoritmos es el man-

tenimiento de información relacionada con el coste de expansiones previas. Cuando en *RBFS* un nodo n se expande, emplea como coste el máximo de $f(n)$ y el coste informado por su padre, el cual será superior en caso de haberse expandido previamente esa misma rama. En la sección 2 se vio como el algoritmo trasladaba esa información por el árbol de búsqueda haciendo uso de una función de actualización. Aunque el algoritmo funciona correctamente sin esta cota, su rendimiento mejora notablemente, pues palía el balanceo de la búsqueda entre ramas cuando se exploran subárboles de búsqueda ya expandidos. Es necesario aplicar esta técnica a los algoritmos multiobjetivo, pues de lo contrario el impacto en el rendimiento será mucho mayor que en el caso escalar.

5.2. El algoritmo *Pareto-MO-RBFS*

Como ya se comentó en la sección anterior, la cota superior de un algoritmo *RBFS* fija la profundización máxima a realizar en la rama actual. Superar dicha cota equivale a realizar *backtracking* en el árbol de búsqueda para retomar algún camino discontinuado previamente, pero cuyo coste es ahora mejor que el del camino actual.

El algoritmo *Pareto-MO-RBFS*, al expandir un nodo del árbol, le asigna como cota superior el conjunto de vectores de coste no dominados asociados a todos los caminos que han sido temporalmente discontinuados en otras ramas. Tal y como apreciamos en la figura 5.2, al expandir el nodo n_{11}, su cota superior está formada por los costes no dominados asociados a padre y hermanos. Esta cota superior se propagará también a los descendientes de n_{11}, de tal modo que, cuando a través de ese subárbol se alcance un coste que esté dominado por alguno de los vectores reflejados en la cota, el algoritmo intentará retomar el camino asociado a dicho vector mediante los necesarios *backtrackings* y reexpansiones.

Por lo tanto, en todo momento *Pareto-MO-RBFS* mantiene la frontera de Pareto de todos los caminos discontinuados durante la búsqueda. De modo similar a su contrapartida de profundización iterativa, $PIDMOA^*$, al mantener una cota con información completa, *Pareto-MO-RBFS* sólo localiza soluciones dominadas, nunca soluciones subóptimas. Esto hace que, cuando localiza una nueva solución, no tenga que compararla con las encontradas previamente por si fuera necesario descartar alguna que estuviera dominada. Esta característica de *Pareto-MO-RBFS* se demostrará formalmente en el Capítulo 7.

El otro aspecto a destacar de este algoritmo es el uso de una función de actualización similar a la usada por *RBFS*, pero de tipo vectorial, que realizará la propagación de costes actualizados en ambos sentidos, tanto hacia los descendientes cuando se expande una rama como hacia los ascendientes cuando se produce un *backtracking*. Este flujo de información permitirá saber cuándo una rama ya ha sido previamente expandida, propagando hacia los nodos hijo los costes vectoriales alcanzados en las diferentes hojas de la expansión previa. Con esto se consigue una importante reducción del número de reexpansiones.

El funcionamiento básico del algoritmo se puede resumir en los siguientes puntos:

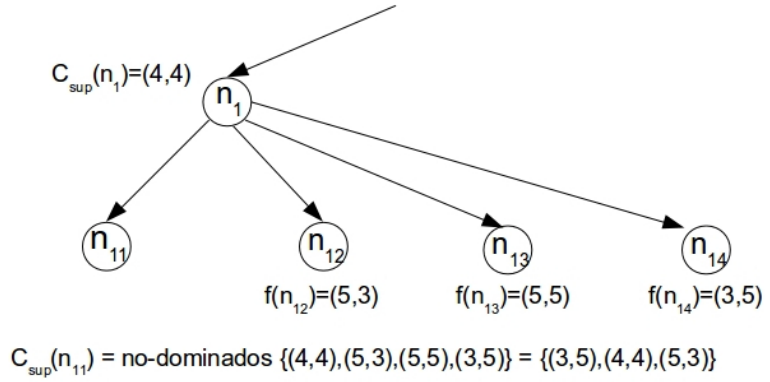


Figura 5.2: Cálculo de la cota superior en *Pareto-MO-RBFS*.

- La búsqueda se discontinúa en una rama del árbol de modo definitivo únicamente por los siguientes motivos:
  - cuando el camino tiene costes dominados por soluciones localizadas previamente, ó
  - cuando el camino termina en un nodo sin sucesores que no es solución.
- La cota superior de un nodo refleja el coste de los caminos más prometedores de las otras ramas del árbol explorado a partir de sus antecesores. Cuando un nodo está dominado por su cota superior, se realiza *backtracking* para intentar retomar alguno de esos caminos.
- Al realizar *backtracking*, un nodo informa a sus antecesores del coste alcanzado en su exploración. De este modo, si el camino pudiera ser de interés para posibles reexpansiones futuras (por no estar todavía dominado por ninguna solución), se empleará para computar cotas de expansión (superiores) en otras ramas del árbol.

En la siguiente sección se realiza una descripción detallada del algoritmo *Pareto-MO-RBFS*.

5.2.1. Descripción del algoritmo

En la tabla 5.1 se muestra el pseudocódigo del algoritmo *Pareto-MO-RBFS*. Definimos el operador $\preceq_I$ sobre dos vectores, $\vec{a} \preceq_I \vec{b}$, como $(\vec{a} \preceq \vec{b}) \vee (\vec{a} \sim \vec{b})$[1]. Los parámetros de la función base del algoritmo, *Pareto-MO-RBFS* son:

- Nodo a procesar.

[1]Por simplicidad, y abusando un poco de la notación, aplicaremos las operaciones habituales sobre vectores ($\sim$, $\prec$, $\preceq$, $\ll$, $\not\ll$, $\preceq_I$) también a conjuntos, de tal modo que A *op* B denota $\forall \vec{a} \in A, \forall \vec{b} \in B, \vec{a}$ *op* $\vec{b}$

- Valor de actualización del coste del nodo, que representa la información obtenida de posibles expansiones previas del mismo.
- Cota superior del nodo, que representa los costes de los mejores caminos *olvidados* momentáneamente por presumirse más prometedora la rama actual.
- Conjunto temporal de soluciones encontradas.

En la primera llamada, el algoritmo recibe como argumentos la siguiente información:

- El nodo inicial s.
- La estimación de coste para dicho nodo. Al no tener s predecesores, la única información disponible la proporciona la función heurística.
- El conjunto $\{\vec{\infty}\}$ como cota superior. Dado que no ha habido todavía ninguna expansión del árbol de búsqueda, no hay información de la calidad de los caminos. Con esta cota superior el algoritmo indica que está dispuesto a aceptar cualquier camino, independientemente de su coste.
- Un conjunto de soluciones iniciales vacío.

El algoritmo devuelve el conjunto SOL, en el cual *Pareto-MO-RBFS* va almacenando las soluciones no dominadas del problema que va localizando, siendo cada solución un par $(\gamma, \vec{c^*})$, donde γ es el nodo objetivo localizado y $\vec{c^*}$ es el vector de coste del camino seguido para alcanzar dicho nodo. Por simplificación se denota C_{sol} como el conjunto de vectores de coste de las soluciones óptimas localizadas, es decir, $C_{sol} = \{\vec{c^*} \mid (\gamma, \vec{c^*}) \in SOL\}$.

Antes de expandir un nodo n para generar sus sucesores, es necesario comprobar si el nodo es de interés para el proceso de búsqueda. Por ello se comparan sus vectores de coste con las soluciones ya encontradas (SOL) y con la cota superior (C_{sup}). Si cada vector de coste $\vec{f}(n)$ está dominado por alguna solución, el camino actual no es de interés y debe ser descartado, tanto en la expansión actual como en futuras reexpansiones. Cuando se discontinúa la exploración en un nodo, éste debe informar del coste de los mejores caminos encontrados a través del mismo. Por lo tanto, para autodescartarse del proceso de búsqueda, el nodo informa con un vector de coste $\vec{\infty}$. Esta situación se refleja en la figura 5.3a, donde el nodo n_{13} tiene un coste $(10, 10)$, habiendo ya una solución con coste $(7, 7)$ que la domina.

De modo análogo, si todos los vectores de coste del nodo no dominados por ninguna solución están dominados por algún elemento de la cota superior, entonces la exploración debe discontinuarse, pero en este caso devolviéndose dicho conjunto de vectores de tal modo que sean usados como posible cota superior en otras ramas del árbol, por si interesa reconsiderarlos en exploraciones posteriores.

Si por el contrario, el nodo n es susceptible de ser explorado en la llamada actual, por tener algún vector de coste no dominado por las soluciones encontradas ni por la

Pareto-MO-RBFS-base ()

$SOL \leftarrow \emptyset$
(FA, SOL) $\leftarrow$ Pareto-MO-RBFS (s, nodomset($H(s)$), $\{\vec{\infty}\}$, SOL)
return (SOL)

Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL)

$N_{Sol} \leftarrow \{\vec{f} \in F(n) \mid \nexists \vec{c^*} \in C_{sol}, \vec{c^*} \prec \vec{f}\}$
IF ($N_{Sol} = \emptyset$)
    return ($\{\vec{\infty}\}$, SOL)
$N_{C_{sup}} \leftarrow \{\vec{f} \in N_{Sol} \mid \nexists \vec{v} \in C_{sup}, \vec{v} \prec \vec{f}\}$
IF ($N_{Csup} = \emptyset$)
    return (N_{Sol}, SOL)
IF ($n \in \Gamma$)
    $SOL \leftarrow SOL \cup \{(n, \vec{f}(n))\}$
    return ($\{\vec{\infty}\}$, SOL)
IF (Sucessors(n) = $\emptyset$)
    return ($\{\vec{\infty}\}$, SOL)
IF ($FA_n \preceq_I$ nodomset(F(n)))
    For each $n_i \in$ Sucessors(n)
        FA[i] $\leftarrow$ nodomset(F(n_i))
ELSE
    For each $n_i \in$ Sucessors(n)
        FA[i] $\leftarrow$ nodomset ($\bigcup_{\vec{f} \in nodomset(F(n_i))} MaxVector(\vec{f}, FA_n)$)
Discontinued $\leftarrow \emptyset$
LOOP
    IF ($\bigcup_i$FA[i] = $\emptyset$)
        IF (Discontinued = $\emptyset$) return ($\{\vec{\infty}\}$, SOL)
        ELSE
            Disc2 = $\{\vec{v} \in$ Discontinued $\mid \nexists \vec{c^*} \in C_{sol}, \vec{c^*} \prec \vec{v}\}$
            return (nodomset(Disc2), SOL)
    $N_{suc} \leftarrow$ nodomset($\bigcup_i$FA[i])
    $N1_s \leftarrow \{\vec{f} \in FA[1] \mid \nexists \vec{c^*} \in C_{sol}, \vec{c^*} \prec \vec{v}\}$
    $N1_{sc} \leftarrow \{\vec{f} \in N1_s \mid \nexists \vec{c} \in C_{sup}, \vec{c} \prec \vec{f}\}$
    IF ($N1_{sc} = \emptyset$)
        Remove FA[1]
        IF ($N1_s \neq \emptyset$)
            Discontinued $\leftarrow$ Discontinued $\cup$ $N1_s$
    ELSE IF (($N1_{sc} \cap N_{suc}$) $\neq \emptyset$)
        (FA[1], SOL) $\leftarrow$ Pareto-MO-RBFS (n_1, $N1_{sc}$,
                nodomset($C_{sup} \cup (\bigcup_{i \neq 1} FA[i])$), SOL)
    ELSE
        Move FA[1] to the end of FA[]
END LOOP

Tabla 5.1: Algoritmo *Pareto-MO-RBFS*

$MaxVector$ **(**$\vec{v}$**,** $VectorSet$**)**

```
Maxv ← ∅
For each v⃗s ∈ VectorSet
      m⃗axv ← 0⃗
      For i = 1  to  ♯objectives
            m⃗axv[i] = máx{v⃗[i], v⃗s[i]}
      Maxv ← Maxv ∪{m⃗axv}
return (Maxv)
```

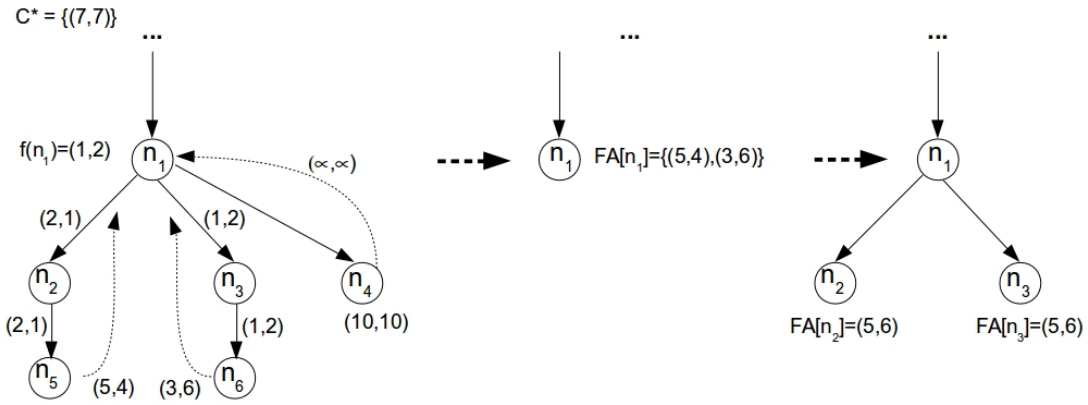
Tabla 5.2: Función $MaxVector$.

cota superior, entonces se comprueba en primer lugar si es un nodo meta. En caso afirmativo, el nodo se incorpora al conjunto SOL sin ningún tipo de test adicional, ya que, como se demostrará en el Capítulo 7, *Pareto-MO-RBFS* únicamente localiza soluciones óptimas, nunca subóptimas.

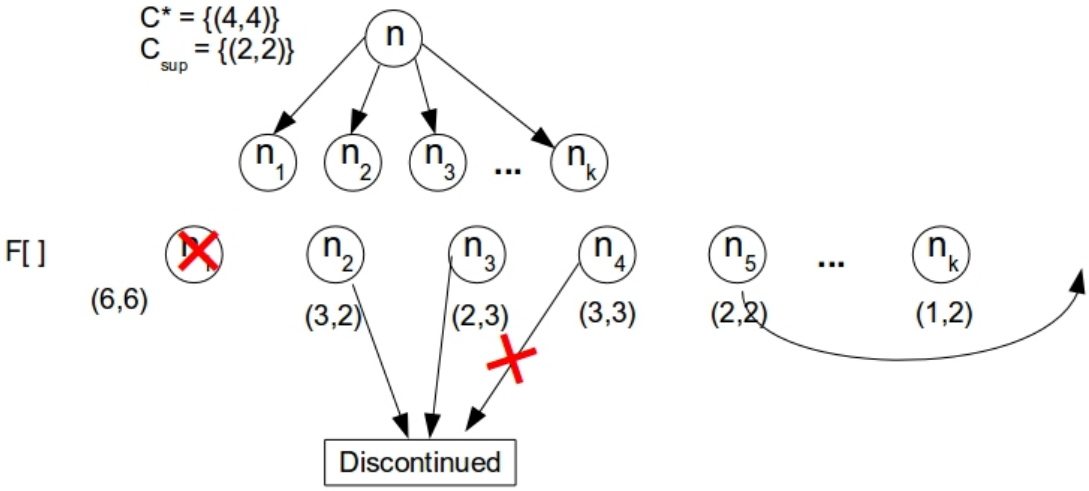
En caso de que el nodo n no sea meta, se procede a expandirlo. Si no tuviera hijos, el nodo se autodescarta para futuras expansiones, devolviendo como coste $\vec{\infty}$, de modo similar a cuando era dominado totalmente por soluciones ya encontradas.

Si el nodo n tiene sucesores, genera una lista ($FA[]$), con los costes actualizados por expansiones previas. Esta actualización de costes se realiza a través de dicha lista, y su funcionamiento se ilustra en la figura 5.3a. Cuando el nodo n_1 se expande por primera vez, su coste asociado es el devuelto por su función de coste, que en este caso devuelve el vector $(1, 2)$. Tras generar su subárbol de búsqueda correspondiente hasta la profundización que marque la cota superior, los nodos discontinuados (en este caso n_5, n_6 y n_3) devuelven sus vectores de coste ($(5, 4)$, $(3, 6)$ y (∞, ∞) respectivamente) hacia sus antecesores. De este modo el nodo n_1 actualiza su función de coste con los vectores no dominados de los caminos discontinuados.

Si en algún momento es necesario realizar *backtracking* para poder explorar otras ramas más prometedoras, el nodo n_1 informa a sus antecesores de sus nuevos vectores de coste, al igual que hicieron sus descendientes con él. En la siguiente reexpansión del nodo, n_1 recibirá estos mismos vectores de coste ($\{(5, 4), (3, 6)\}$) como valor de actualización, trasladándolos a sus sucesores n_2 y n_3. Estos nodos actualizan sus vectores de coste aplicando la función $MaxVector$ (tabla 5.2), tal y como se puede apreciar en la parte derecha de la figura 5.3a. La función $MaxVector$ recibe un vector de coste $\vec{f}(n)$ de un nodo y el posible coste actualizado del mismo FA_n a través del padre, y devuelve un conjunto de vectores formado por los valores máximos de cada componente en cada par $(\vec{f}(n), \vec{fa})$, donde $\vec{fa}$ es un vector de dicho conjunto FA_n. De este modo, por ejemplo, el nodo n_2 se expandirá directamente, sin *backtrackings*, mientras por la rama no se encuentre un nodo cuyo coste sea dominado por uno de los vectores $(5, 4)$ ó $(3, 6)$. Esta actualización de los costes de los nodos mediante la función $MaxVector$



(a) Actualización de costes mediante el valor FA_n



(b) Gestión del conjunto *FA*

Figura 5.3: Comportamiento de *Pareto-MO-RBFS*.

únicamente tiene sentido cuando ha habido una expansión previa. En caso contrario, los vectores de coste se mantienen.

La lista FA que contiene los sucesores no está ordenada con ningún criterio y su funcionamiento básico, ilustrado en la figura 5.3b, es el siguiente:

- Cada elemento de la lista está asociado a un sucesor del nodo actual e inicialmente contiene sus vectores de coste (posiblemente *actualizados* mediante la función *MaxVector* empleando la información de FA_n, tal y como se ha explicado anteriormente).
- Siempre se procesan los vectores almacenados en el primer elemento de la lista.
- Cuando los vectores de coste de un sucesor están dominados por soluciones ya encontradas, el sucesor se elimina de la lista FA. Este es el caso del nodo n_1 en la figura 5.3b, cuyo coste $(6, 6)$ está dominado por la solución de coste $(4, 4)$.

- Cuando los vectores de coste de un sucesor están dominados por la cota superior (es el caso de los nodos n_2, n_3 y n_4 en la figura 5.3b) se elimina el nodo de FA, añadiendo sus vectores de coste a la variable *Discontinued*. Este conjunto almacena los costes no dominados de los caminos discontinuados, pero no descartados definitivamente. En nuestro caso el nodo n informa con su cota superior $(2, 2)$ que hay un camino de más interés que el que ofrecen n_2 y n_3. En el caso de n_4, su vector de coste no se tiene en cuenta al estar dominado por los costes de n_2 y n_3.

- Si un sucesor está dominado por otros sucesores, entonces se pasa al final de la lista FA para ser procesado en último lugar, desplazando el resto de elementos en orden a las primeras posiciones de dicha lista. Es decir, se da preferencia a los sucesores con costes no dominados frente a los dominados, evitando así encontrar soluciones dominadas. Es el caso del nodo n_5 en la figura 5.3b, el cual, aún siendo candidato a ser explorado, ya que no está dominado ni por soluciones previas ni por la cota superior de n, ofrece un camino de *peor calidad* que n_k, motivo por el cual se promueve el procesamiento de este último nodo.

- En cualquier otro caso, el nodo se expande usando su coste (posiblemente actualizado), y tomando como cota superior los vectores no dominados de los caminos pendientes (ver ejemplo de la figura 5.2).

- Cuando ya no hay nodos pendientes de procesar en la lista FA, entonces se devuelve hacia el nodo n la información de los caminos discontinuados que puedan servir de interés en reexpansiones futuras. Esta información se almacena en el conjunto *Discontinued*. Si no hubiera nodos en esta situación, porque fueran soluciones ya encontradas o porque su coste exceda el de soluciones ya encontradas, entonces el nodo n recibe como actualización de su coste el vector $\vec{\infty}$ para que se autodescarte en la búsqueda.

En la sección 5.2.2 presentamos la resolución detallada de un problema multiobjetivo con el algoritmo *Pareto-MO-RBFS*.

5.2.2. Ejemplo de *Pareto-MO-RBFS*

Una vez visto el funcionamiento básico del algoritmo *Pareto-MO-RBFS*, a continuación resolveremos un sencillo problema con dos objetivos. El grafo asociado a este problema se muestra en la figura 5.4. En cada arco se refleja el coste asociado a la transición de un nodo a otro. Asimismo la estimación de coste para cada nodo es nula, viene dada por el vector $(0, 0)$, el cual es un heurístico trivialmente admisible.

El problema tiene cuatro soluciones, representadas por los nodos γ_1, γ_2, γ_3 y γ_4. De ellas, únicamente los nodos γ_1 y γ_4 son óptimos de Pareto. El coste de γ_2, $(5, 5)$, está dominado por el coste de γ_4, $(4, 5)$, y el coste de γ_3, $(3, 8)$, está dominado por el coste de γ_1, $(3, 6)$.

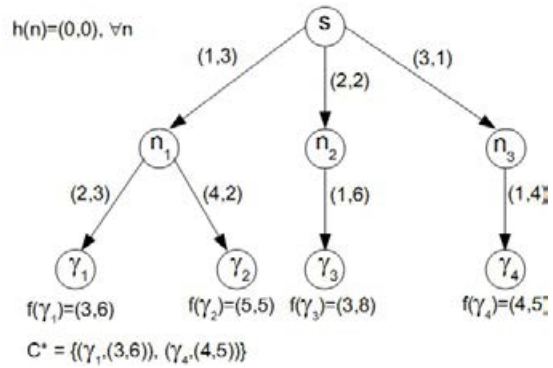


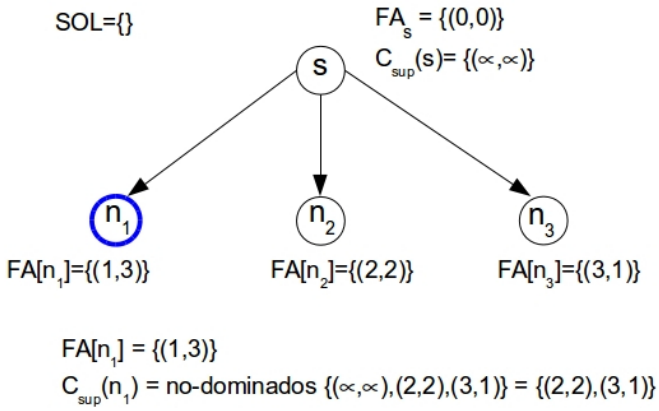
Figura 5.4: Problema de ejemplo para *Pareto-MO-RBFS*.

En la figura 5.5a se muestra la situación inicial del árbol de búsqueda. El valor *actualizado* del nodo raíz s, FA_s, es $\vec{h}(n)$. La cota superior, formada por el vector (∞, ∞), refleja que, hipotéticamente, no hay ningún camino fuera de esta rama que sea de interés, lo cual es obvio al no haber nodos por encima de s en el árbol de búsqueda.

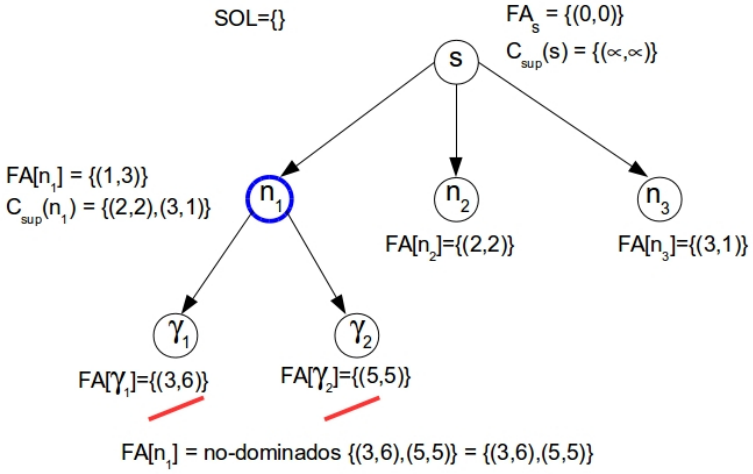
Tras expandir el nodo s, se generan y procesan sus sucesores sin un orden particular. A efectos de este ejemplo, supondremos que la expansión se realiza de izquierda a derecha. Dado que no se ha encontrado todavía ninguna solución y que el nodo n_1 es más prometedor que lo que *recuerda* el nodo s (su cota superior (∞, ∞)), el nodo se expande. Como valor de actualización se usa su vector de coste al no haberse realizado ninguna expansión previa, y como cota superior se usa el conjunto de vectores de coste no dominados de los caminos susceptibles de ser reexpandidos, bien a través de su padre s (no hay ninguno, al ser nodo raíz), bien a través de sus hermanos n_2 y n_3.

En la figura 5.5b se muestra la expansión del nodo n_1. El coste de sus sucesores tampoco se actualiza con la función $MaxVector$, al ser la primera expansión de dicho nodo. Ambos sucesores, γ_1 y γ_2, no están dominados por ninguna solución previa, ya que el algoritmo todavía no ha localizado ninguna. Sin embargo sí están dominados por la cota superior de n_1. Esto indica que, aunque estos caminos no se descartan definitivamente, sí es necesario explorar primero otros descartados anteriormente (a través de n_2 y n_3). La función de coste de n_1, almacenada en el conjunto FA, se actualiza al conjunto de vectores de coste no dominados y discontinuados a través de n_1. Es decir, $FA[n_1]$ muestra que a través de dicho nodo se ha profundizado hasta dos caminos de coste $(3, 6)$ y $(5, 5)$, y esa información la propaga hacia la raíz del árbol, por si algún ascendiente de n_1 considera en algún momento que interesa la reexpansión.

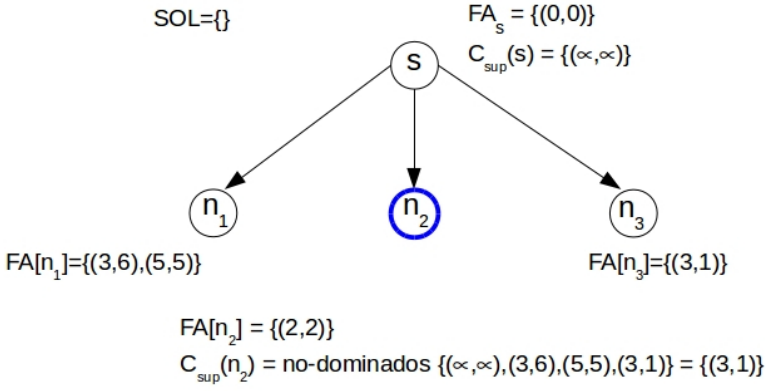
Tras realizar el *backtracking* sobre el nodo n_1, al tener éste ahora un coste (actualizado) dominado por los nodos n_2 y n_3, se pasa al final de la lista FA para ser evaluado posteriormente. El siguiente nodo a explorar, tal y como se muestra en la figura 5.5c, es el nodo n_2. Al igual que en el caso de n_1, n_2 se expande, pues no está dominado por ninguna solución ni por la cota superior de n. Al expandirlo, su valor de actuali-



(a) Paso 1

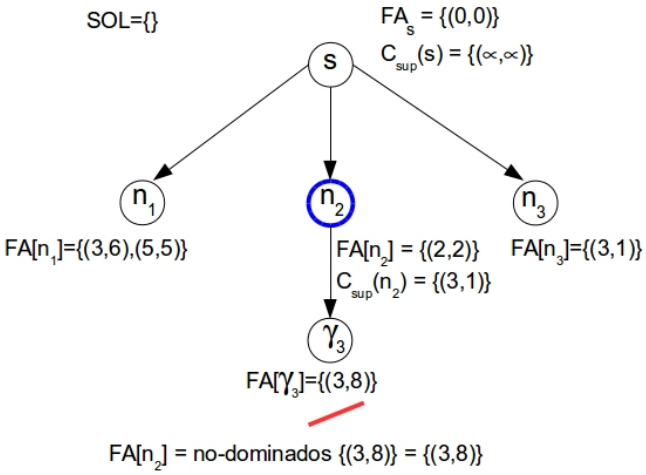


(b) Paso 2

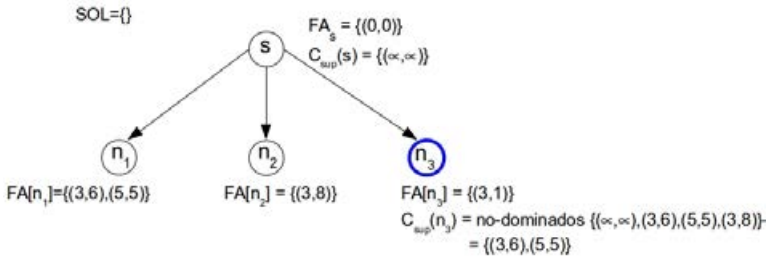


(c) Paso 3

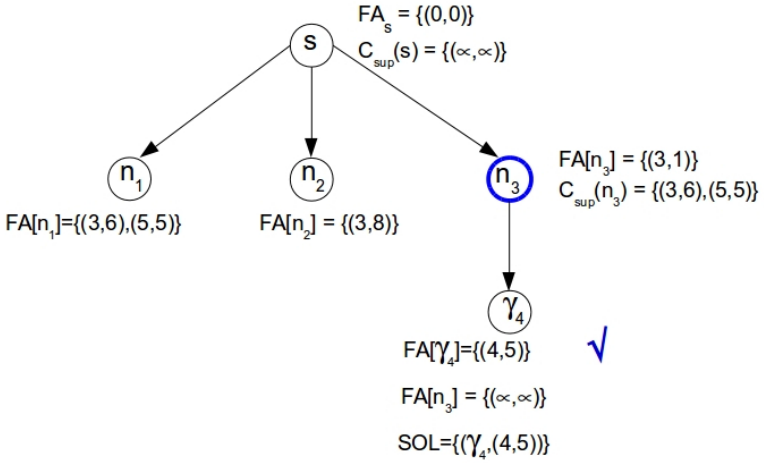
Figura 5.5: Resolución mediante *Pareto-MO-RBFS* (1).



(a) Paso 4

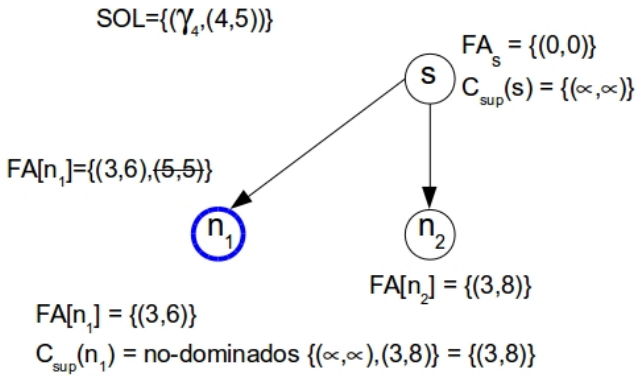


(b) Paso 5

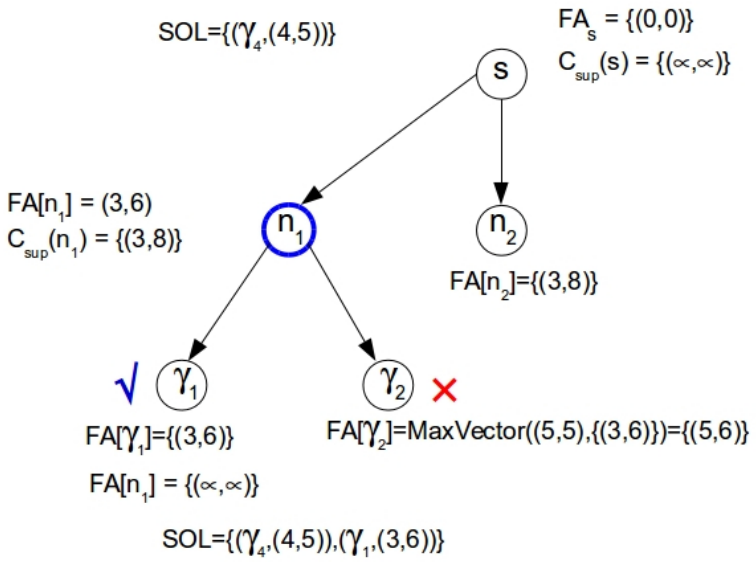


(c) Paso 6

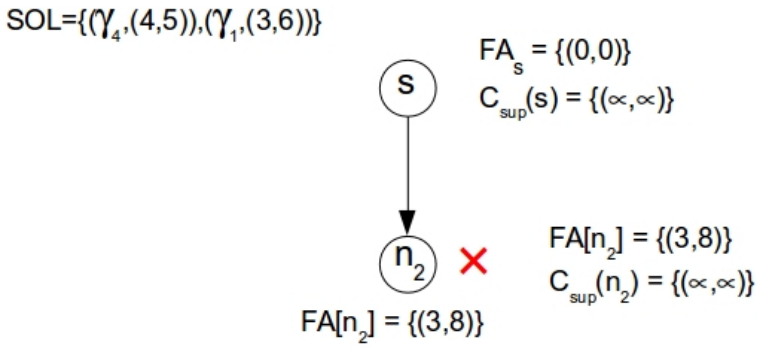
Figura 5.6: Resolución mediante *Pareto-MO-RBFS* (2).



(a) Paso 7



(b) Paso 8



(c) Paso 9

Figura 5.7: Resolución mediante *Pareto-MO-RBFS* (3).

zación coincide con su función de coste (pues no ha sido expandido previamente) y su cota superior refleja los mejores caminos pendientes. Dado que los caminos a través de n_1 son peores (dominados) que el que refleja n_3, la cota superior de n_2 *recuerda* este último camino, de coste $(3, 1)$.

La figura 5.6a muestra la expansión del nodo n_2. Su sucesor, γ_3, con coste $(3, 8)$, ofrece un camino de coste peor que el recordado por n_2, aunque no sea descartable por no haber todavía soluciones localizadas. El vector de coste de n_2 se actualiza convenientemente, con el coste de este camino discontinuado.

Al igual que sucedio con n_1, al ofrecer ahora n_2 con su coste actualizado un camino peor (dominado) que el alcanzable a través de n_3, se pospone el procesamiento de n_2, pasándolo al final de la lista FA usada en el algoritmo. En la figura 5.6b se muestra el tratamiento de n_3. De nuevo el nodo se expande, recordando en su cota superior los mejores caminos todavía disponibles, pero ya con los costes actualizados. En este caso las mejores opciones son las soluciones alcanzables a través de n_1, pues una de ellas, $(3, 6)$, domina a la opción que ofrece n_2, $(3, 8)$.

En la figura 5.6c se muestra la expansión del nodo n_3. Su sucesor, γ_4, ofrece un camino de coste $(4, 5)$, el cual no está dominado por la cota superior de n_3. Por ello se selecciona este nodo para expansión. Dado que es un nodo final, se añade al conjunto de soluciones ya encontradas, SOL. Al no haber más opciones de expansión, el nodo γ_4 actualiza el coste de su padre, n_3, con el vector (∞, ∞), evitando futuras exploraciones de este subárbol explorado durante la búsqueda.

Al actualizar el vector de coste de n_3 al valor (∞, ∞), este nodo ya estaría dominado por el coste de la primera solución encontrada, con lo cual se descarta de la exploración, tal y como se refleja en la figura 5.7a. Ahora contemplamos la reexpansión del nodo n_1, pues uno de los caminos que recuerda, el representado por el vector $(3, 6)$ tiene todavía interés para la búsqueda. El segundo camino, de coste $(5, 5)$, no se contempla, al estar dominado por la solución γ_4. Al expandir n_1 usamos su vector de coste actualizado, $(3, 6)$, y como cota superior el coste del camino pendiente a través de n_2, pues tampoco está dominado por ninguna solución.

La reexpansión de n_1 se muestra en la figura 5.7b. El nodo γ_1 no está dominado por ninguna solución previa, y además es más interesante que el recordado por la cota superior de n_1, pues lo domina. Es por ello que γ_1 se expande y, al ser nodo solución, se añade al conjunto SOL. Al expandir γ_2, con su vector de coste actualizado, dado que está dominado por una solución ya encontrada, γ_4, el nodo se descarta. Ambos sucesores, γ_1 y γ_2, devuelven como coste actualizado el vector (∞, ∞): en un caso, γ_1, por haberse localizado una solución óptima y en otro caso, γ_2, por haber podado un camino peor que una solución ya existente. De cualquier modo, ambos caminos no deberían ser reexpandidos.

Dado que todos los caminos a través de n_1 se descartan definitivamente, su vector de coste toma el valor (∞, ∞) y por lo tanto este nodo también se elimina. En la figura 5.7c se muestra el procesamiento del único nodo pendiente, n_2. La cota superior refleja los mejores caminos pendientes de exploración. En este caso, ya que los caminos

a través de n_1 y n_3 ya se han discontinuado definitivamente, esta cota toma el valor (∞, ∞). De todos modos, al estar n_2 dominado por una solución ya encontrada, γ_1, el nodo se descarta, finalizando el proceso de búsqueda. Por lo tanto el algoritmo $Pareto\text{-}MO\text{-}RBFS$ finaliza, devolviendo el conjunto de soluciones óptimas.

5.3. El algoritmo $IP\text{-}MO\text{-}RBFS$

En la sección 5.2 se analizaba $Pareto\text{-}MO\text{-}RBFS$, una variante de $RBFS$ que emplea la información de todos los caminos pendientes no dominados como cota superior de expansión. Esta información, generalmente multivectorial, debe ser comparada con vectores de coste de los caminos explorados, para determinar si estos caminos son susceptibles de seguir siendo explorados o si, por el contrario, se discontinúan. Estos chequeos, que en el caso de $Pareto\text{-}MO\text{-}RBFS$ son tests de dominancia, son un factor determinante en el rendimiento de cualquier algoritmo multiobjetivo. Es por ello que su reducción puede conllevar una mejora importante en las prestaciones.

En los algoritmos de profundización iterativa mostrados en el Capítulo 4, uno de ellos, $PIDMOA^*$, mantenía también un conjunto de vectores que representaba la frontera de Pareto asociada a los nodos discontinuados, de modo similar a como funciona $Pareto\text{-}MO\text{-}RBFS$. En el caso de $PIDMOA^*$ la frontera (denominada umbral) se calcula para cada iteración y en el caso de $Pareto\text{-}MO\text{-}RBFS$, la frontera (denominada cota superior) se calcula para cada nodo.

Para disminuir el número de tests de dominancia realizados contra esta frontera, en el Capítulo 4 presentábamos $LEXIDMOA^*$ e $IPID^*$, sendos algoritmos que reducían el umbral a un único vector, calculado como el mejor lexicográfico y el punto ideal, respectivamente, de la frontera de Pareto. En esta sección proponemos el uso de un planteamiento similar en la generalización de $RBFS$ al caso multiobjetivo. Dado que el uso del mejor lexicográfico ya ha sido abordado en el algoritmo $MOMA^*0$ (ver sección 3.6.3), aplicaremos el concepto de *Punto Ideal* al cómputo de la cota, dando lugar al algoritmo $IP\text{-}MO\text{-}RBFS$.

El funcionamiento de $IP\text{-}MO\text{-}RBFS$ es similar al de $Pareto\text{-}MO\text{-}RBFS$, con la excepción de que la cota superior del primero es el punto ideal de la cota superior del segundo. En la figura 5.8 se puede ver un sencillo ejemplo que ilustra este comportamiento. Mientras que la cota superior calculada por $Pareto\text{-}MO\text{-}RBFS$ sería el conjunto de vectores no dominados $\{(4,4),(5,3),(3,5)\}$, $IP\text{-}MO\text{-}RBFS$ aplica el operador punto ideal a este conjunto, resultando en una cota formada por un único vector, $(3,3)$. Además los tests de dominancia se sustituyen por tests del tipo *estrictamente mejor*, excepto cuando se compara el coste de un nodo con el de las soluciones ya localizadas.

En la siguiente sección se realiza una descripción detallada del algoritmo $IP\text{-}MO\text{-}RBFS$.

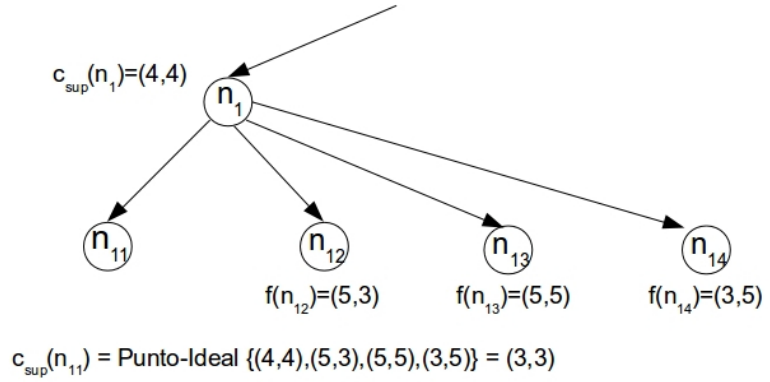


Figura 5.8: Cálculo de la cota superior en $IP\text{-}MO\text{-}RBFS$

5.3.1. Descripción del algoritmo

En la tabla 5.3 se muestra el pseudocódigo del algoritmo $IP\text{-}MO\text{-}RBFS$. Tanto los parámetros de la función base del algoritmo, $IP\text{-}MO\text{-}RBFS$, como sus valores iniciales son idénticos a los empleados en el algoritmo $Pareto\text{-}MO\text{-}RBFS$. Debido a la gran similitud entre $Pareto\text{-}MO\text{-}RBFS$ e $IP\text{-}MO\text{-}RBFS$, en esta sección únicamente haremos hincapié en las principales diferencias entre ambos algoritmos. Los conjuntos SOL y C_{sol} son los mismos descritos para $Pareto\text{-}MO\text{-}RBFS$ en la sección 5.2.1.

Al igual que en $Pareto\text{-}MO\text{-}RBFS$, $IP\text{-}MO\text{-}RBFS$ descarta definitivamente un nodo, devolviendo como coste el vector $\vec{\infty}$, cuando este nodo está dominado por alguna solución. En caso contrario, se comprueba si el nodo es solución. Un aspecto a tener en cuenta en $IP\text{-}MO\text{-}RBFS$ es que este algoritmo puede localizar soluciones subóptimas durante el proceso de búsqueda, las cuales serán incluídas en el conjunto SOL. Es por ello que cuando se localiza un nuevo nodo solución, se debe comprobar:

- si la nueva solución encontrada es subóptima, en cuyo caso se descarta.
- si la nueva solución encontrada domina a alguna de las localizadas previamente e incluídas en el conjunto SOL. En este caso se procede a purgar este conjunto, eliminando todos los vectores de coste dominados.

Veremos ejemplos de ambas situaciones en la sección 5.3.2.

Si el nodo de trabajo no es un nodo solución, se expande. Al igual que en $Pareto\text{-}MO\text{-}RBFS$, si no tiene hijos, devuelve como coste $\vec{\infty}$, para evitar posteriores exploraciones de esta rama.

Si el nodo n tiene sucesores, genera una lista (FA), con los costes actualizados. El cálculo del valor de actualización se ilustra en la figura 5.9a. La primera vez que se expande un nodo n, su coste asociado es $\vec{f}(n)$. En nuestro ejemplo, el coste inicial del nodo n_1 es $(1, 2)$. Tras generar un subárbol de búsqueda hasta donde indique la cota superior, los nodos discontinuados (en este caso n_5, n_6 y n_4) devuelven sus vectores

```
IP-MO-RBFS-base ()
     SOL ← ∅
     (FA, SOL) ← IP-MO-RBFS (s, nodomset(H(s)), ⃗∞, SOL)
     return (SOL)


IP-MO-RBFS (n, FA_n, c⃗_sup, SOL)

N_Sol ← {f⃗ ∈ F(n) | ∄c⃗* ∈ C_sol, c⃗* ≺ f⃗}
IF (N_Sol = ∅)
     return (⃗∞, SOL)
N_csup ← {f⃗ ∈ N_Sol | csup⃗ ≪ f⃗}
IF (N_csup = ∅)
     return (N_Sol, SOL)
IF (n ∈ Γ)
     return (⃗∞, nodomset (SOL ∪ {(n, f⃗(n))}))
IF (Sucessors(n) = ∅)
     return (⃗∞, SOL)
IF (FA_n ⪯∼ nodomset(F(n)))
     For each n_i ∈ Sucessors(n)
          FA[i] ← nodomset(F(n_i))
ELSE
     For each n_i ∈ Sucessors(n)
          FA[i] ← nodomset (⋃_{f⃗∈nodomset(F(n_i))} MaxVector(f⃗, FA_n))
Discontinued ← ∅
LOOP
     IF (⋃_i FA[i] = ∅)
          IF (Discontinued = ∅) return (⃗∞, SOL)
          ELSE
               Disc2 = {v⃗ ∈ Discontinued | ∄(γ, c⃗) ∈ SOL, c⃗ ≺ v⃗}
               return (iPoint(Disc2), SOL)
     ip⃗_suc ← iPoint(⋃_i FA[i])
     N1_s ← {f⃗ ∈ FA[1] | ∄(γ, c⃗) ∈ SOL, c⃗ ≺ v⃗)}
     N1_sc ← {f⃗ ∈ N1_s | c⃗_sup ≪ f⃗)}
     IF (N1_sc = ∅)
          Remove FA[1]
          IF (N1_s ≠ ∅)
               Discontinued ← Discontinued ∪ N1_s
     ELSE IF (∃v⃗ ∈ N1_sc | ip⃗_suc ≪ v⃗)
          ip⃗_csup ← iPoint ({c⃗_sup ∪ (⋃_{i≠1} FA[i])
          IF (∃v⃗ ∈ N1_sc | ip⃗_csup ≪ v⃗)
               (FA[1], SOL) ← IP-MO-RBFS (n_1, N1_sc, ip⃗_csup, SOL)
     ELSE
          Move FA[1] to the end of FA[]
END LOOP
```

Tabla 5.3: Algoritmo $IP\text{-}MO\text{-}RBFS$.

de coste ((5, 4), (3, 6) y (∞, ∞) respectivamente) hacia sus antecesores. A diferencia de *Pareto-MO-RBFS*, *IP-MO-RBFS* actualiza el vector de coste del nodo n_1 con el punto ideal de los vectores no dominados de los caminos que acabamos de discontinuar. Es decir, el coste actualizado del nodo n_1 contiene información acerca de los mejores valores para cada uno de los objetivos que hemos alcanzado en la exploración del árbol de búsqueda bajo n_1. Estas componentes de coste mínimo pueden pertenecer (como es el caso de nuestro ejemplo) a caminos distintos. El coste mínimo para el primer objetivo se alcanza a través de n_6, mientras que para el segundo objetivo se alcanza a través de n_5.

El nodo n_1 envía este nuevo vector de coste a sus antecesores, lo cual permite que nodos superiores del árbol (y por propagación de la información mediante la cota superior, nodos de cualquier rama del árbol), tengan conocimiento de la calidad de los caminos disponibles a través de n_1. Cuando se vuelve a reexpandir n_1, traslada este coste a sus sucesores n_2 y n_3. Estos nodos actualizan su vector de coste empleando la función $MaxVector$ ya vista en la sección 5.2, tal y como se puede apreciar en la parte derecha de la figura 5.9a. Esto generará reexpansiones más directas del subárbol de búsqueda bajo n_1.

En *IP-MO-RBFS*, al igual que en *Pareto-MO-RBFS*, la lista FA que contiene los sucesores no está ordenada, lo cual favorece el rendimiento del algoritmo. Su funcionamiento básico, ilustrado en la figura 5.9b, es el siguiente:

- Cada elemento de la lista está asociado a un sucesor del nodo actual y contiene su vector de coste, el cual se actualizará con la expansión de los diferentes nodos bajo dicho sucesor.
- Siempre se procesan los vectores almacenados en el primer elemento de la lista.
- Cuando los vectores de coste de un sucesor están dominados por soluciones ya encontradas, el sucesor se elimina de FA. Este es el caso del nodo n_1 en la figura 5.3b, cuyo coste (7, 7) está dominado por la solución de coste (6, 6).
- Cuando los vectores de coste de un sucesor son *estrictamente peores* que la cota superior (es el caso de los nodos n_2 y n_3), se elimina el nodo de FA, alimentando su vector de coste el cálculo del punto ideal que servirá como nuevo vector de coste para el padre n. Estos vectores referencian costes de caminos discontinuados, pero no descartados definitivamente. En nuestro caso el nodo n informa con su cota superior (3, 3) que hay un camino de más interés que el que ofrecen n_2 y n_3.
- Si un sucesor es *estrictamente peor* que otros sucesores, entonces se pasa al final de la lista FA para ser procesado en último lugar. Dados varios nodos sucesores, se da preferencia a aquellos que son estrictamente mejores que otros. Entre los mejores no dominados no se establece ningún tipo de preferencia. En nuestro ejemplo, el nodo n_4, el cual, aún siendo candidato a ser explorado, ya que no está dominado por soluciones previas ni es estrictamente peor que la cota superior

de n, ofrece un camino de *peor coste* que n_k, motivo por el cual se promueve el procesamiento de este último nodo.

- En cualquier otro caso, el nodo se expande usando por un lado su coste (posiblemente actualizado), y como cota superior el punto ideal de los vectores de coste de los caminos pendientes (ver ejemplo de la figura 5.8).
- Cuando ya no hay nodos pendientes de procesar en la lista FA, entonces se devuelve hacia el nodo n la información de los caminos discontinuados que puedan servir de interés en reexpansiones futuras. Esta información se almacena en el conjunto *Discontinued*. De dicho conjunto $IP\text{-}MO\text{-}RBFS$ calcula el punto ideal y devuelve este vector como nuevo coste de n. Si no hubiera nodos en esta situación, porque fueran soluciones ya encontradas o porque su coste exceda el de soluciones ya encontradas, entonces el nodo n recibe como actualización de su coste el vector (∞, ∞) para que se autodescarte en la búsqueda.

En la sección 5.3.2 presentamos la resolución detallada de un problema multiobjetivo con el algoritmo $IP\text{-}MO\text{-}RBFS$.

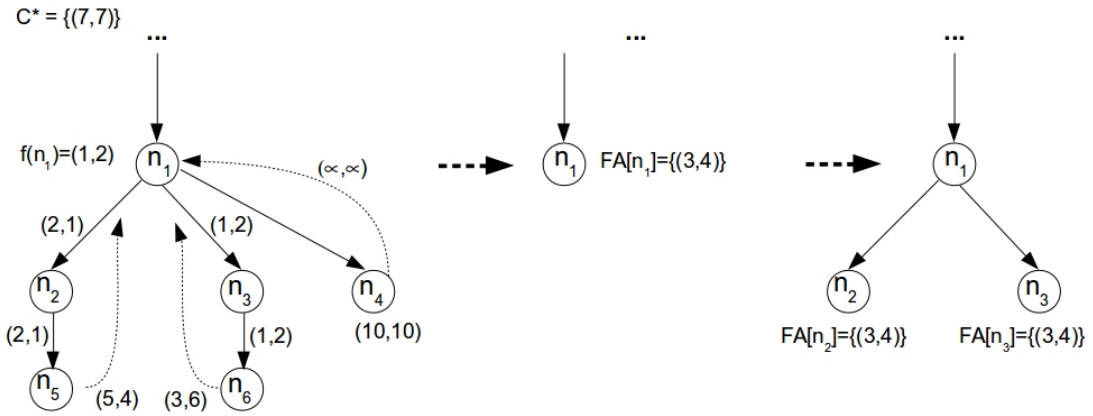
5.3.2. Ejemplo de IP-MO-$RBFS$

El problema multiobjetivo a resolver en esta sección es el mismo que se ha empleado para $Pareto\text{-}MO\text{-}RBFS$. El grafo se puede consultar en la figura 5.4.

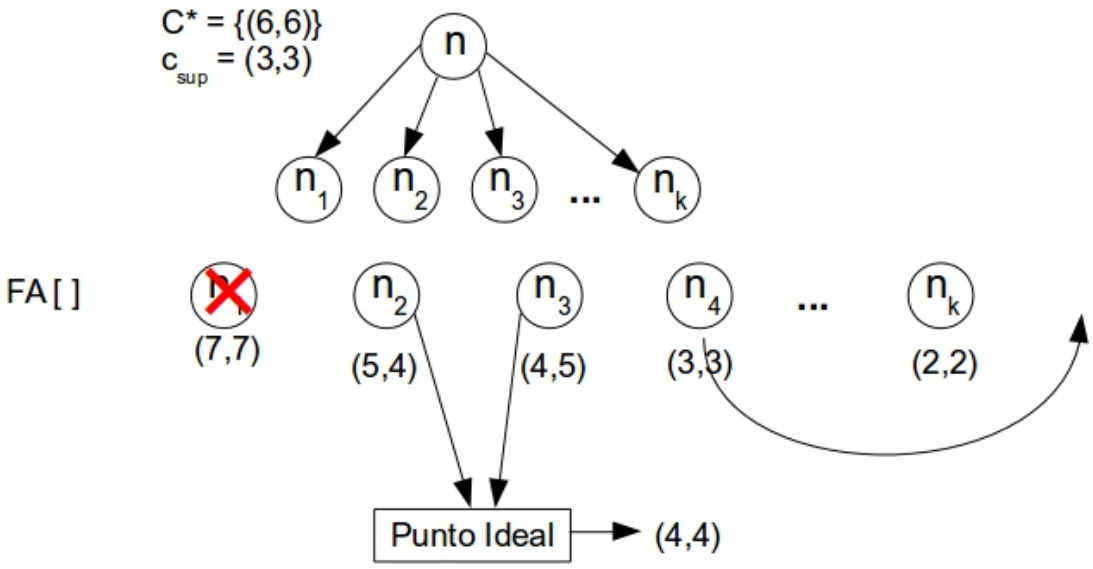
La situación inicial, ilustrada en la figura 5.10a, es prácticamente idéntica a la ya vista para $Pareto\text{-}MO\text{-}RBFS$. La única diferencia es que en este caso, la información relativa a los dos caminos de interés que quedan pendientes a través de n_2 y n_3 no se almacena de modo independiente (guardando ambos vectores de coste), sino que se compacta con el cálculo del punto ideal para ambos vectores. El vector resultado, $(2, 1)$ le indica al nodo n_1 que se ha encontrado un camino donde se consigue un coste 2 para el primer objetivo y otro camino (que puede o no ser el mismo) con un coste 1 para el segundo objetivo.

El nodo n_1 explorará su subárbol de búsqueda mientras sea capaz de mejorar o igualar alguno de los valores fijados. Cuando las ramas exploradas sean estrictamente peores que estos valores mínimos, se realizará *backtracking* para intentar retomar alguno de los caminos abandonados.

La figura 5.10b muestra la expansión del nodo n_1. Su valor de actualización FA_{n_1} es su vector de coste y su cota superior, el vector calculado anteriormente. Dado que no se ha encontrado ninguna solución que pueda dominar este camino, el nodo n_1 se expande, generando los nodos γ_1 y γ_2. Ambos nodos son estrictamente peores que la cota superior, $(2, 1)$, con lo cual la búsqueda se discontinúa. Ambos nodos informan a su antecesor del coste de los caminos alcanzados, $(3, 6)$ y $(5, 5)$. Tras calcular el punto ideal, el resultado, $(3, 5)$, es adoptado por n_1 como su nuevo vector de coste. Esto indica que a través de n_1 el mejor coste conseguido para el primer objetivo es 3 y para el segundo objetivo 5.

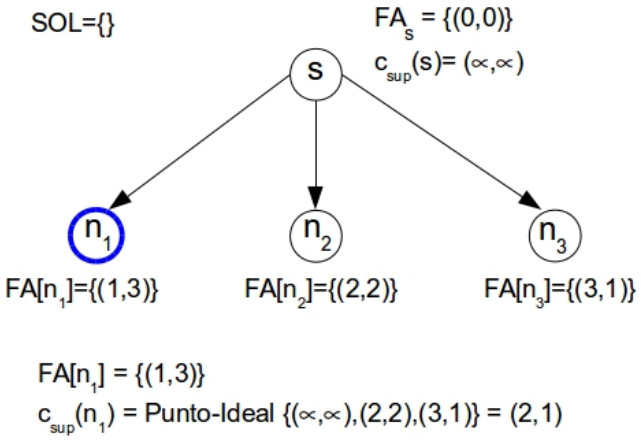


(a) Actualización de costes mediante FA_n

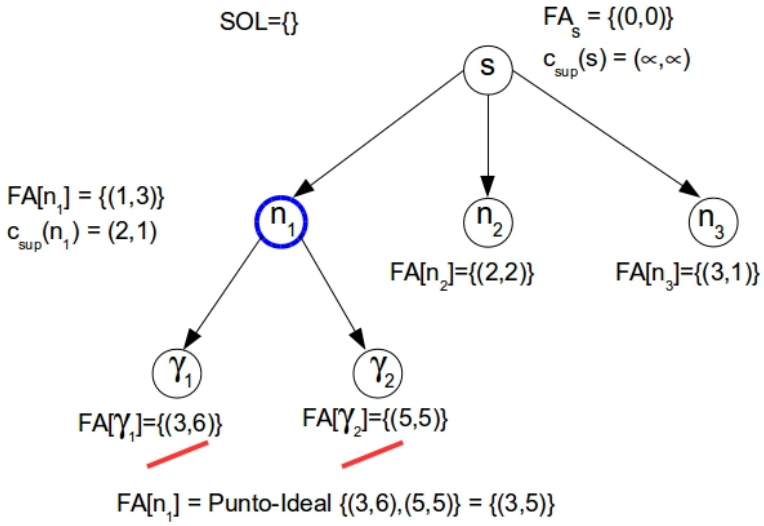


(b) Gestión del conjunto *FA*

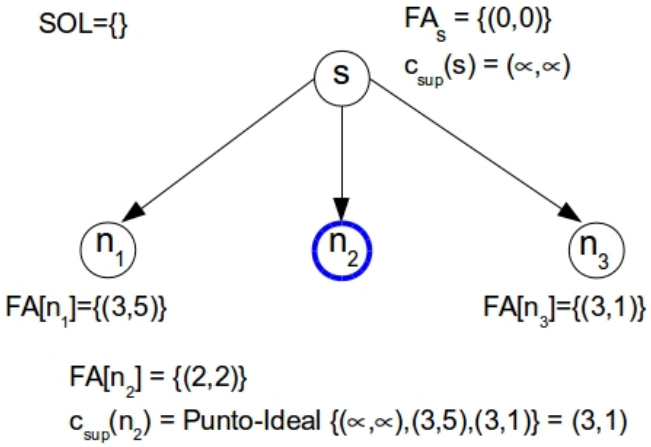
Figura 5.9: Comportamiento de *IP-MO-RBFS*



(a) Paso 1

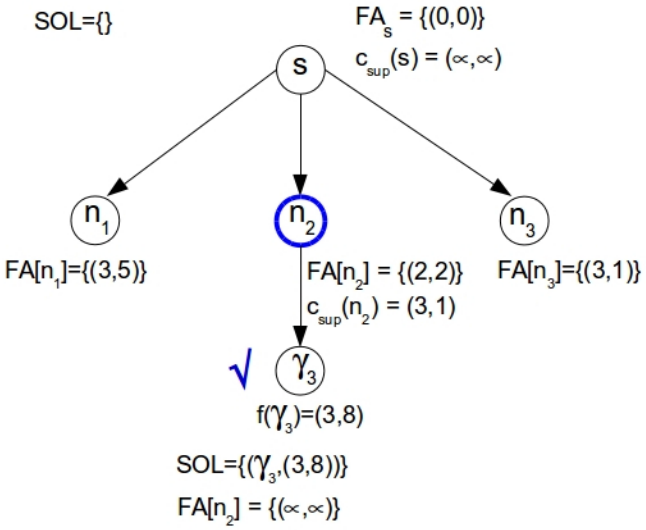


(b) Paso 2

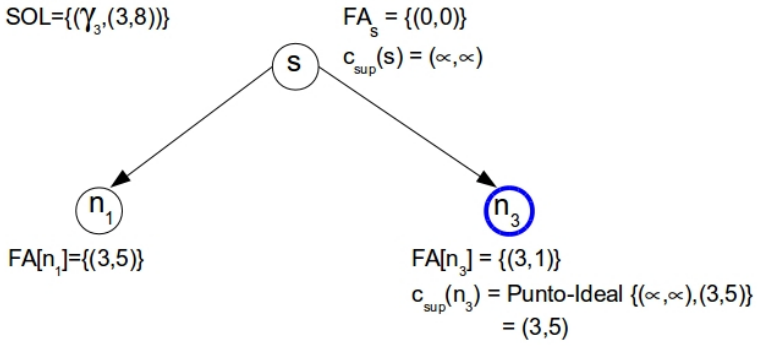


(c) Paso 3

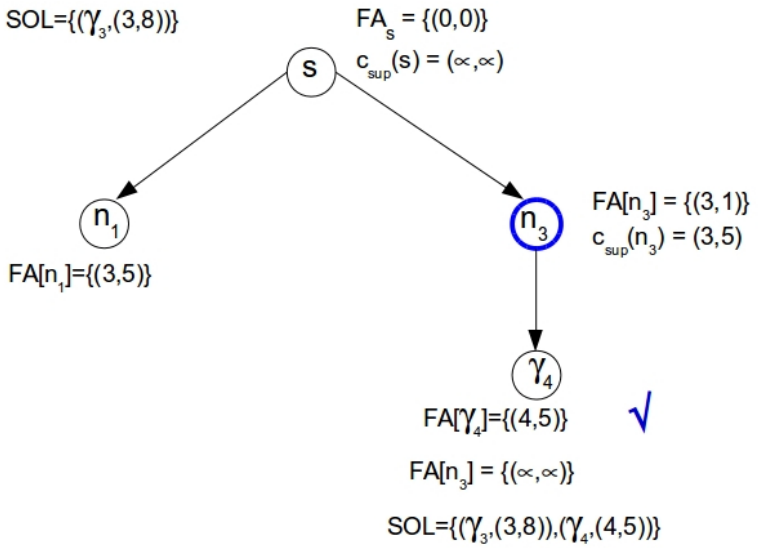
Figura 5.10: Resolución mediante $IP\text{-}MO\text{-}RBFS$ (1).



(a) Paso 4

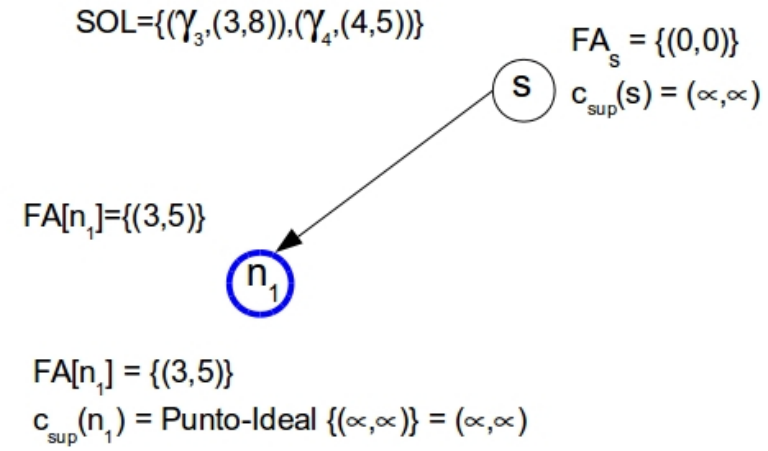


(b) Paso 5

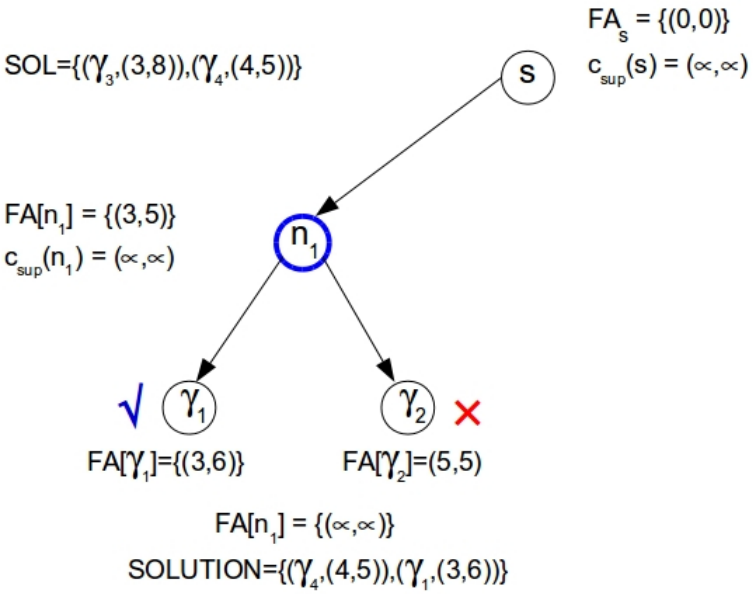


(c) Paso 6

Figura 5.11: Resolución mediante *IP-MO-RBFS* (2).



(a) Paso 7



(b) Paso 8

Figura 5.12: Resolución mediante $IP\text{-}MO\text{-}RBFS$ (3).

Dado que el vector de coste actualizado de n_1 es estrictamente peor que alguno de los nodos pendientes en el árbol (en este caso, estrictamente peor que n_2 y n_3), se postpone el procesamiento de n_1. El procesamiento de n_2 se refleja en la figura 5.10c. La cota superior de este nodo se calcula de modo análogo a como se realizó para n_1, resultando en el vector $(3, 1)$.

La expansión de n_2, reflejada en la figura 5.11a, da lugar al nodo γ_3, cuyo coste es $(3, 8)$. Este nodo, aún cuando empeora cualquier otro valor previo para el segundo objetivo, iguala el mejor valor obtenido para el primer objetivo, 3. Al no ser, por lo tanto, estrictamente peor que la cota superior, se expande. Como γ_3 es un nodo solución, se añade al conjunto SOL, aún cuando es una solución subóptima. Será necesario descartarla en exploraciones posteriores.

Al no haber nada pendiente de exploración en la rama actual, el coste de n_2 se actualiza a (∞, ∞), para evitar reexpansiones posteriores. Tras actualizar el coste de n_2, este nodo ya pasa a estar dominado por la solución que acabamos de encontrar, con lo cual el nodo se descarta definitivamente, tal y como se refleja en la figura 5.11b. El nodo n_3 es el nuevo candidato a ser expandido. Dado que no está dominado por la solución ya localizada, se expande, usando como cota superior la información relativa al único camino con posibilidades de ser expandido, a través del nodo n_1.

La figura 5.11c muestra la expansión de n_3. De modo similar a lo que sucedió con n_2, se genera su único sucesor, γ_4, el cual no es estrictamente peor que la cota superior (iguala la calidad del coste para el segundo objetivo, 5), así que se expande. Al ser nodo solución, se añade al conjunto SOL. Dado que los costes de las dos soluciones encontradas son no dominados entre sí, ambas soluciones se mantienen. Al no haber más opciones de exploración a través de n_3, éste actualiza su coste a (∞, ∞) (valor devuelto por γ_4). Debido a esta actualización, n_3 pasa a estar dominado por soluciones ya encontradas y se descarta.

La única opción pendiente es explorar el nodo n_1. Dado que no hay otros caminos pendientes, su cota superior toma el valor (∞, ∞), tal y como podemos ver en la figura 5.12a. La figura 5.12b muestra la expansión de n_1. Sus sucesores, γ_1 y γ_2, calculan su nuevo vector de coste usando la información almacenada en el valor actualizado de n_1, empleando la función $MaxVector$ de modo similar a lo ya visto para *Pareto-MO-RBFS*.

El nodo γ_1 no está dominado por ninguna solución y, obviamente, tampoco es estrictamente peor que la cota superior, con lo cual se expande. Al ser nodo solución, se añade al conjunto SOL. Este conjunto se purga a continuación, eliminando soluciones dominadas, con lo cual desaparece del mismo γ_3. En cuanto al nodo γ_2, al estar dominado por el vector de coste de γ_4, se discontinúa también de modo definitivo. El vector de coste de n_1 se actualiza a (∞, ∞), descartándolo de modo definitivo. Al no haber más nodos pendientes de expansión, la búsqueda finaliza e $IP\text{-}MO\text{-}RBFS$ devuelve el conjunto de soluciones óptimas.

	Cota superior	Cálculo de la cota	Retroceso
Pareto-MO-RBFS	Un conjunto de vectores por cada nodo	Conjunto de vectores no dominados de las ramas en las cuales se ha retrocedido, incluyendo también los de los *hermanos* no expandidos del nodo actual	Si el coste del camino está dominado por alguno de los vectores de la cota
$MOMA^*0$	Un vector cota por cada nodo	Menor lexicográfico de los vectores de coste actualizados de las ramas en las cuales se ha retrocedido, incluyendo también los de los *hermanos* no expandidos del nodo actual	Si el coste del camino es peor lexicográfico que la cota
IP-MO-RBFS	Un vector cota por cada nodo	Punto Ideal de los vectores de coste actualizados de las ramas en las cuales se ha retrocedido, incluyendo también los de los *hermanos* no expandidos del nodo actual	Si el coste del camino es estrictamente peor que la cota

Tabla 5.4: Resumen de algoritmos multiobjetivo exactos *depth-first* basados en profundización iterativa

5.4. Resumen

En este Capítulo se han mostrado sendas variantes de *RBFS* para el caso multiobjetivo: *Pareto-MO-RBFS* e *IP-MO-RBFS*. En la tabla 5.4 se muestra un pequeño resumen de las características de estos algoritmos, incluyendo además la variante lexicográfica, $MOMA^*0$, presentada en la sección 3.6.3

Al igual que sus homólogos de profundización iterativa, estos algoritmos juegan con el tamaño de la frontera de Pareto para intentar disminuir la cantidad de reexpansiones y tests de dominancia, los cuales son los dos principales factores a tener en cuenta para evaluar el rendimiento de la búsqueda multiobjetivo de tipo *depth-first*. Mientras *Pareto-MO-RBFS* mantiene una cota multivectorial, *IP-MO-RBFS* y $MOMA^*0$ mantienen una cota sencilla, formada por un solo vector.

Esta cota representa la calidad de la búsqueda realizada por ramas del árbol distintas de la actual. Si a medida que profundizamos en la búsqueda por la rama actual, los caminos pasan a tener peor calidad que los que están pendientes, entonces se reconsidera la rama a expandir. En el caso de *Pareto-MO-RBFS*, peor calidad implica que la rama actual está dominada por la cota; en el caso de *IP-MO-RBFS*, implica que la rama actual tiene un coste estrictamente peor que la cota; en el caso de $MOMA^*0$, implica que la rama actual tiene un coste peor lexicográfico que la cota.

Al estar la cota de *IP-MO-RBFS* calculada a base de mínimos (Punto Ideal) para cada componente, la profundización avanza más lentamente que en el caso de *Pareto-MO-RBFS*. Esto hace que la cantidad de reexpansiones de nodos en *IP-MO-RBFS* sea mayor que en *Pareto-MO-RBFS*. Sin embargo, al mantener la cota como un vector, el número de comparaciones a realizar contra la misma disminuye, aumentando el rendimiento en este sentido. La evaluación presentada en el Capítulo 8 permitirá determinar cual de los dos parámetros, reexpansiones o tests de dominancia, es el dominante en lo que respecta al rendimiento de estos algoritmos.

En el Capítulo 6 analizaremos el tercer gran grupo de algoritmos multiobjetivo estudiados en esta tesis, los algoritmos secuenciales y los basados en *DF-BnB*.

Capítulo 6

DF-BnB multiobjetivo y algoritmos secuenciales

En capítulos anteriores hemos presentado diferentes algoritmos multiobjetivo, tanto basados en profundización iterativa como en búsqueda recursiva *best-first* multiobjetivo. Tal y como se comentaba en el Capítulo 3, en problemas de búsqueda en árboles estos algoritmos presentan unos requisitos de memoria lineales con la profundidad del árbol generado para el proceso de búsqueda, frente a los requisitos exponenciales de algoritmos con un enfoque *best-first* puro. El principal inconveniente de estos algoritmos es que el número de expansiones de nodos es superior al de un algoritmo *best-first*: muchos de los nodos se generan repetidamente en iteraciones o reexpansiones sucesivas. De hecho los algoritmos $PIDMOA^*$ (sección 4.2) y *Pareto-MO-RBFS* (sección 5.2) tratan precisamente de disminuir esas reexpansiones mediante el tratamiento simultáneo de los diferentes objetivos del problema.

Existe un tercer gran grupo de algoritmos, los algoritmos secuenciales ó por fases, que dividen la búsqueda en una serie de fases, en cada una de las cuales aplican un algoritmo de búsqueda. En el capítulo 3 se presentaba uno de estos algoritmos, $IDMOA^*$, que divide la búsqueda en tantas fases como objetivos a considerar tiene el problema multiobjetivo. En cada una de estas fases aplica IDA^* sobre el correspondiente objetivo. Asimismo también se esbozaba una variante de *DF-BnB* al caso multiobjetivo, la cual será definida con detalle en este capítulo.

Los algoritmos vistos en los capítulos 4 y 5 basan su expansión en una cota que limita la profundización en el árbol de búsqueda. Existe una tercera aproximación al uso de cotas para la búsqueda *depth-first*, la empleada por el algoritmo *DF-BnB* (Lawler y Wood, 1966), el cual destaca por su sencillez y buen rendimiento. En este Capítulo presentaremos una extensión de *DF-BnB* al caso multiobjetivo. Esta extensión, tanto desde el punto de vista de algoritmo exacto como de algoritmo aproximado, ya ha sido considerada anteriormente por otros autores (White, 1982) (White, 1984) (Bausch, 1992) (Sourd y Spanjaard, 2008) (Galand et al., 2008) (Rollón y Larrosa, 2009) (Galand et al., 2010) (Buer y Pankratz, 2010) (Delort y Spanjaard, 2013).

En la sección 6.1 comentaremos los aspectos principales del algoritmo *DF-BnB*. La sección 6.2 muestra una aproximación ya existente de *DF-BnB* al caso multiobjetivo. Por último, en la sección 6.3 mostraremos una extensión de *DF-BnB* al caso multiobjetivo que permite realizar búsquedas en grafos infinitos.

6.1. *DF-BnB*

Como ya comentamos en el capítulo 3, *DF-BnB* establece una cota superior para el coste de una solución óptima, la cual se va reduciendo a medida que localizamos soluciones de mejor calidad. Una vez fijada la cota, se realiza una exploración *depth-first*. Un camino en el árbol de búsqueda seguirá expandiéndose mientras su estimación de coste no supere la cota; en caso de que sí la supere, la búsqueda se discontinúa por esa rama (acotación o poda). Si durante la búsqueda se localiza un nodo final γ y el coste para llegar al mismo, $\vec{f}(\gamma)$, no excede la cota, entonces $\vec{f}(\gamma)$ se usa como nueva cota superior durante el resto de la búsqueda. A medida que se van localizando mejores soluciones, las ramas se podan más cerca de la raíz, contribuyendo a una importante disminución en el número de nodos generados.

Aunque *DF-BnB* tiene un consumo de memoria lineal con la profundidad del árbol, puede expandir, al igual que la búsqueda *depth-first*, más nodos que una búsqueda del tipo *best-first*. Como ya hemos comentado anteriormente, por ejemplo en el caso de la profundización iterativa este *exceso* de nodos se debe a reexpansiones. En el caso de *DF-BnB* no hay ninguna reexpansión, ya que cada rama se recorre una única vez. El problema viene dado por la calidad de la cota inicial, puesto que si dicha cota no es óptima, es posible que *DF-BnB* expanda ramas que lleven a soluciones subóptimas o que sencillamente no lleven a solución alguna y tengan coste superior al de la solución óptima.

La versión básica del algoritmo no incluye ningún tipo de cota inicial, sino que expande las ramas hasta encontrar una solución cualquiera. El coste de esta primera solución es el que sí se usará como cota tentativa mientras no se localicen mejores soluciones. Esta versión plantea un problema con problemas representados por grafos ó árboles infinitos, pues es posible que el algoritmo no sea completo, es decir, *DF-BnB* puede no encontrar ninguna solución y no finalizar aún habiendo soluciones.

La calidad de la cota inicial es clave para el rendimiento de *DF-BnB*. Si la cota es demasiado baja, podría ser incluso inferior al coste óptimo de la solución, con lo cual el algoritmo finaliza pero no encuentra soluciones al problema. Si la cota es demasiado alta, *DF-BnB* expandirá ramas que conducen a nodos y soluciones de calidad baja, convergiendo más lentamente a la solución óptima.

Se han realizado análisis de los casos en los cuales interesa emplear un enfoque *DF-BnB* ó un enfoque *iterative-deepening* en problemas con un único objetivo (Vempaty et al., 1991) (Zhang y Korf, 1995). La conclusión es que en problemas con una alta densidad de soluciones se debe aplicar un enfoque *DF-BnB*. Esto se debe a que a mayor número de soluciones hay una mayor probabilidad de realizar podas. Por el contrario,

```
MO-DF-BnB (n, SOLUTION)
     Generate all children of n: n_1, n_2, ..., n_k
     FOR (i from 1 to k)
          NdomSol ← {f⃗ ∈ F(n_i)  |  ∄c⃗ ∈ SOLUTION, c⃗ ≺ f⃗}
          IF (NdomSol ≠ ∅) THEN
               IF (n_i is a goal node) THEN
                    SOLUTION ← nodomset (SOLUTION ∪ NdomSol)
               ELSE
                    MO-DF-BnB (n_i, SOLUTION)
```

Tabla 6.1: Algoritmo MO-DF-BnB

cuando la densidad de soluciones es baja, la profundización iterativa es más eficiente que *DF-BnB*, ya que a éste último se le hace más difícil localizar soluciones, y por tanto no puede podar con tanta frecuencia.

6.2. *DF-BnB* Multiobjetivo

El algoritmo *DF-BnB* puede ser fácilmente generalizado al caso multiobjetivo. Esta generalización, que denominaremos *MO-DF-BnB* (*MultiObjective Depth-first Branch & Bound*), realiza una búsqueda de tipo *depth-first* expandiendo cada uno de los caminos en el árbol de búsqueda hasta que localiza un nodo final, el cual, obviamente, no tiene por qué ser óptimo. Este nodo y su vector de coste se añaden al conjunto de soluciones tentativo *SOLUTION*, y el vector de coste pasa a ser la primera cota superior, reanudando la exploración del árbol. Esta cota, en el caso multiobjetivo, ya no será un escalar, sino un conjunto de vectores, concretamente los vectores de coste no dominados de todas las soluciones encontradas hasta ahora.

Si el camino que se está explorando tiene un coste dominado por alguna de las soluciones ya localizadas, entonces dicho camino no podrá conducir a un óptimo de Pareto (asumimos costes no negativos en los arcos del grafo tal y como se indica en 7.1), y por lo tanto la rama se poda. Si se localiza un nuevo nodo final γ_i, se compara igualmente su estimación de coste $\vec{f}(\gamma_i)$, contra el conjunto *SOLUTION*. En caso de estar dominado por algún vector de coste del mismo, la solución se descarta; en caso contrario (la solución no está dominada por ninguna de las ya encontradas), se añade el par $(\gamma_i, \vec{f}(\gamma_i))$ al conjunto *SOLUTION*, eliminando del conjunto aquellos vectores dominados por la solución que acabamos de encontrar.

En cualquier caso, lo que contiene *SOLUTION* en todo momento es información de nodos solución, óptimos ó subóptimos, que emplea simultáneamente como cota. Cuando el algoritmo finaliza, el algoritmo *MO-DF-BnB* devuelve el conjunto C^* con todas las soluciones óptimas del problema.

La Tabla 6.1 muestra el pseudocódigo de *MO-DF-BnB* basado en el algoritmo escalar

incluído en (Zhang y Korf, 1995). La llamada inicial del algoritmo es *MO-DF-BnB* $(s, \emptyset)$, siendo s el estado inicial. La función *nd(X)* recibe como parámetro un conjunto de vectores $\vec{X}$ y devuelve un subconjunto de $\vec{X}$ con todos los vectores no dominados entre sí. $F(n_i)$ denota el conjunto de vectores de estimación de coste no dominados asociados al nodo n_i, definidos como $\vec{f}(n_i) = \vec{g}(n_i) + \vec{h}(n_i)$, donde $\vec{h}(n_i) \in H(n_i)$. El vector $\vec{g}(n_i)$ denota el coste del camino que el algoritmo está explorando actualmente desde el nodo inicial s_0 hasta n_i, y $H(n_i)$ el conjunto de vectores heurísticos no dominados del nodo n_i.

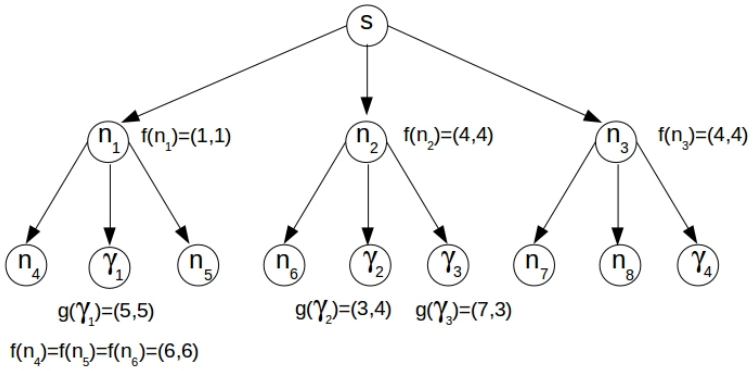
6.2.1. Ejemplo de *DF-BnB* Multiobjetivo

En la figura 6.1a se muestra un espacio de estados en forma de árbol que incluye 4 nodos solución, dos de los cuales (γ_2 y γ_3) son óptimos de Pareto. Suponiendo que la expansión del árbol se realiza de izquierda a derecha, el algoritmo *DF-BnB* multiobjetivo genera todas las ramas completas (hasta llegar a un nodo hoja) mientras no encuentre un nodo solución, tal y como se puede apreciar en la figura 6.1b. En nuestro caso el primer camino solución localizado lleva a γ_1, con un coste $(5, 5)$. A partir de este momento, dicho coste le sirve de cota para ir podando las sucesivas ramas exploradas, hasta que localiza alguna otra solución. Esta situación se da en los nodos n_5 y n_6; no así en el nodo n_2, al no estar su vector de coste, $(2, 2)$, dominado por la cota actual.

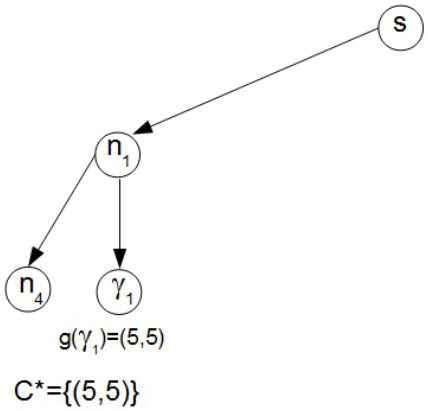
Cuando *DF-BnB* multiobjetivo localiza el nodo γ_2 (figura 6.1c), el cual es un óptimo de Pareto, su coste domina al de la solución γ_1, con lo cual esta primera solución es descartada al igual que su vector de coste. Además, el vector de coste γ_2 se incorpora al conjunto C^*. Siguiendo con la expansión del árbol, el algoritmo localiza la solución γ_3 (figura 6.2a), también óptimo de Pareto, añadiéndola por tanto a C^*. Como se puede apreciar, el conjunto C^* actúa en todo momento como cota superior multivectorial para la expansión. En la generación de la última rama del árbol (figura 6.2b), el coste del nodo n_3 está dominado por el vector $(3, 4)$ asociado a γ_2, con lo cual esta rama se poda, descartando todo el subárbol correspondiente.

En el ejemplo se puede apreciar que *DF-BnB* multiobjetivo puede localizar y trabajar con soluciones subóptimas, aunque posteriormente las descarta a favor de otras soluciones mejores (y posiblemente óptimas) a medida que las va localizando durante la exploración del árbol de búsqueda. Dependiendo de como estén distribuidas las soluciones en el árbol de búsqueda, las podas que realice *DF-BnB* multiobjetivo tendrán mayor o menor eficiencia. Por ejemplo, si las soluciones están ordenadas en orden creciente de *optimalidad* y la exploración del árbol se hace en este mismo orden, la cantidad de podas realizadas será pequeña en comparación a una situación en la cual se localicen óptimos de Pareto en las etapas iniciales del algoritmo.

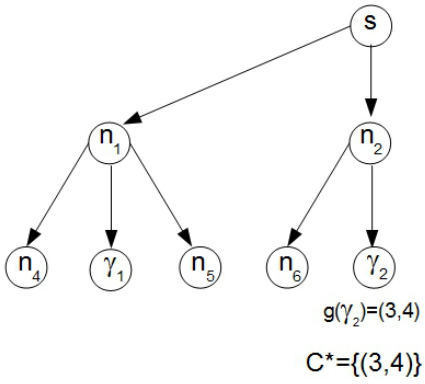
Suponiendo que tratamos con árboles finitos, costes positivos mínimos y no nulos en todos los objetivos para los arcos y funciones heurísticas admisibles, es trivial comprobar que en estos casos el algoritmo *MO-DF-BnB* es completo y óptimo. Sin em-



(a) Espacio de estados

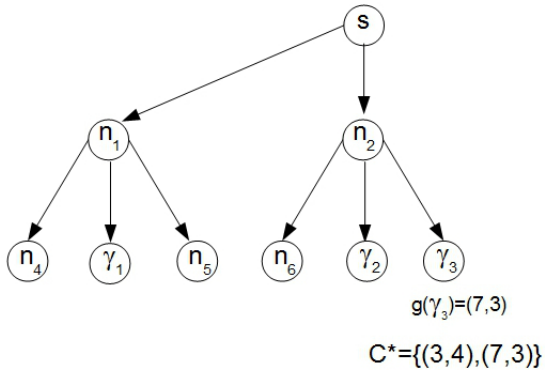


(b) Etapa 1 de la búsqueda

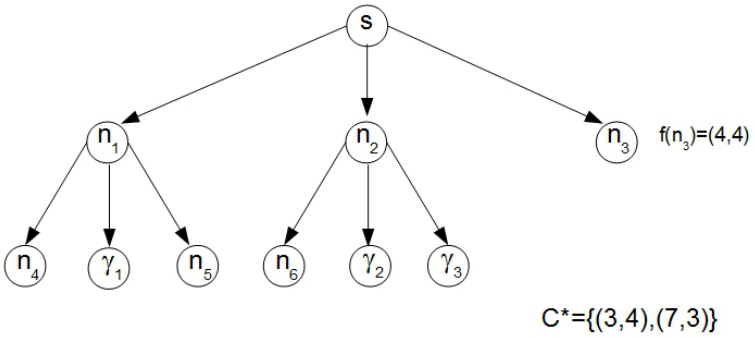


(c) Etapa 2 de la búsqueda

Figura 6.1: Búsqueda multiobjetivo con *DF-BnB* multiobjetivo (1)



(a) Etapa 3 de la búsqueda



(b) Etapa 4 de la búsqueda

Figura 6.2: Búsqueda multiobjetivo con *DF-BnB* multiobjetivo (2)

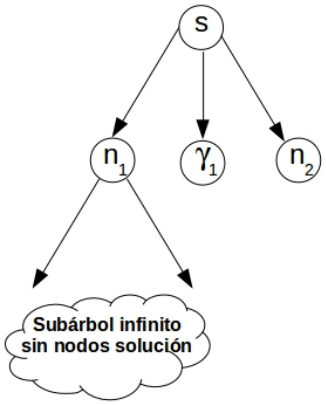


Figura 6.3: Patología de *DF-BnB* en grafos infinitos

bargo, si *DF-BnB* multiobjetivo, al igual que *DF-BnB* para un único objetivo, tuviera que lidiar con un espacio de estados infinito es posible que el algoritmo no finalizase, dado que es posible que no llegue a encontrar una primera solución que le sirva de cota superior inicial.

Esta situación se daría en el árbol ejemplo de la figura 6.1a, suponiendo que *DF-BnB* multiobjetivo expande el árbol de izquierda a derecha. En este caso el algoritmo, al no disponer de una cota inicial, es incapaz de abandonar la exploración de la primera rama a través del nodo n_1, y por tanto no finalizaría nunca. Por el contrario, si *DF-BnB* multiobjetivo realizara la expansión de derecha a izquierda, el algoritmo acabaría, devolviendo todos los óptimos de Pareto. En la siguiente sección mostramos como solventar esta situación realizando la búsqueda en dos fases, una para localizar una cota inicial y otra para recuperar la frontera de Pareto.

6.3. *DF-BnB* Multiobjetivo en grafos infinitos

Como se comentará en detalle en el Capítulo 8, las pruebas realizadas para evaluar los diferentes algoritmos objeto de esta tesis incluyen tanto árboles finitos como infinitos. Este último tipo de problemas no son directamente resolubles por *MO-DF-BnB*, por lo cual se ha optado por emplear un algoritmo híbrido en dos fases, en la primera de las cuales, mediante el uso de un algoritmo completo, se localiza una primera solución, la cual servirá de cota superior para la segunda fase, en la cual se emplea el algoritmo *MO-DF-BnB*. El algoritmo candidato para la primera fase es cualquier algoritmo, tanto multiobjetivo como para un objetivo, que sea completo cuando trabaja con espacios de estados infinitos. Los algoritmos que emplean cota para la profundización (BID, IDA^*, $RBFS$, $IDMOA^*$, $PIDMOA^*$ o $IPID^*$), convenientemente adaptados para devolver una única solución, son alternativas claras para fijar la cota inicial de *MO-DF-BnB*.

Obviamente no es necesario ejecutar completamente los algoritmos anteriores para

```
IPID-1* (G, s, Γ )
SOLUTION ← ∅; ThresholdSet ← H(s)
WHILE (ThresholdSet ≠ ∅) ∧ (SOLUTION = ∅)
     threshold ← iPoint (ThresholdSet)
     (ThresholdSet, SOLUTION) ← DFS (s, threshold, SOLUTION)
return (SOLUTION)


DFS (n, currentTh, SOLUTION)
NdomTh ← {f ∈ F(n) | ¬(currentTh ≪ f)}
IF (NdomTh = ∅) THEN return (NdomSol, SOLUTION);
IF (n ∈ Γ ) THEN
     SOLUTION ← SOLUTION ∪ {(n, f(n))}
     return (∅, SOLUTION)
ELSE
     ThresholdDFS ← ∅
     sucessors ← expand_node (n)
     FOR each suc in sucessors DO
          (ThresholdRT, SOLUTION) ← DFS(suc, currentTh, SOLUTION)
          IF (SOLUTION ≠ ∅) THEN
               return (∅, SOLUTION)
          ThresholdDFS ← nodomset (ThresholdDFS ∪ ThresholdRT)
     return (ThresholdDFS, SOLUTION)
```

Tabla 6.2: Algoritmo $IPID$-1*

obtener una solución inicial que sirva de cota a *MO-DF-BnB*, así que se han reescrito para que devuelvan únicamente la primera solución que encuentren. Como ejemplo, vemos en la siguiente tabla 6.2 el pseudocódigo del algoritmo $IPID$-1*, una variante de $IPID^*$ que devuelve la primera solución que localiza.

Por lo tanto es necesario incluír una llamada adicional al algoritmo $IPID$-1* y con el resultado, la primera solución localizada por este algoritmo (la cual puede ser subóptima según lo visto), invocar a *MO-DF-BnB*, tal y como podemos ver en el algoritmo base de la Tabla 6.3.

Es necesario destacar que el rendimiento del algoritmo *MO-DF-BnB* debería ser

```
MO-DF-BnB-base (G,s)
     cota = IPID-1* (G,s,Γ)
     IF (cota)
          MO-DF-BnB (s, cota)
```

Tabla 6.3: Algoritmo MO-DF-BnB-base

mejor cuanta más calidad tenga la solución que se le proporciona como cota inicial, tanto en el caso escalar como en el multiobjetivo. En el caso de los algoritmos de profundización iterativa que pueden *alimentar* a *MO-DF-BnB* con una solución inicial, tanto $IDMOA^*$ como $IPID^*$ pueden devolver temporalmente soluciones subóptimas, mientras que todas las soluciones que localiza $PIDMOA^*$ son óptimas (ver ejemplos del Capítulo 4 y propiedades de los algoritmos en el Capítulo 7). Por lo tanto, en cuanto a calidad de la cota inicial, el candidato sería $PIDMOA^*$. De todos modos se hace necesario un estudio detallado de la calidad de esta cota y el coste temporal para su obtención.

No se pueden descartar tampoco enfoques tales como obtener una solución inicial usando el algoritmo IDA^* (Korf, 1985a) sobre uno de los objetivos del problema. De todos modos, aunque IDA^* es óptimo para el caso de un objetivo, no tenemos asegurado que la solución obtenida con esta opción será un óptimo de Pareto. Por ejemplo, en el problema podría haber dos soluciones γ_1 y γ_2 con vectores de coste $\vec{f}(\gamma_1) = (4, 7)$ y $\vec{f}(\gamma_1) = (4, 6)$. Si utilizamos IDA^* sobre el primer objetivo y el algoritmo localiza en primera instancia el nodo γ_1, entonces devuelve una cota (4,7), de la cual podemos asegurar que tiene el valor mínimo para el objetivo considerado, pero no podemos asegurar que sea un óptimo de Pareto. De modo similar, otras opciones como BID ó $RBFS$, deben contemplarse también en los análisis de rendimiento.

6.4. Resumen

En este Capítulo hemos presentado sendas versiones simplificadas del algoritmo *DF-BnB* que permite trabajar con espacios de estados finitos e infinitos respectivamente. Para el último caso, el algoritmo se alimenta inicialmente de una cota proporcionada por un algoritmo completo, lo cual le permite asegurar su terminación.

A modo de resumen, las principales similitudes y diferencias entre los enfoques de profundización iterativa (ID) y los enfoques *DF-BnB* son las siguientes:

- Los algoritmos de profundización iterativa utilizan una cota superior que delimita el subespacio de soluciones explorado durante cada iteración, avanzando dicha cota paulatinamente hacia los óptimos de Pareto. Por otro lado los algoritmos *DF-BnB* usan una cota superior del Frente de Pareto, la cual se va ajustando a medida que se van localizando soluciones de mejor calidad.

- Los algoritmos de profundización iterativa generan sobrecarga computacional debida a la reexpansión de nodos en las diferentes iteraciones. Además algunos pueden expandir caminos subóptimos ($IDMOA^*$ e $IPID^*$, pero no $PIDMOA^*$). *DF-BnB* expande los nodos una única vez, pero también puede explorar ramas que conducen a soluciones subóptimas.

De igual modo, las principales similitudes y diferencias entre los enfoques de búsqueda recursiva *best-first* y los enfoques *DF-BnB* son las siguientes:

- Los algoritmos basados en *RBFS* utilizan una cota superior de expansión que es local, es decir, calculada para cada nodo, y que determina la calidad de los caminos disponibles por las ramas diferentes de la actual. Si durante la búsqueda se excede dicha cota, los algoritmos realizan *backtracking* y retoman otra rama más prometedora. Por otro lado los algoritmos *DF-BnB* usan una cota superior de tipo global, que viene dada por el conjunto de soluciones localizadas.

- Los algoritmos basados en *RBFS* generan sobrecarga computacional debida a la reexpansión de nodos y al cálculo de cotas. Además, en el caso de $IP\text{-}MO\text{-}RBFS$, puede expandir caminos subóptimos. *DF-BnB* expande los nodos una única vez, pero también puede explorar ramas que conducen a soluciones subóptimas.

El otro algoritmo secuencial contemplado en este trabajo, $IDMOA^*$, ya fue convenientemente analizado en los capítulos 3 y 4. Sería posible construir otros algoritmos secuenciales similares, empleando por ejemplo *RBFS* para cada una de las fases (objetivos). De todos modos los resultados obtenidos en los experimentos presentados en el capítulo 8 no permiten suponer unos resultados prometedores.

Hasta donde alcanza nuestra investigación, no nos consta que se haya realizado ningún estudio sistemático de rendimiento que involucre de modo general a algoritmos multiobjetivo *depth-first*. En el Capítulo 8 realizaremos una evaluación detallada de todos los algoritmos comentados, tanto sobre árboles finitos como infinitos, analizando además los parámetros de rendimiento idóneos para su correcta interpretación.

Capítulo 7

Análisis formal de los algoritmos

En los Capítulos anteriores presentamos nuevos algoritmos para la búsqueda multiobjetivo. En este capítulo formalizaremos algunas de las propiedades de estos algoritmos, concretamente su completitud y su admisibilidad. Es decir, demostraremos que estos algoritmos siempre terminan, independientemente de si el problema tiene un espacio de estados finito o infinito, y que además, cuando acaban devuelven siempre el conjunto completo de soluciones óptimas C^*.

En el caso de algunos algoritmos, como $PIDMOA^*$, $LEXIDMOA^*$ o *Pareto-MO-RBFS*, también demostraremos un resultado muy importante para su funcionamiento, y es el hecho de que únicamente localizan soluciones óptimas. En ningún caso estos algoritmos añadirán al conjunto *SOLUTION*, ni siquiera de modo temporal, una solución subóptima, lo que le permite evitar tests de dominancia extra para mantener *SOLUTION* como un conjunto de vectores no dominados.

La estructura del capítulo es la siguiente. En primer lugar en la sección 7.1 se presentarán una serie de supuestos que se deben cumplir para la demostración de las diferentes propiedades. A continuación, en la sección 7.2 se presentan una serie de consideraciones y resultados generales empleados en diferentes demostraciones. Las secciones 7.3, 7.4, 7.5, 7.6 y 7.7 demuestran las propiedades de completitud y admisibilidad para los algoritmos $PIDMOA^*$, $LEXIDMOA^*$, $IPID^*$, *Pareto-MO-RBFS* e *IP-MO-RBFS* respectivamente, así como el hecho de que algunos de ellos únicamente localizan óptimos de Pareto durante el proceso de búsqueda.

7.1. Supuestos

Para conseguir que los algoritmos sean completos y óptimos es necesario establecer una serie de supuestos de partida, por lo demás bastante habituales en la gran mayoría de problemas reales susceptibles de resolverse con búsqueda multiobjetivo. Estos supuestos son los siguientes:

1. *El grafo que representa el problema a resolver es conexo y el factor de ramificación está acotado.* Si el problema estuviera representado por varios grafos

inconexos y el estado inicial y algún estado óptimo de Pareto no coexistieran en el mismo subgrafo, los algoritmos no serían óptimos, pues no habría modo de conseguir un camino válido entre ambos nodos. Por otro lado, al acotar el factor de ramificación, éste no puede ser infinito y por lo tanto eliminamos el problema de incompletitud por no poder explorar todas las ramas de un nodo.

2. *Existe por lo menos una solución o estado final (también denominado estado o nodo meta), es decir,* $\Gamma \neq \emptyset$. No es un supuesto estrictamente necesario para el caso de espacios de estados finitos, puesto que si no hubiera nodos finales, los algoritmos devolverían un conjunto de soluciones vacío.

3. *Para cada objetivo i existe un coste positivo mínimo* ε_i *tal que para todos los costes de arista* $\vec{c}$ *entre dos nodos se cumple que* $\forall i \quad 0 < \varepsilon_i \leq c_i$. Con este supuesto el coste de un camino explorado en el árbol de búsqueda será estrictamente creciente para cualquiera de los objetivos, es decir, el pasar de un nodo n a un sucesor cualquiera incrementa el coste del camino para todos los objetivos. Con esto evitamos que pueda haber caminos cuyo coste asociado a las aristas sea el vector $\vec{0}$, ya que esto puede generar caminos infinitos que no están dominados por ningún vector de los conjuntos *SOLUTION* y *Threshold*, es decir, incompletitud de los algoritmos.

4. *Los valores heurísticos por cualquiera de los caminos* P^* *que conducen a un nodo final son todos no negativos.*

5. *La función heurística multiobjetivo* $H(n)$ *es admisible.* Una función heurística multiobjetivo $H(n)$ es admisible si para cada camino $P^* = (s,n_1, ...,n_i,n_{i+1}, ...,\gamma_k)$, $\gamma_k \in \Gamma$ (caminos que conducen a un nodo final) y cada subcamino $P_i^* = (s,n_1, ...,n_i)$ incluído en P^*, existe un vector heurístico $\vec{h} \in H(n_i)$ que verifica $\vec{g}(P_i^*) + \vec{h} \preceq \vec{g}(P^*)$. Esta condición de admisibilidad del heurístico es básica para asegurar la optimalidad de muchos algoritmos de búsqueda heurística, tales como A^*

De los supuestos 2 y 3 se deduce directamente que el conjunto de óptimos de Pareto (soluciones no dominadas) es no vacío y finito.

7.2. Consideraciones generales

Dado un camino P cualquiera a un nodo meta, $P = (s, n_1, \ldots, n_i, \ldots, n_k, \gamma)$, la estimación de coste de un subcamino P^i de P, $P^i = (s, n_1, \ldots, n_i) \subset P$ viene dada por $F(P^i) = \{\vec{g}(P^i) + \vec{h}(n_i) \quad \forall \vec{h}(n_i) \in H(n_i)\}$. Dado que n_i puede conducir a diferentes nodos meta, es posible que tenga varios vectores de estimación de coste admisibles, es decir, $\mid H(n_i) \mid \geq 1$, con lo cual la estimación de coste de P^i puede estar formada por varios vectores. Sin embargo, dado que por definición $H(\gamma) = \{\vec{0}\}$, $F(P)$ está formado

por un único vector, $\vec{F}(P) = \{\vec{g}(P) + \vec{h}(\gamma)\} = \{\vec{g}(\gamma)\}$. Por simplicidad denominaremos también a este vector de coste $\vec{f}(P)$.

A efectos de nomenclatura, dado un camino $P = (s, n_1, \ldots, n_k)$ consideraremos $F(P) = F(n_k)$, donde $\vec{g}(n_k)$ es el coste de alcanzar el nodo n_k a través del camino P.

Lema 7.2.1. *Para todo camino $P = (s, n_1, \ldots, n_i, \ldots, n_k, \gamma)$ y cada subcamino $P^i = (s, n_1, \ldots, n_i) \subseteq P$, se verifica que $\exists \vec{f} \in F(P^i) / \vec{f} \preceq \vec{f}(P)$.*

Demostración. Por definición, $F(P^i) = \{\vec{g}(n_i) + \vec{h}(n_i) \quad | \ \vec{h}(n_i) \in H(n_i)\}$ y $\vec{F}(\gamma) = \{\vec{g}(\gamma) + \vec{h}(\gamma)\} = \{\vec{g}(\gamma)\}$. Dado que $H(\gamma) = \{\vec{0}\}$, $F(P) = \{\vec{g}(P)\}$. Sea $P = P^i P^r$, con $P^r = (n_i, n_{i+1}, \ldots, \gamma)$. Como $H(n_i)$ es admisible (supuesto 5), entonces $\exists \vec{h} \in H(n_i) \quad / \quad \vec{g}(P^i) + \vec{h} \preceq \vec{g}(P^i) + \vec{g}(P^r) = \vec{g}(P) = \vec{f}(P)$. □

Lema 7.2.2. *Sea $P^* = (s, n_1, \ldots, n_i, \ldots, n_k, \gamma)$ un camino de coste óptimo $\vec{g}(P^*)$ y P^i un subcamino de P^*, $P^i = (s, n_1, \ldots, n_i) \subseteq P$. Para cualquier camino P' a un nodo meta de coste $\vec{c}$ no necesariamente óptimo se cumple que $\exists \vec{f} \in F(P^i) / (\vec{f} \preceq \vec{c}) \vee (\vec{f} \sim \vec{c})$.*

Demostración. Dado que P^* es un camino de coste óptimo, entonces se cumple que o bien $\vec{f}(P^*) = \vec{g}(P^*) \preceq \vec{c}$ o bien $\vec{f}(P^*) \sim \vec{c}$. Por el lema 7.2.1 sabemos que $\exists \vec{f} \in F(P^i) / \vec{f} \preceq \vec{f}(P^*)$, con lo cual $\exists \vec{f} \in F(P^i) / (\vec{f} \preceq \vec{c}) \vee (\vec{f} \sim \vec{c})$. □

7.3. Propiedades de $PIDMOA^*$

Lema 7.3.1. *Para cada iteración i y cada vector $\vec{t}_{i+1} \in Threshold_{i+1}$, existe un vector $\vec{t}_i \in Threshold_i$ tal que $\vec{t}_i \prec \vec{t}_{i+1}$.*

Demostración. La demostración es trivial a partir del pseudocódigo del algoritmo $PIDMOA^*$. Un vector se añade al conjunto *Threshold* de la siguiente iteración solo cuando está dominado por otro vector del conjunto *Threshold* de la iteración actual. □

Lema 7.3.2. *En un número finito de iteraciones el valor de todas las componentes de coste de cada uno de los vectores del umbral de $PIDMOA^*$ crece estrictamente.*

Demostración. El enunciado plantea la imposibilidad de que alguno de los vectores del umbral usado por $PIDMOA^*$ no crezca estrictamente en alguna de las componentes de coste. Supongamos que puede haber alguna componente del umbral que no crezca (ejemplo de la figura 7.1 para dos objetivos). En este ejemplo tenemos una secuencia infinita de umbrales que únicamente crece en la segunda componente de coste, lo cual conlleva que no se localiza el camino óptimo a γ con coste $(1, c')$. Además, el algoritmo $PIDMOA^*$ no finalizaría.

Sin embargo esta situación sólo es posible con un factor de ramificación infinito, lo cual contradice el supuesto 1 de factor de ramificación acotado. Asimismo, por el supuesto 3, al expandir un nodo, los vectores de coste de sus sucesores crecen estrictamente en todas sus componentes. Dado que son estos vectores los que se tendrán en

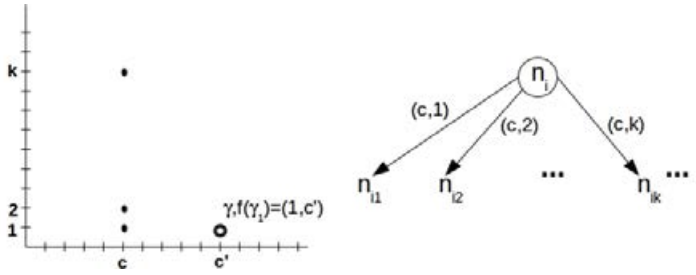


Figura 7.1: Crecimiento del *threshold* en una componente.

cuenta para el cómputo de los umbrales de las siguientes iteraciones, estos umbrales crecerán también de modo estricto en todas sus componentes.

□

Lema 7.3.3. *Dada una iteración cualquiera del algoritmo $PIDMOA^*$, con umbral $Threshold$, dicha iteración siempre termina.*

Demostración. Según el pseudocódigo del algoritmo $PIDMOA^*$ (tabla 4.1), una iteración del algoritmo termina cuando se discontinúan cada uno de los caminos de búsqueda explorados, lo cual sucede si se da alguno de los siguientes casos:

- Si en la rama explorada se localiza un nodo sin sucesores (solución o no).
- El coste del camino está dominado por alguno de los vectores que forman el umbral.

Dado que la discontinuidad en el primer caso es trivial, únicamente es necesario demostrar que para todo camino P explorado en la iteración actual y para el cual no se localice un nodo sin sucesores, todos sus vectores de coste ($F(P)$) estarán dominados por algún vector del umbral $Threshold$ en un número finito de pasos, con lo cual la iteración termina.

Sea $L_{max} = \text{máx}\{\lceil \frac{\text{máx}\{\vec{th}[1]\}}{\varepsilon_1} \rceil, \ldots, \lceil \frac{\text{máx}\{\vec{th}[k]\}}{\varepsilon_k} \rceil\} \forall \vec{th} \in Threshold$, donde k es el número de objetivos considerados, ε_i los costes mínimos positivos para cada componente i del vector de coste (supuesto 3) y $\vec{th}[i]$ son las componentes *i-ésimas* de cada uno de los vectores $\vec{th}$ que forman el umbral actual.

Dado un camino cualquiera P en el árbol de búsqueda, tenemos asegurado que al llegar a la profundidad $(L_{max}+1)$ en dicho camino, el coste del camino $P' \subseteq P$ generado, $\vec{g}(P')$, estará dominado por cualquier vector $\vec{th}$ del umbral. Esto se debe a que a profundidad $(L_{max}+1)$, $\vec{g}(P)[i] \geq (L_{max}+1) \times \varepsilon_i$.

Dado que $L_{max} \geq \lceil \frac{\text{máx}\{\vec{th}[i]\}}{\varepsilon_i} \rceil, \forall i = 1, \ldots, k \quad \forall \vec{th} \in Threshold$, $(L_{max}+1) > \lceil \frac{\text{máx}\{\vec{th}[i]\}}{\varepsilon_i} \rceil \forall \vec{th} \in Threshold$ y por lo tanto $((L_{max}+1) \times \varepsilon_i) > \text{máx}\{\vec{th}[i]\} \quad \forall \vec{th} \in Threshold$. Es decir, $\vec{g}(P)[i] > \text{máx}\{\vec{th}[i]\}, \forall i = 1, \ldots, k \quad \forall \vec{th} \in Threshold$, y por lo

tanto $\vec{th} \prec \vec{g}(P)$. Como las estimaciones heurísticas para cualquier nodo son no negativas (supuesto 4), también se verifica que $\forall \vec{f} \in F(P), \forall i, \text{máx}\{\vec{th}[i]\} < \vec{f}[i] \quad \forall \vec{th} \in Threshold$ y por lo tanto $\forall \vec{f} \in F(P), \vec{th} \prec \vec{f}$, y la búsqueda se discontinúa en el camino P.

□

Lema 7.3.4. *$PIDMOA^*$ encuentra cada solución no dominada P^* en un número finito de pasos.*

Demostración. Supongamos que $PIDMOA^*$ no localiza una solución no dominada P^* de coste $\vec{g}(P^*)$. Por el lema 7.2.1 sabemos que $\forall P^i \subseteq P^* \quad \exists \vec{f} \in F(P^i)/\vec{f} \preceq \vec{g}(P^*)$. Sea $\vec{c}$ el coste de un camino solución cualquiera P' localizado por $PIDMOA^*$. Por el lema 7.2.2 se cumple que $\exists \vec{f} \in F(P^i)/(\vec{f} \preceq \vec{c}) \vee (\vec{f} \sim \vec{c})$.

Por lo tanto cualquier subcamino de P^* nunca será discontinuado por estar dominado por alguna solución localizada por $PIDMOA^*$. La exploración de P^* sólo puede discontinuarse si el coste de este camino está dominado por algún vector del umbral. Ya que siempre hay un subcamino $P^i \subseteq P^*$ con algún vector de coste no dominado por ningún vector del umbral actual antes de que P^* se encuentre, la búsqueda no puede terminar sin encontrar P^*.

La única posibilidad es que haya una exploración infinita de caminos antes de alcanzar un umbral que incluya el vector $\vec{g}(P^*)$. Sin embargo, al existir un coste mínimo ε_i por cada componente de coste, sólo puede haber un número finito de umbrales no dominados por $\vec{g}(P^*)$ y, por los lemas 7.3.1, 7.3.2 y 7.3.3 se exploran en tiempo finito.

□

Teorema 7.3.1. *Si existe solución, entonces $PIDMOA^*$ es admisible.*

Demostración. Únicamente es necesario demostrar que $PIDMOA^*$ termina si hay solución, puesto que el lema 7.3.4 muestra que $PIDMOA^*$ localiza todas las soluciones óptimas. Para ello es suficiente con demostrar que hay un número finito de caminos no dominados por estas soluciones óptimas.

Sea $\vec{f}_{max} = (c_1, \ldots, c_k), c_i = \text{máx}\{\vec{f}(\gamma_1)[i], \ldots, \vec{f}(\gamma_n)[i]\} \, \forall (\gamma_j, \vec{f}(\gamma_j)) \in SOLUTION$, con $j = 1, \ldots, n$. Es decir, $\vec{f}_{max}$ contiene el mayor valor para cada componente de coste de los diferentes óptimos de Pareto. Por el supuesto 2 tenemos garantizado que al menos habrá una solución.

Sea $L_{max} = \text{máx}\{\lceil \frac{\text{máx}\{\vec{th}[1]\}}{\varepsilon_1} \rceil, \ldots, \lceil \frac{\text{máx}\{\vec{th}_j[k]\}}{\varepsilon_k} \rceil\} \quad \forall \vec{th} \in Threshold$, ya definido en la prueba del lema 7.3.3. Tenemos asegurado que en un máximo de $(L_{max} + 1)$ expansiones cualquier camino estará dominado for $\vec{f}_{max}$, y por lo tanto por cualquier solución óptima. Por el lema 7.3.3, todas las iteraciones terminan, y por los lemas 7.3.1 y 7.3.3 sabemos que los vectores del umbral crecen estrictamente en un número finito de pasos. Por ello también lo hará la profundidad de los caminos explorados, hasta un máximo de $(L_{max} + 1)$ expansiones, discontinuándose todos los caminos abiertos

(al estar dominados por alguna solución) y quedar vacío el umbral de la siguiente iteración. □

Lema 7.3.5. *$PIDMOA^*$ no localiza soluciones dominadas.*

Demostración. Supongamos que existe una solución óptima P^* y otra solución subóptima P' tal que $\vec{f}(P^*) \prec \vec{f}(P')$. Si $PIDMOA^*$ hubiera localizado P^*, entonces P' nunca se incluirá en el conjunto *SOLUTION*, ya que el algoritmo $PIDMOA^*$ (tabla 4.1), antes de expandir un camino, comprueba si está dominado por alguna solución ya localizada, en cuyo caso descarta dicho camino. Supongamos que $PIDMOA^*$ ha localizado la solución subóptima P', pero no la óptima P^*. Entonces,

$$\forall \vec{t} \in Threshold, P' \sim \vec{t} \vee P' \prec \vec{t} \tag{7.1}$$

Pero si $PIDMOA^*$ no ha localizado P^*, entonces existe un vector umbral $\vec{t} \in Threshold$ tal que $\vec{t} \prec \vec{f}(P^*)$. Así que existe un vector umbral $\vec{t} \in Threshold$ tal que $\vec{t} \prec \vec{f}(P^*) \prec \vec{f}(P^*)$, lo cual contradice la ecuación 7.1. □

7.4. Propiedades de $LEXIDMOA^*$

Dado un conjunto de vectores $X = \{x_1, \ldots, x_n\}$, el menor lexicográfico de dicho conjunto es el vector x_i tal que $x_i <_{lex} x_j, \forall j, 1 < j < n, i \neq j$.

Lema 7.4.1. *Sean dos vectores $\vec{x}, \vec{y} \in \mathbb{R}^q$. Se verifica que:*

$$\vec{x} \not<_{lex} \vec{y} \Leftrightarrow \vec{y} <_{lex} \vec{x} \tag{7.2}$$

Lema 7.4.2. *Sean dos vectores $\vec{x}, \vec{y} \in \mathbb{R}^q$. Se verifica que:*

$$\vec{x} \preceq \vec{y} \Rightarrow \vec{x} \leq_{lex} \vec{y} \tag{7.3}$$

Los dos lemas anteriores se demuestran de modo trivial a partir de las definiciones de dominancia y de orden lexicográfico (ver Capítulo 3). Asimismo se deduce claramente que el menor lexicográfico de un conjunto de vectores es un vector no dominado en dicho conjunto.

Lema 7.4.3. *Para todo camino $P = (s, n_1, \ldots, n_i, \ldots, n_k, \gamma)$ y cada subcamino $P^i = (s, n_1, \ldots, n_i) \subseteq P$, se verifica que $\exists \vec{f} \in F(P^i) / \vec{f} \leq_{lex} \vec{f}(P)$.*

Demostración. Por el lema 7.2.1 se cumple que $\exists \vec{f} \in F(P^i) / \vec{f} \preceq \vec{f}(P)$. Combinando este resultado con el lema 7.4.2 se cumple que $\vec{f} \leq_{lex} \vec{f}(P)$. □

Lema 7.4.4. *Para cada iteración $i > 1$, el umbral de dicha iteración es mejor lexicográfico que el de la iteración siguiente, es decir, $\vec{threshold}_i <_{lex} \vec{threshold}_{i+1}$. Por lo tanto cada nueva iteración explora al menos un camino nuevo respecto a la iteración anterior.*

Demostración. Por construcción del algoritmo $LEXIDMOA^*$, durante la iteración i los únicos vectores de coste que se usan para el cómputo del umbral de la siguiente iteración ($\vec{threshold}_{i+1}$) son aquellos para los cuales el umbral de la iteración i ($\vec{threshold}_i$) es menor lexicográfico. Dado que el umbral $\vec{threshold}_{i+1}$ siempre será uno de estos vectores (el menor lexicográfico), el resultado es trivial. □

Lema 7.4.5. *En un número finito de iteraciones el valor de todas las componentes de coste del vector umbral de $LEXIDMOA^*$ crece estrictamente.*

Demostración. El enunciado plantea la imposibilidad de que el umbral no crezca estrictamente en alguna de las componentes de coste. Supongamos que puede haber alguna componente del umbral que no crezca (ejemplo de la figura 7.1 para dos objetivos). En este ejemplo tenemos una secuencia infinita de umbrales que únicamente crece en la segunda componente de coste, lo cual conlleva que no se localiza el camino óptimo a γ con coste $(1, c')$. Además, el algoritmo $LEXIDMOA^*$ no finalizaría.

Sin embargo esta situación sólo es posible con un factor de ramificación infinito, lo cual contradice el supuesto 1 de factor de ramificación acotado. Asimismo, por el supuesto 3, al expandir un nodo, los vectores de coste de sus sucesores crecen estrictamente en todas sus componentes. Dado que son estos vectores los que se tendrán en cuenta para el cómputo de los umbrales de las siguientes iteraciones, estos umbrales crecerán también de modo estricto en todas sus componentes.

□

Lema 7.4.6. *Dada una iteración cualquiera del algoritmo $LEXIDMOA^*$, con umbral $\vec{th} = (c_1, \ldots, c_k)$, dicha iteración siempre termina.*

Demostración. Según el pseudocódigo del algoritmo $LEXIDMOA^*$ (tabla 4.2), una iteración del algoritmo termina cuando se discontinúan cada uno de los caminos de búsqueda explorados, lo cual sucede si se da alguno de los siguientes casos:

- Si en la rama explorada se localiza un nodo sin sucesores (solución o no).
- El coste del camino es peor lexicográfico que el umbral.

Dado que la discontinuidad en el primer caso es trivial, únicamente es necesario demostrar que para todo camino P explorado en la iteración actual y para el cual no se localice un nodo sin sucesores, todos sus vectores de coste ($F(P)$) serán peores lexicográficos que el umbral $\vec{th}$ en un número finito de pasos, con lo cual la iteración termina.

Sea $L_{max} = \text{máx}\{\lceil \frac{c_1}{\varepsilon_1} \rceil, \ldots, \lceil \frac{c_k}{\varepsilon_k} \rceil\}$, donde k es el número de objetivos considerados, ε_i los costes mínimos positivos para cada componente i del vector de coste (supuesto 3) y c_i las componentes de coste del umbral actual.

Dado un camino cualquiera P en el árbol de búsqueda, tenemos asegurado que al llegar a la profundidad $(L_{max} + 1)$ en dicho camino, el coste del camino $P' \subseteq$

P generado, $\vec{g}(P')$, será peor lexicográfico que el umbral $t\vec{h}$. Esto se debe a que a profundidad $(L_{max}+1)$, $\vec{g}(P)[i] \geq (L_{max}+1) \times \varepsilon_i$.

Dado que $L_{max} \geq \lceil \frac{c_i}{\varepsilon_i} \rceil, \forall i = 1, \ldots, k$, $(L_{max}+1) > \lceil \frac{c_i}{\varepsilon_i} \rceil$ y por lo tanto $((L_{max}+1) \times \varepsilon_i) > c_i$. Es decir, $\vec{g}(P)[i] > c_i, \forall i = 1, \ldots, k$, y por lo tanto $t\vec{h} \prec \vec{g}(P)$. Como las estimaciones heurísticas para cualquier nodo son no negativas (supuesto 4), también se verifica que $\forall \vec{f} \in F(P), \forall i, c_i < \vec{f}[i]$ y por lo tanto $\forall \vec{f} \in F(P), t\vec{h} \prec \vec{f}$. Por el lema 7.4.1 deducimos que $\forall \vec{f} \in F(P), t\vec{h} <_{lex} \vec{f}$, y por lo tanto la búsqueda se discontinúa en el camino P.

□

Lema 7.4.7. *$LEXIDMOA^*$ encuentra cada solución no dominada P^* en un número finito de pasos.*

Demostración. Supongamos que $LEXIDMOA^*$ no localiza una solución no dominada P^* de coste $\vec{g}(P^*)$. Por el lema 7.2.1 sabemos que $\forall P^i \subseteq P^* \quad \exists \vec{f} \in F(P^i) / \vec{f} \preceq \vec{g}(P^*)$. Sea $\vec{c}$ el coste de un camino solución cualquiera P' localizado por $LEXIDMOA^*$. Por el lema 7.2.2 se cumple que $\exists \vec{f} \in F(P^i) / (\vec{f} \preceq \vec{c}) \vee (\vec{f} \sim \vec{c})$.

Por lo tanto cualquier subcamino de P^* nunca será discontinuado por estar dominado por alguna solución localizada por $LEXIDMOA^*$. La exploración de P^* sólo puede discontinuarse porque el coste de este camino sea peor lexicográfico que el umbral. Ya que siempre hay un subcamino $P^i \subseteq P^*$ con algún vector de coste mejor lexicográfico o igual que el umbral antes de que P^* se encuentre (ver lema 7.4.3), la búsqueda no puede terminar sin encontrar P^*.

La única posibilidad es que haya una exploración infinita de caminos antes de alcanzar el umbral $t\vec{h} = \vec{g}(P^*)$. Sin embargo, al existir un coste mínimo ε_i por cada componente de coste, sólo puede haber un número finito de umbrales mejores lexicográficos que $\vec{g}(P^*)$ y, por los lemas 7.4.4, 7.4.5 y 7.4.6 se exploran en tiempo finito.

□

Teorema 7.4.1. *Si existe solución, entonces $LEXIDMOA^*$ es admisible.*

Demostración. Únicamente es necesario demostrar que $LEXIDMOA^*$ termina si hay solución, puesto que el lema 7.4.7 muestra que $LEXIDMOA^*$ localiza todas las soluciones óptimas. Para ello es suficiente con demostrar que hay un número finito de caminos no dominados por estas soluciones óptimas.

Sea $\vec{f}_{max} = (c_1, \ldots, c_k), c_i = \text{máx}\{\vec{f}(\gamma_1)[i], \ldots, \vec{f}(\gamma_n)[i]\}\ \forall (\gamma_j, \vec{f}(\gamma_j)) \in SOLUTION$, con $j = 1, \ldots, n$. Es decir, $\vec{f}_{max}$ contiene el mayor valor para cada componente de coste de los diferentes óptimos de Pareto. Por el supuesto 2 tenemos garantizado que al menos habrá una solución.

Sea $L_{max} = \text{máx}\{\lceil \frac{c_1}{\varepsilon_1} \rceil, \ldots, \lceil \frac{c_k}{\varepsilon_k} \rceil\}$, ya definido en la prueba del lema 7.4.6. Tenemos asegurado que en un máximo de $(L_{max}+1)$ expansiones cualquier camino estará dominado for $\vec{f}_{max}$, y por lo tanto por cualquier solución óptima. Por el lema 7.4.6, todas las iteraciones terminan, y por los lemas 7.4.4 y 7.4.6 sabemos que los umbrales

crecen de modo lexicográfico y, en un número finito de iteraciones avanzan en todas las componentes de coste. Por ello, al crecer el umbral, también lo hará la profundidad de los caminos explorados, hasta un máximo de $(L_{max}+1)$ expansiones, discontinuándose todos los caminos abiertos (al estar dominados por alguna solución) y quedar vacío el umbral de la siguiente iteración. □

Corolario 7.4.1. *Cuando $LEXIDMOA^*$ encuentra una solución P^*, el umbral de la iteración actual, $\vec{th}$, verifica que $\vec{th} = \vec{g}(P^*)$.*

Demostración. Si $LEXIDMOA^*$ localiza la solución P^*, entonces el camino ha sido expandido. Esto sólo es posible si $\vec{th} \not<_{lex} \vec{g}(P^*)$. Por lo tanto, o bien $\vec{g}(P^*) <_{lex} \vec{th}$ o bien $\vec{g}(P^*) = \vec{th}$. El caso $\vec{g}(P^*) <_{lex} \vec{th}$ no es posible, ya que entonces $\vec{g}(P^*)$ se hubiera seleccionado como umbral en alguna iteración anterior, en la cual habría sido localizada la solución P^*. Por lo tanto la única posibilidad es que $\vec{g}(P^*) = \vec{th}$. □

Lema 7.4.8. *$LEXIDMOA^*$ no localiza soluciones dominadas.*

Demostración. Sea P' una solución dominada por una solución óptima P^*, *i.e.*, $\vec{g}(P^*) \prec \vec{g}(P') \Rightarrow \vec{g}(P^*) <_{lex} \vec{g}(P')$.

Por el lema 7.4.4 y el corolario 7.4.1, P^* se encontrará antes que P', y por lo tanto, una vez encontrada la solución P^*, la búsqueda a lo largo de P' se discontinuará en todas las iteraciones. □

7.5. Propiedades de $IPID^*$

Lema 7.5.1. *Para cada iteración i, $\vec{th}_i \ll \vec{th}_{i+1}$.*

Demostración. Supongamos que sucede lo contrario, es decir, existe una componente j tal que $\vec{th}_{i+1}[j] \leq \vec{th}_i[j]$. Sin embargo $\vec{th}_{i+1}$ es el punto ideal del conjunto T que contiene los vectores de coste de todos los nodos en los que la búsqueda fue discontinuada en el paso i. Por definición de $IPID^*$, para cada vector $\vec{t} \in T$ y cada componente j tenemos $\vec{t}[j] > \vec{th}_i[j]$ y por lo tanto $\vec{th}_{i+1}[j] > \vec{th}_i[j]$, lo cual es una contradicción. Por consiguiente $\vec{th}_i \ll \vec{th}_{i+1}$.

Además, por definición de la relación *estrictamente mejor*, este resultado nos permite asegurar que de una iteración a la siguiente, todas las componentes del vector umbral crecen estrictamente. □

Lema 7.5.2. *Dada una iteración cualquiera del algoritmo $IPID^*$, con umbral $\vec{th}$, dicha iteración siempre termina.*

Demostración. Según el pseudocódigo del algoritmo $IPID^*$ (tabla 4.3), una iteración del algoritmo termina cuando se discontinúan cada uno de los caminos de búsqueda explorados, lo cual sucede si se da alguno de los siguientes casos:

- Si en la rama explorada se localiza un nodo sin sucesores (solución o no).
- El vector umbral es estrictamente mejor que cualquiera de los vectores de coste del camino considerado.

Dado que la discontinuidad en el primer caso es trivial, únicamente es necesario demostrar que para todo camino P explorado en la iteración actual y para el cual no se localice un nodo sin sucesores, todos sus vectores de coste ($F(P)$) serán estrictamente peores que el umbral $\vec{th}$ en un número finito de pasos, con lo cual la iteración termina.

Sea $L_{max} = \text{máx}\{\lceil \frac{c_1}{\varepsilon_1} \rceil, \ldots, \lceil \frac{c_k}{\varepsilon_k} \rceil\}$, donde k es el número de objetivos considerados, ε_i los costes mínimos positivos para cada componente i del vector de coste (supuesto 3) y c_i es la componente *i-ésima* del umbral actual.

Dado un camino cualquiera P en el árbol de búsqueda, tenemos asegurado que al llegar a la profundidad $(L_{max}+1)$ en dicho camino, el coste del camino $P' \subseteq P$ generado, $\vec{g}(P')$, será *estrictamente peor* que el vector umbral $\vec{th}$. Esto se debe a que a profundidad $(L_{max}+1)$, $\vec{g}(P)[i] \geq (L_{max}+1) \times \varepsilon_i$.

Dado que $L_{max} \geq \lceil \frac{c_i}{\varepsilon_i} \rceil, \forall i = 1, \ldots, k$, $(L_{max}+1) > \lceil \frac{c_i}{\varepsilon_i} \rceil$ y por lo tanto $((L_{max}+1) \times \varepsilon_i) > c_i$. Es decir, $\vec{g}(P)[i] > c_i, \forall i = 1, \ldots, k$, y por lo tanto $\vec{th} \ll \vec{g}(P)$. Como las estimaciones heurísticas para cualquier nodo son no negativas (supuesto 4), también se verifica que $\forall \vec{f} \in F(P), \forall i, c_i < \vec{f}[i]$ y por lo tanto $\forall \vec{f} \in F(P), \vec{th} \ll \vec{f}$, y la búsqueda se discontinúa en el camino P.

□

Lema 7.5.3. *$IPID^*$ encuentra cada solución no dominada P^* en un número finito de pasos.*

Demostración. Supongamos que $IPID^*$ no localiza una solución no dominada P^* de coste $\vec{g}(P^*)$. Por el lema 7.2.1 sabemos que $\forall P^i \subseteq P^* \quad \exists \vec{f} \in F(P^i)/\vec{f} \preceq \vec{g}(P^*)$. Sea $\vec{c}$ el coste de un camino solución cualquiera P' localizado por $IPID^*$. Por el lema 7.2.2 se cumple que $\exists \vec{f} \in F(P^i)/(\vec{f} \preceq \vec{c}) \vee (\vec{f} \sim \vec{c})$.

Por lo tanto cualquier subcamino de P^* nunca será discontinuado por estar dominado por alguna solución localizada por $IPID^*$. La exploración de P^* sólo puede discontinuarse porque el coste de este camino sea estrictamente peor que el umbral. Supongamos que $IPID^*$ finaliza sin encontrar la solución P^*. Sea $\vec{th}_{final}$ el umbral de la última iteración. Si la búsqueda se ha discontinuado en un subcamino $P' \subseteq P^*$, entonces $\forall \vec{f} \in F(P'), \vec{th}_{final} \ll \vec{f}$. Pero entonces $IPID^*$ usaría estos vectores en el cálculo del punto ideal que servirá de umbral en la siguiente iteración; así que dicho conjunto no está vacío e $IPID^*$ no debería haber finalizado su ejecución.

La única posibilidad de que no se encuentre P^* es que haya un número infinito de umbrales estrictamente mejores que $\vec{g}(P^*)$. Sin embargo, esto no es posible al existir un coste mínimo ε_i por cada componente de coste; el número de umbrales estrictamente mejores que $\vec{g}(P^*)$ es finito y, por los lemas 7.5.1 y 7.5.2 se exploran en tiempo finito.

□

Teorema 7.5.1. *Si existe solución, entonces $IPID^*$ es admisible.*

Demostración. Únicamente es necesario demostrar que $IPID^*$ termina si hay solución, puesto que el lema 7.5.3 muestra que $IPID^*$ localiza todas las soluciones óptimas.

Para ello es suficiente con demostrar que hay un número finito de caminos no dominados por estas soluciones óptimas. Sea $\vec{f}_{max} = (c_1, \ldots, c_k)$, donde cada componentes c_i toma el valor $c_i = \text{máx}\{\vec{f}(\gamma_1)[i], \ldots, \vec{f}(\gamma_n)[i]\}\ \forall(\gamma_j, \vec{f}(\gamma_j)) \in SOLUTION, j = 1, \ldots, n$. Es decir, $\vec{f}_{max}$ contiene el mayor valor para cada componente de coste de los diferentes óptimos de Pareto. Por el supuesto 2 tenemos garantizado que al menos habrá una solución.

Sea $L_{max} = \text{máx}\{\lceil \frac{c_1}{\varepsilon_1} \rceil, \ldots, \lceil \frac{c_k}{\varepsilon_k} \rceil\}$, ya definido en la prueba del lema 7.5.2. Tenemos asegurado que en un máximo de $(L_{max} + 1)$ expansiones cualquier camino estará dominado for $\vec{f}_{max}$, y por lo tanto por cualquier solución óptima. Por el lema 7.5.2, todas las iteraciones terminan, y por los lemas 7.5.1 y 7.5.2 sabemos que los umbrales crecen estrictamente en todas las componentes de coste. Por ello, al crecer el umbral, también lo hará la profundidad de los caminos explorados, hasta un máximo de $(L_{max} + 1)$ expansiones, discontinuándose todos los caminos abiertos (al estar dominados por alguna solución) y quedar vacío el umbral de la siguiente iteración. □

7.6. Propiedades de *Pareto-MO-RBFS*

Antes de demostrar la completitud y admisibilidad del algoritmo *Pareto-MO-RBFS* definiremos los siguientes términos, análogamente a los definidos por (Korf, 1993):

- Los conjuntos SOL y C_{sol} son los mismos descritos para *Pareto-MO-RBFS* e *IP-MO-RBFS* en la sección 5.2.1.
- $T(n, C_{sup}, SOL)$ es el subárbol bajo el nodo n formado por nodos interiores y nodos frontera. Sea $F(n)$ el conjunto $\{\vec{f} = \vec{g}(n) + \vec{h} \mid \vec{h} \in H(n)\}$. Los nodos interiores n' son aquellos que tienen al menos un vector de coste $\vec{f} \in F(n')$ no dominado por ningún vector del conjunto $(C_{sup} \cup C_{sol})$. Los nodos frontera m son aquellos que verifican que $\forall \vec{f} \in F(m), \exists \vec{v} \in (C_{sup} \cup C_{sol}) \mid \vec{v} \prec \vec{f}$. Además ningún sucesor de un nodo frontera forma parte del árbol, y todos los sucesores de un nodo interior están en el árbol.
- $D(n, C_{sup}, SOL)$ es la profundidad máxima del árbol $T(n, C_{sup}, SOL)$.
- $PF(n, C_{sup}, SOL)$ es la frontera de Pareto del árbol $T(n, C_{sup}, SOL)$, es decir, el conjunto de vectores de los conjuntos $F(n)$ no dominados entre sí de los nodos frontera del árbol $T(n, C_{sup}, SOL)$ tal que no son dominados por ningún vector de C_{sol}.
- Un nodo abierto es aquel que ha sido generado al menos una vez, pero no ha sido expandido.

- $OD(n)$ es el conjunto de vectores de coste no dominados (la frontera de Pareto) de todos los nodos abiertos *descendientes* de n.
- $ON(n)$ es el conjunto de vectores de coste no dominados (la frontera de Pareto) de todos los nodos abiertos *no descendientes* de n.

Lema 7.6.1. *Todas las llamadas Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL) cumplen que* $\exists \vec{f} \in FA_n \mid \nexists \vec{v} \in (C_{sup} \cup C_{sol}), \vec{v} \prec \vec{f}$.

Demostración. Para la llamada inicial se cumple de modo trivial, pues $C_{sup} = \{\vec{\infty}\}$ y $FA_n = nodomset(H(s))$. Consideremos un nodo n' sucesor de n, y una llamada *Pareto-MO-RBFS (n', $FA_{n'}$, C'_{sup}, SOL')* recursiva desde *Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL)*. La propiedad se cumple por construcción de los valores del conjunto FA' (conjunto $N1_{sc}$) y $C'_{sup} = C_{sup} \cup (\bigcup_{i \neq 1} FA[i])$. Del conjunto $N1_{sc}$ se han eliminado todos los vectores dominados por algún vector de C_{sol} y C_{sup}. Además, al haber sido seleccionado el nodo para expansión, sabemos que en FA' existe al menos un vector $\vec{v} \in N_{suc}$, y por lo tanto no dominado por $\bigcup_{i \neq 1} FA[i]$. □

Lema 7.6.2. *Si C_{sup} es finito y $T(n, C_{sup}, SOL)$ no contiene ningún nodo solución, entonces la llamada Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL) explora el árbol $T(n, C_{sup}, SOL)$ y devuelve, además del propio conjunto SOL:*

- $\vec{\infty}$ *si todas las ramas exploradas terminan sin sucesores o si son dominadas por el coste de alguna solución en C_{sol} o de algún vector en la cota superior C_{sup}, ó*
- $PF(n, C_{sup}, SOL)$ *en otro caso.*

Demostración. Por el supuesto 3, si el conjunto $(C_{sup} \cup C_{sol})$ tiene al menos un vector finito en todas sus componentes, entonces el árbol $T(n, C_{sup}, SOL)$ es finito, ya que tenemos asegurado que en un número finito de pasos, cualquier camino explorado tendrá un coste dominado estrictamente por cualquier vector del conjunto $C_{sup} \cup C_{sol}$.

Sea $L_{max} = \max\{\lceil \frac{\max\{\vec{cs}_j[1]\}}{\varepsilon_1} \rceil, \ldots, \lceil \frac{\max\{\vec{cs}_j[k]\}}{\varepsilon_k} \rceil\} \quad \forall \vec{cs}_j \in C_{sup}$, donde k es el número de objetivos considerados, ε_i los costes mínimos positivos para cada componente i del vector de coste (supuesto 3) y $\vec{cs}_j[i]$ las componentes *i-ésimas* de cada uno de los vectores $\vec{cs}_j$ que forman la cota superior C_{sup}. Dado un camino cualquiera P en el subárbol explorado, tenemos asegurado que al llegar a la profundidad $(L_{max} + 1)$ en dicho camino, el coste del mismo, $\vec{g}(P)$, estará dominado por cualquier vector $\vec{cs}_j$ de la cota superior. Esto se debe a que a profundidad $(L_{max} + 1)$, $\forall i \quad \vec{g}(P)[i] \geq (L_{max} + 1) \times \varepsilon_i$.

Dado que $T(n, C_{sup}, SOL)$ es finito, también lo será la profundidad máxima de dicho árbol, es decir, $D(n, C_{sup}, SOL)$.

Demostraremos el lema por inducción sobre la profundidad máxima del árbol considerado, $T(n, C_{sup}, SOL)$, el cual es finito.

Caso base. En el caso base, $D(n, C_{sup}, SOL) = 0$, n es el único nodo del árbol, no llegándose a explorar ninguno de sus hijos. Esto implica que la llamada *Pareto-MO-*

RBFS (n, FA_n, C_{sup}, SOL) finaliza en alguna de las cuatro primeras comprobaciones realizadas en la función *Pareto-MO-RBFS*.

En la primera comprobación, si $\forall \vec{f} \in F(n), \exists \vec{c^*} \in C_{sol} \mid \vec{c^*} \prec \vec{f}$, entonces el algoritmo devuelve $\vec{\infty}$.

Si por el contrario existen vectores de coste $\vec{f} \in F(n)$ no dominados por el coste de ninguna solución, y dichos vectores (almacenados en la variable N_{sol}) están todos dominados por algún vector de la cota superior, es decir, $\forall \vec{f} \in N_{sol}, \exists \vec{c} \in C_{sup} \mid \vec{c} \prec \vec{f}$, entonces la llamada devuelve precisamente $PF(n, C_{sup}, SOL)$.

La tercera comprobación no es aplicable, al suponer que en el árbol explorado no existe ninguna solución. En la cuarta comprobación, si el nodo considerado no tiene sucesores, entonces la llamada devuelve $\vec{\infty}$. Por lo tanto el lema se verifica para un árbol de profundidad cero.

Hipótesis de inducción. Por hipótesis de inducción, suponemos que para toda llamada *Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL)* donde $D(n, C_{sup}, SOL) \leq k$ el lema se cumple.

Paso de inducción. Supongamos una llamada *Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL)* que cumple las condiciones y $D(n, C_{sup}, SOL) = k + 1$. Todas las llamadas recursivas a *Pareto-MO-RBFS (n_i, FA_{n_i}, C_{sup_i}, SOL)*, con $n_i \in Sucessors(n)$, van a cumplir la propiedad, ya que $C_{sup_i} = nodomset(C_{sup} \cup (\bigcup_{j \neq i} FA_{n_j}))$ y, por lo tanto, si un vector $\vec{v}$ está dominado por algún vector de C_{sup}, entonces también estará dominado por algún vector de C_{sup_i}.

Es decir, $T(n_i, C_{sup_i}, SOL)$ es un subárbol de $T(n, C_{sup}, SOL)$ y por lo tanto $D(n_i, C_{sup_i}, SOL) < D(n, C_{sup}, SOL)$. Como n_i es hijo directo de n y $D(n, C_{sup}, SOL) = k + 1$, entonces tenemos que $D(n_i, C_{sup_i}, SOL) \leq k$.

Por el lema 7.6.1 sabemos que en $FA[1] = N1_{sc}$ existe al menos un vector indiferente con $FH = \bigcup_{i \neq 1} FA[i]$ (la frontera de Pareto definida por los hermanos del nodo considerado), y no dominado por C_{sup} (la frontera de Pareto definida por los nodos no descendientes del nodo considerado) ni C_{sol}. Además sabemos que todos los vectores devueltos por la llamada *Pareto-MO-RBFS (n', $FA_{n'}$, C'_{sup}, SOL)* estarán dominados por $C'_{sup} = C_{sup} \cup FH$.

Pasaremos a demostrar que en cada llamada el conjunto N_{suc}, definido como $nodomset(\bigcup_i FA[i])$, progresa o avanza de algún modo. Sea A_1 el (hiper)área dominada por $C_{sup} \cup N_{suc}$, y que define la frontera de la búsqueda antes de la llamada. Sea A_2 el (hiper)área dominada por $C_{sup} \cup FH$. Los nuevos vectores de $FA[1]$, $FA[1]'$, devueltos por la llamada recursiva, estarán dominados por $C_{sup} \cup FH$. Puesto que $N_{suc} = nodomset(FA[1]' \cup FH)$, tenemos que $A_2 \subseteq A_1$.

- Supongamos que para algún vector $\vec{f} \in (FA[1]' \cap N_{suc})$, $\vec{f} \notin FH$. Este caso, representado en la figura 7.2a, muestra como el vector sustituido en $\vec{f} \in FA[1]$, dominado por la cota superior actual, es sustituido por otro al cual domina (en el caso de un heurístico monótono) o incluso por otro vector indiferente con $\vec{f}$ (en el caso de un heurístico no monótono). De cualquier modo, tal y como se

refleja en la figura, el área dominada se reduce y por lo tanto $A_2 \subset A_1$, con lo cual la frontera avanza.

- Supongamos ahora que para algún vector $\vec{f} \in (FA[1] \cap N_{suc})$, $\vec{f} \in FH$. Es el caso representado en la figura 7.2b, donde el nodo expandido n_1 tiene un vector de coste igual al de su hermano b_3. En este caso, aunque el vector de coste de n_1 avance estrictamente en al menos una de sus componentes, moviéndose al área descartada, al emplearse también el coste del nodo b_3 para calcular la cota superior, ésta permanece invariante, y por lo tanto $A_2 = A_1$, si bien $\vec{f}$ ya no formará parte de $FA[1]$. Al haber un factor de ramificación acotado, el nodo expandido tendrá un número finito m de hermanos. Cada vez que se expanda un hermano, el valor $\vec{f}$ desaparecerá de su conjunto FA, y en un número finito m de tales llamadas volveremos a tener en el peor caso nuevamente $A_2 \subset A_1$.

Por tanto, la secuencia de valores de N_{suc} (la frontera de Pareto definida por los sucesores de n) es monótona no decreciente, y en un número finito de pasos será dominada por C_{sup} y la llamada terminará. En el conjunto FA se han introducido vectores que empeoran estrictamente el valor de alguna de sus componentes con respecto al conjunto de vectores previo. Dado que el conjunto C_{sup} permanece invariante entre llamadas, y dado que existe un coste mínimo por componente (supuesto 3), habrá un número finito de llamadas antes de que todo vector del conjunto FA esté dominado por algún vector de la cota superior original, C_{sup}, momento en el cual la llamada actual finaliza, devolviendo $PF(n_i, C_{sup}, SOL)$.

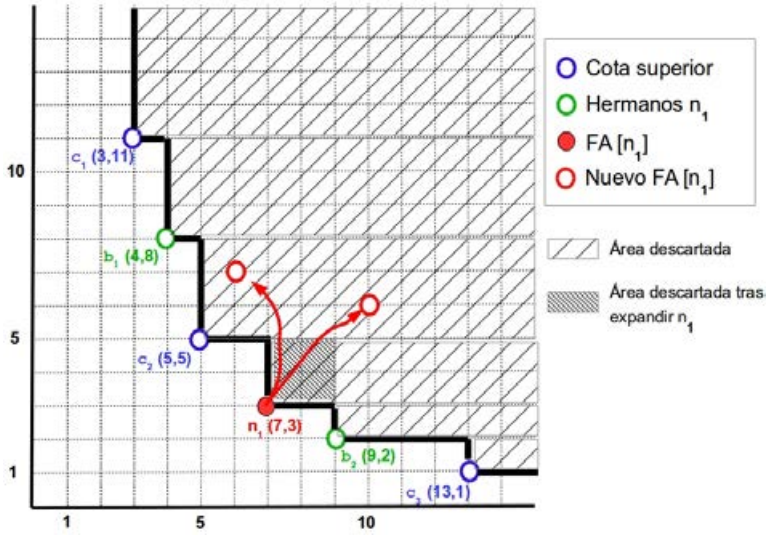
En este punto está demostrado que para todo nodo n_i sucesor de n, la llamada *Pareto-MO-RBFS (n_i, FA_{n_i}, C_{sup_i}, SOL)* devuelve $PF(n_i, C_{sup_i}, SOL)$ en un número finito de pasos. Ahora es necesario demostrar que la consideración de las fronteras de Pareto devueltas por todos los sucesores de n hace que se cumpla la propiedad para la llamada *Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL)*.

Pero esto es trivial, puesto que al no haber solución en el árbol $T(n, C_{sup}, SOL)$, todos los sucesores de n se expanden en la última llamada con $C_{sup_i} = C_{sup}$ y exploran $T(n_i, C_{sup}, SOL)$, devolviendo $PF(n_i, C_{sup}, SOL)$. Pero el conjunto no dominado de $\bigcup_{n_i} PF(n_i, C_{sup}, SOL), n_i \in Sucessors(n)$ es, por definición, $PF(n, C_{sup}, SOL)$. □

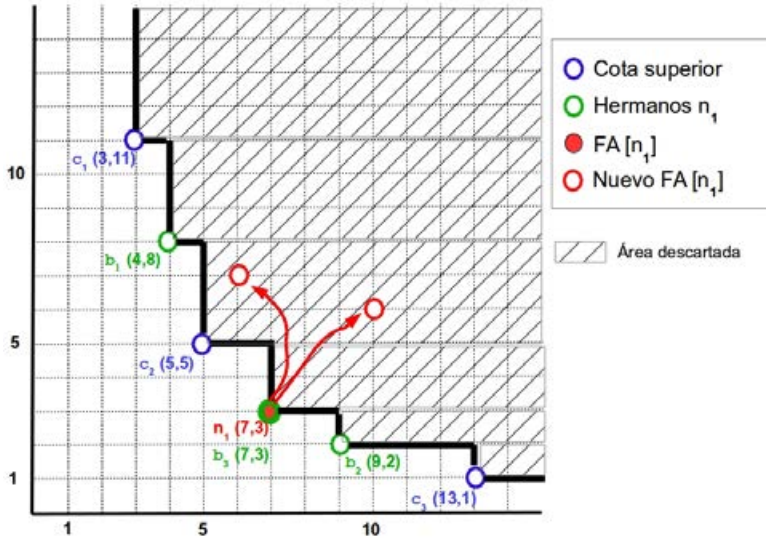
Lema 7.6.3. *Para toda llamada Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL) se cumplen las siguientes propiedades*[1]*:*

- $FA_n \preceq_I OD(n)$.
- $C_{sup} \preceq_I ON(n)$.

[1]Tal y como se comentaba en el Capítulo 5, se emplea la notación $\vec{a} \preceq_I \vec{b}$ para denotar la expresión $(\vec{a} \preceq \vec{b}) \vee (\vec{a} \sim \vec{b})$. Asimismo aplicaremos las operaciones habituales sobre vectores ($\sim$, $\prec$, $\preceq$, $\ll$, $\not\ll$, $\preceq_I$) a conjuntos, de tal modo que A *op* B denota $\forall \vec{a} \in A, \forall \vec{b} \in B, \vec{a}$ *op* $\vec{b}$



(a) Actualización del conjunto FA (caso 1)



(b) Actualización del conjunto FA (caso 2)

Figura 7.2: Actualización del conjunto FA en *Pareto-MO-RBFS*.

Demostración. El lema se demuestra por inducción sobre la profundidad del árbol explorado bajo el nodo raíz.

Caso base. A profundidad cero, $n = r$, donde r es el nodo raíz del árbol de búsqueda. En la llamada inicial $FA_n = nodomset(H(r)) = nodomset(F(r))$ y $C_{sup} = \{\vec{\infty}\}$. El único nodo abierto es $n = r$, con lo cual $OD(r) = nodomset(F(r)) = FA_n$, verificándose de modo trivial la primera propiedad. Dado que $ON(n) = \emptyset$, se puede considerar correcta también la segunda propiedad.

Hipótesis de inducción. Se asume que se verifican las propiedades para toda llamada *Pareto-MO-RBFS* (n, FA_n, C_{sup}, SOL), con el nodo n a profundidad k.

Paso de inducción. Sea una llamada *Pareto-MO-RBFS* $(n', FA_{n'}, C'_{sup}, SOL)$, donde el nodo n' se encuentra a profundidad $k+1$. Sea n el padre de n'.

Si el nodo n no se ha expandido previamente, entonces $OD(n) = nodomset(F(n))$. Por hipótesis de inducción, $FA_n \preceq_I OD(n)$, y por lo tanto $FA_n \preceq_I nodomset(F(n))$. En ese caso el valor inicial de $FA_{n'}$ es $nodomset(F(n'))$. Dado que n no ha sido expandido, tampoco lo ha sido ninguno de sus sucesores, con lo cual $OD(n') = nodomset(F(n'))$, y por lo tanto $FA_{n'} = OD(n')$, verificando $FA_{n'} \preceq_I OD(n')$.

Si el nodo n ya ha sido expandido previamente, entonces existen dos posibilidades:

- Si el sucesor n' no ha sido expandido, entonces $OD(n') = nodomset(F(n'))$.
- Si el sucesor n' ha sido expandido previamente, entonces n' es un nodo interior del árbol más grande explorado bajo n, y por definición de nodo interior se cumple que $F(n') \preceq OD(n')$. En efecto, ambos nodos n y n' fueron expandidos en la última llamada. En dicha llamada, por el lema 7.6.2 sabemos que el padre n devolvió $PF(n, C_{sup}, SOL)$, es decir, $OD(n)$, mientras que su hijo n' devolvió $PF(n', C'_{sup}, SOL)$, es decir, $OD(n')$. Pero por dicho lema sabemos también que esta frontera $OD(n')$ avanza hacia vectores dominados o indiferentes, con lo cual $F(n') \preceq_I OD(n')$.

Es decir, tanto si n' ha sido expandido como si no, se cumple que $nodomset(F(n')) \preceq OD(n')$. Además, como n ya ha sido expandido, por hipótesis de inducción tenemos que $FA_n \preceq_I OD(n)$. Pero por el lema 7.6.2 sabemos que el árbol explorado por n' es un subárbol del explorado por su padre n, y $OD(n')$ es un subconjunto de $OD(n)$, con lo cual $FA_n \preceq_I OD(n')$.

Los valores iniciales para el conjunto $FA_{n'}$ pueden ser $nodomset(F(n'))$ o bien $\bigcup_{\vec{f} \in F(n')} MaxVector(\vec{f}, FA_n)$. En ambos casos se cumple la primera propiedad, puesto que:

- Sea $FA_{n'} = nodomset(F(n'))$. Dado que $nodomset(F(n')) \preceq OD(n')$, entonces tenemos que $FA_{n'} \preceq OD(n')$.
- Sea $FA_{n'} = \bigcup_{\vec{f} \in F(n')} MaxVector(\vec{f}, FA_n)$. Tenemos que $FA_n \preceq_I OD(n)$ (por hipótesis de inducción) y $FA_n \preceq_I OD(n')$ (ya que $OD(n') \subseteq OD(n)$). Además sabemos que $nodomset(F(n')) \preceq OD(n')$. Sean $\vec{v} \in OD(n')$, $\vec{f_1} \in FA_n$ |

$(\vec{f_1} \preceq \vec{v}) \vee (\vec{f_1} \sim \vec{v})$, y $\vec{f_2} \in nodomset(F(n')) \mid \vec{f_2} \preceq \vec{v}$. Entonces el vector $\vec{m} = \text{máx}(\vec{f_1}, \vec{f_2})$ verifica que $(\vec{m} \preceq \vec{v}) \vee (\vec{m} \sim \vec{v})$, y por lo tanto $FA_{n'} \preceq_I OD(n')$.

En resumen, para los valores iniciales de $FA_{n'}$ se cumple $FA_{n'} \preceq_I OD(n')$.

Veamos ahora si la primera propiedad se cumple para los valores $FA_{n'}$ obtenidos mediante las llamadas recursivas. En una llamada recursiva sobre un hijo n' hecha durante la llamada *Pareto-MO-RBFS (n, FA_n, C_{sup_n}, SOL)*, por el lema 7.6.2 sabemos que el valor $FA[1]$ (es decir, $FA_{n'}$) se actualiza a $PF(n', C'_{sup}, SOL)$, siendo $T(n', C'_{sup}, SOL)$ el subárbol explorado durante dicha llamada recursiva sobre n'.

Sea $T(n', C_s, SOL)$ el mayor subárbol explorado bajo n' durante las diferentes llamadas recursivas. Los nodos frontera de este árbol son los descendientes abiertos de n', y $PF(n', C_s, SOL)$ es el conjunto de vectores no dominados de dicho árbol, es decir, $OD(n')$. Sea $T(n', C'_{sup}, SOL)$ el último árbol explorado bajo n', en cualquier caso se cumple entre ambos árboles que $T(n', C'_{sup}, SOL) \subseteq T(n', C_s, SOL)$:

- Si $T(n', C'_{sup}, SOL) = T(n', C_s, SOL)$ (última expansión de n'), entonces se verifica que $FA_{n'} = PF(n', C'_{sup}, SOL) = PF(n', C_s, SOL) = OD(n')$.
- Si $T(n', C'_{sup}, SOL) \subset T(n', C_s, SOL)$, entonces, debido a la progresión de la frontera analizada en el lema 7.6.2, $C'_{sup} \prec_I C_s$ y por lo tanto, como $FA_{n'} = PF(n', C'_{sup}, SOL)$ y $OD(n') = PF(n', C_s, SOL)$, se cumple que $FA_{n'} \preceq_I OD(n')$.

Es decir, para cualquier llamada recursiva sobre n', $FA_{n'} \preceq_I OD(n')$, con lo cual queda demostrado que $FA_{n'} \preceq_I OD(n')$ para cualquier valor de $FA_{n'}$ durante una llamada *Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL)* sobre su padre n. Pasamos ahora a demostrar la segunda propiedad de este lema: $C_{sup} \preceq_I ON(n)$.

Sea una llamada recursiva *Pareto-MO-RBFS (n', $FA_{n'}$, C'_{sup}, SOL)*, $FA[1]$ (que es $FA_{n'}$) cumple que para un hermano cualquiera m' de n', $FA_{n'} \preceq_I FA_{m'}$, debido a las operaciones de selección y filtrado que realiza el algoritmo previamente a la expansión de n'.

Por construcción de la cota superior en las llamadas recursivas, sabemos que $C'_{sup} = nodomset(C_{sup} \cup (\bigcup_{m'} FA_{m'}))$ y, por lo tanto $C'_{sup} \preceq_I C_{sup}$. Además, por hipótesis de inducción tenemos que $C_{sup} \preceq_I ON(n)$, con lo cual se cumple que $C'_{sup} \preceq_I ON(n)$.

También por construcción tenemos que $C'_{sup} = nodomset(C_{sup} \cup (\bigcup_{m'} FA_{m'}))$ y, por lo tanto $C'_{sup} \preceq_I FA_{m'}$. Además, al haber demostrado ya la primera propiedad del lema, tenemos que $FA_{m'} \preceq_I OD(m')$, y por lo tanto se cumple que $C'_{sup} \preceq_I OD(m')$ para todo hermano m' de n'.

El conjunto de nodos abiertos no descendientes de n', $ON(n')$ está formado por la unión de los nodos abiertos que no son descendientes de su padre n (siendo $ON(n)$ el conjunto de sus vectores de coste no dominados) y los nodos abiertos descendientes de los hermanos m' de n' (siendo $\bigcup_{m'} OD(m')$ el conjunto de sus vectores de coste no dominados). Es decir, $ON(n') = ON(n) \cup (\bigcup_{m'} OD(m'))$. En consecuencia, dado que $C'_{sup} \preceq_I ON(n)$ y $C'_{sup} \preceq_I OD(m')$, se verifica también la segunda propiedad del lema.

□

Lema 7.6.4. *Cuando se expande un nodo n mediante una llamada Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL), se verifica que* $\forall \vec{f} \in FA_n \quad \nexists \vec{v} \in (OD(n) \cup ON(n)) \mid \vec{v} \prec \vec{f}$. *Es decir, Pareto-MO-RBFS expande los nodos en orden best − first multiobjetivo.*

Demostración. Supongamos la expansión de un nodo n' mediante la llamada *Pareto-MO-RBFS (n', $FA_{n'}$, C'_{sup}, SOL).* Antes de expandirlo, se comprueba que $FA_{n'} \preceq_I C'_{sup}$. Además, por el lema 7.6.3 sabemos que $C'_{sup} \preceq_I ON(n')$, y por lo tanto $FA_{n'} \preceq_I ON(n')$.

Por otra parte, el valor inicial del conjunto $FA_{n'}$ es, o bien $nodomset(F(n'))$ o bien $\bigcup_{\vec{f} \in F(n')} MaxVector(\vec{f}, FA_n)$, donde n es el padre de n'. En consecuencia, el valor inicial de $FA_{n'}$ verifica que $F(n') \preceq FA_{n'}$. Además, por construcción de C'_{sup}, tenemos garantizado que $FA_{n'} \sim C'_{sup}$. Dado que la llamada *Pareto-MO-RBFS (n', $FA_{n'}$, C'_{sup}, SOL)* devuelve un conjunto de vectores $FA_{n'_{bis}}$ dominados estrictamente por C'_{sup} ($C'_{sup} \prec FA_{n'_{bis}}$), tenemos que $FA_{n'} \preceq_I FA_{n'_{bis}}$ y, por lo tanto $F(n') \preceq_I FA_{n'}$ para toda actualización de $FA_{n'}$ en una llamada recursiva.

Además, por el lema 7.6.3, sabemos que $FA_n \preceq_I OD(n)$ y, por lo tanto, $FA_{n'} \preceq_I OD(n)$. Como también se verifica $FA_{n'} \preceq_I ON(n')$, entonces tenemos que $F(n') \preceq_I ON(n')$.

Todo nodo abierto del árbol es o *descendiente* o *no descendiente* del nodo n. Como $F(n') \preceq_I OD(n)$ y $F(n') \preceq_I ON(n)$, se verifica que $\forall \vec{f} \in F(n'), \nexists \vec{v} \in (OD(n') \cup ON(n')) \mid \vec{v} \prec \vec{f}$, y por lo tanto los nodos se expanden en orden *best-first* multiobjetivo.

□

Lema 7.6.5. *Pareto-MO-RBFS nunca expandirá una solución subóptima.*

Demostración. La demostración es trivial a partir del lema 7.6.4. Sea $(\vec{c}, \gamma)$ una solución subóptima, y sea $(\vec{c^*}, \gamma')$ una solución óptima tal que $\vec{c^*} \prec \vec{c}$. Para todo nodo n en el camino $P = (s, \ldots, n, \ldots, \gamma')$ correspondiente a la solución óptima se verifica que $\exists \vec{f} \in F(n) \mid \vec{f} \preceq \vec{f}(\gamma) = \vec{c^*}$.

Si la solución óptima $(\vec{c^*}, \gamma')$ ya ha sido localizada y añadida al conjunto SOL, el filtrado previo a la expansión no permitirá seleccionar como candidato a expandir el nodo final γ.

Si por el contrario la solución óptima $(\vec{c^*}, \gamma')$ no ha sido localizada, de modo trivial existe en el árbol un nodo abierto que pertenece al camino P, y por lo tanto $\exists \vec{v} \in (ON(\gamma) \cup OD(\gamma)) \mid \vec{v} \preceq \vec{f}(\gamma') \prec \vec{f}(\gamma)$. Por lo tanto no es posible expandir el nodo γ. □

Lema 7.6.6. *Dada una llamada Pareto-MO-RBFS (n, FA_n, C_{sup}, SOL), donde entre los nodos interiores del árbol $T(n, C_{sup}, SOL)$ existe un conjunto de soluciones no dominadas SOL2, entonces la llamada devuelve $PF(n, C_{sup}, SOL \cup SOL2)$ y el conjunto de soluciones $SOL \cup SOL2$.*

Demostración. Por el lema 7.6.5 tenemos garantizado que el algoritmo nunca selecciona una solución dominada. Además, por el lema 7.6.2 sabemos que el algoritmo explorará el árbol $T(n, C_{sup}, SOL)$, encontrando la primera de las soluciones de $SOL2$, de coste $\vec{c^*}$. A partir de ese momento, la solución se añade al conjunto SOL, por lo que el resto de la exploración será análoga a la realizada hasta el momento, salvo que se descartarán adicionalmente los caminos dominados por $\vec{c^*}$. Esto no impedirá que se encuentren el resto de soluciones de $SOL2$, al ser todas no dominadas entre sí, devolviendo la llamada finalmente el valor $PF(n, C_{sup}, SOL \cup SOL2)$.

□

Teorema 7.6.1. *Si existe al menos una solución, Pareto-MO-RBFS encuentra todas las soluciones no dominadas.*

Demostración. Supongamos que no se encuentra una solución no dominada de coste $\vec{c^*}$. Sabemos que dicha solución no se descarta, y que por los lemas 7.6.1 y 7.6.2 el valor de los diferentes FA_n, para todo nodo n sucesor de la raíz, se va incrementando, por lo que en un número finito de pasos se alcanzará $\vec{c^*}$. □

Teorema 7.6.2. *Si existe al menos una solución, entonces el algoritmo Pareto-MO-RBFS finaliza, devolviendo todas las soluciones no dominadas.*

Demostración. Sabemos por el teorema 7.6.1 que el algoritmo encuentra todas las soluciones no dominadas, es decir, $SOL = C^*$. Dado que todas las componentes de coste tienen un valor mínimo, ϵ_i, el árbol $T(s, C_{sup}, SOL)$ va progresando, al igual que el conjunto N_{suc}, hasta que todos sus vectores están dominados por el coste de alguna solución en un número finito de pasos.

□

7.7. Propiedades de IP-MO-$RBFS$

Antes de demostrar la completitud y admisibilidad del algoritmo *Pareto-MO-RBFS* definiremos los siguientes términos, análogamente a los definidos para *Pareto-MO-RBFS* en la sección 7.6 o para *RBFS* en (Korf, 1993):

- Los conjuntos SOL y C_{sol} son los mismos descritos para *Pareto-MO-RBFS* e *IP-MO-RBFS* en la sección 5.2.1.
- $IPT(n, \vec{c_{sup}}, SOL)$ es el subárbol bajo el nodo n formado por nodos interiores y nodos frontera. Sea $F(n)$ el conjunto $\{\vec{f} = \vec{g}(n) + \vec{h} \mid \vec{h} \in H(n)\}$. Los nodos interiores n' son aquellos que tienen al menos un vector de coste $\vec{f} \in F(n')$ que no está dominado por el coste de ninguna solución y no es estrictamente peor que $\vec{c_{sup}}$, es decir, $\exists \vec{f} \in F(n') \mid \vec{c_{sup}} \not\ll \vec{f}'$. Los nodos frontera m son aquellos que verifican que cada uno de sus vectores de coste es estrictamente peor que $\vec{c_{sup}}$ o está dominado por alguna solución de SOL. Además, ningún sucesor de un nodo

frontera forma parte del árbol, y todos los sucesores de un nodo interior están en el árbol.

- $IPD(n, c_{sup}^{\rightarrow}, SOL)$ es la profundidad máxima del árbol $IPT(n, c_{sup}^{\rightarrow}, SOL)$.
- $IPF(n, c_{sup}^{\rightarrow}, SOL)$ es el punto ideal del conjunto de vectores de los conjuntos $F(n)$ no dominados de los nodos frontera del árbol $IPT(n, c_{sup}^{\rightarrow}, SOL)$, es decir, el conjunto de vectores de coste no dominados entre sí de los nodos frontera del árbol $T(n, c_{sup}^{\rightarrow}, SOL)$ tal que no son dominados por ningún vector de C_{sol}.

Lema 7.7.1. *Todas las llamadas IP-MO-RBFS (n, FA_n, $c_{sup}^{\rightarrow}$, SOL) cumplen que* $\exists \vec{c} \in FA_n \mid \nexists \vec{v} \in (\{c_{sup}^{\rightarrow}\} \cup C_{sol}), \vec{v} \ll \vec{c}$.

Demostración. Para la llamada inicial se cumple de modo trivial, pues $c_{sup}^{\rightarrow} = \vec{\infty}$ y $FA_n = nodomset(H(s))$ ($\vec{\infty} \not\ll \vec{h} \quad \forall \vec{h} \in H(s)$). Consideremos un nodo n' sucesor de n, y una llamada *IP-MO-RBFS (n', $FA_{n'}$, $c_{sup}^{\prime\rightarrow}$, SOL')* recursiva desde *IP-MO-RBFS (n, FA_n, $c_{sup}^{\rightarrow}$, SOL)*. La propiedad se cumple por construcción de los valores del conjunto FA (conjunto $N1_{sc}$), pues antes de realizar la llamada:

- se eliminan los vectores dominados por alguna solución.
- se comprueba que haya al menos un vector que no sea estrictamente peor que la cota superior del padre, $c_{sup}^{\rightarrow}$.

□

Lema 7.7.2. *Si $c_{sup}^{\rightarrow}$ es finito e $IPT(n, c_{sup}^{\rightarrow}, SOL)$ no contiene ningún nodo solución, entonces la llamada IP-MO-RBFS (n, FA_n, $c_{sup}^{\rightarrow}$, SOL) explora el árbol $T(n, c_{sup}^{\rightarrow}, SOL)$ y devuelve:*

- $\vec{\infty}$ *si todas las ramas exploradas terminan sin sucesores, son estrictamente peores que la cota superior $c_{sup}^{\rightarrow}$ o están dominadas por el coste de una solución, ó*
- $IPF(n, c_{sup}^{\rightarrow}, SOL)$ *en otro caso.*

Demostración. Por el supuesto 3, si el conjunto $(\{c_{sup}^{\rightarrow}\} \cup C_{sol})$ tiene valores finitos en todas las componentes de sus vectores, entonces el árbol $T(n, c_{sup}^{\rightarrow}, SOL)$ es finito, ya que tenemos asegurado que en un número finito de pasos, cualquier camino explorado tendrá un coste estrictamente peor que cualquier vector de $\{c_{sup}^{\rightarrow}\} \cup C_{sol}$.

Dado que $IPT(n, c_{sup}^{\rightarrow}, SOL)$ es finito, también lo será la profundidad máxima de dicho árbol, es decir, $IPD(n, c_{sup}^{\rightarrow}, SOL)$. Demostraremos el lema por inducción sobre la profundidad máxima del árbol considerado, $IPT(n, c_{sup}^{\rightarrow}, SOL)$, el cual es finito.

Caso base. En el caso base, $IPD(n, c_{sup}^{\rightarrow}, SOL) = 0$, n es el único nodo del árbol, no llegándose a explorar ninguno de sus hijos. Esto implica que la llamada *IP-MO-RBFS (n, FA_n, $c_{sup}^{\rightarrow}$, SOL)* finaliza en alguna de las cuatro primeras comprobaciones realizadas en la función *IP-MO-RBFS*.

En la primera comprobación, si $\forall \vec{f} \in F(n), \exists \vec{c^*} \in C_{sol} \mid \vec{c^*} \prec \vec{f}$, entonces el algoritmo devuelve $\vec{\infty}$.

Si por el contrario existen vectores de coste $\vec{f} \in F(n)$ no dominados por el coste de ninguna solución, y dichos vectores (almacenados en la variable N_{sol}) son todos estrictamente peores que la cota superior, es decir, $\forall \vec{f} \in N_{sol}, \vec{c_{sup}} \ll \vec{f}$, entonces la llamada devuelve precisamente $IPF(n, \vec{c_{sup}}, SOL)$.

La tercera comprobación no es aplicable, al suponer que en el árbol explorado no existe ninguna solución. En la cuarta comprobación, si el nodo considerado no tiene sucesores, entonces la llamada devuelve $\vec{\infty}$. Por lo tanto el lema se verifica para un árbol de profundidad cero.

Hipótesis de inducción. Por hipótesis de inducción, suponemos que el lema se cumple para toda llamada *IP-MO-RBFS (n, FA_n, $\vec{c_{sup}}$, SOL)* cumpliendo la profundidad del árbol explorado que $IPD(n, \vec{c_{sup}}, SOL) \leq k$.

Paso de inducción. Supongamos una llamada *IP-MO-RBFS (n, FA_n, $\vec{c_{sup}}$, SOL)* que cumple las condiciones e $IPD(n, \vec{c_{sup}}, SOL) = k + 1$. Todas las llamadas recursivas *IP-MO-RBFS (n_i, FA_{n_i}, $\vec{c_{sup_i}}$, SOL)*, con $n_i \in Sucessors(n)$, van a cumplir la propiedad, ya que $\vec{c_{sup_i}} = iPoint(\vec{c_{sup}} \cup (\bigcup_{j \neq i} FA_{n_j}))$[2]. Por lo tanto, si un vector $\vec{v}$ es estrictamente peor que $\vec{c_{sup}}$, entonces también será estrictamente peor que $\vec{c_{sup_i}}$. Es decir, $IPT(n_i, \vec{c_{sup_i}}, SOL)$ es un subárbol de $IPT(n, \vec{c_{sup}}, SOL)$ y por lo tanto $IPD(n_i, \vec{c_{sup_i}}, SOL) < IPD(n, \vec{c_{sup}}, SOL)$.

Dado que n_i es un hijo directo de n y la profundidad $IPD(n, \vec{c_{sup}}, SOL) = k + 1$, entonces $IPD(n_i, \vec{c_{sup_i}}, SOL) \leq k$.

Por el lema 7.7.1 sabemos que en $FA[1] = N1_{sc}$ existe al menos un vector que no es estrictamente peor que algún vector de $FH = \bigcup_{i \neq 1} FA[i]$ (la frontera de Pareto definida por los hermanos del nodo considerado), ni estrictamente peor que $\vec{c_{sup}}$ (calculado a partir de la frontera de Pareto definida por los nodos abiertos no descendientes del nodo considerado), y además no dominado por ninguna solución ya localizada (conjunto SOL). Además sabemos que todos los vectores devueltos por la llamada *IP-MO-RBFS (n_i, FA_{n_i}, $\vec{c_{sup_i}}$, SOL)* son estrictamente peores que $\vec{c_{sup_i}}$.

Pasaremos a demostrar que en cada llamada el conjunto N_{suc}, definido como $nodomset(\bigcup_i FA[i])$, progresa o avanza de algún modo. Sea A_1 el (hiper)área estrictamente peor que $\vec{a_1} = iPoint(\{\vec{c_{sup}}\} \cup N_{suc})$, y que define la *frontera* de la búsqueda antes de la llamada. Sea A_2 el (hiper)área estrictamente peor que $\vec{a_2} = iPoint(\{\vec{c_{sup}}\} \cup N_{suc})$, con $N_{suc} = iPoint(\{\vec{c_{sup}}\} \cup FA'[1] \cup FH)$ y $FA'[1]$ el valor actualizado de $FA[1]$ devuelto por la llamada recursiva tras la expansión.

- Por construcción del algoritmo, $\exists \vec{v} \in FA[1] \mid \vec{a_1} \not\ll \vec{v}$, y además dicho vector $\vec{v}$ tiene alguna componente en común con $\vec{a_1}$ (ver figura 7.3a). En este caso, el vector empeora estrictamente en al menos dicha componente, y por lo tanto $\vec{a_1} \prec \vec{a_2}$. De este modo la cota de búsqueda avanza hasta localizar el valor $\vec{c_{sup}}$.

[2] La función *iPoint* calcula el *punto ideal* de un conjunto de vectores

- Si existe algún otro hermano del nodo expandido con un vector de coste que tenga idéntico valor para la componente anteriormente mencionada, caso reflejado en la figura 7.3b, tras la llamada $\vec{a_1} = \vec{a_2}$. En este caso la cota de búsqueda no avanza de modo inmediato. Al estar acotado el factor de ramificación, y dado que la componente en cuestión irá empeorando en todos los hermanos, en un número finito de pasos se cumplirá que $\vec{a_1} \prec \vec{a_2}$.

Por tanto la secuencia de valores $\vec{ip}_{suc}$ (área de búsqueda delimitada por los sucesores) es monótona no decreciente, y en un número finito de pasos será estrictamente peor que $c_{\vec{sup}}$, terminando la llamada actual. Es decir, el valor $\vec{ip}_{suc}$ avanza, mientras que $c_{\vec{sup}}$ permanece invariante. Dado que existe un coste mínimo por componente (supuesto 3), habrá un número finito de llamadas antes de que todo vector del conjunto FA sea estrictamente peor que $c_{\vec{sup}}$, momento en el cual la llamada finaliza, devolviendo $IPF(n_i, c_{\vec{sup}_i}, SOL)$.

En este punto está demostrado que para todo nodo n_i sucesor de n, la llamada *IP-MO-RBFS* (n_i, FA_{n_i}, $c_{\vec{sup}_i}$, *SOL)* devuelve $IPF(n_i, c_{\vec{sup}_i}, SOL)$ en un número finito de pasos. Ahora es necesario demostrar que la consideración de todos los puntos ideales devueltos por todos los sucesores n_i de n hace que se cumpla la propiedad para la llamada *IP-MO-RBFS* (n, FA_n, $c_{\vec{sup}}$, *SOL)*.

Pero esto es trivial, puesto que al no haber solución en el árbol $IPT(n, c_{\vec{sup}}, SOL)$, todos los sucesores de n se expanden en la última llamada con $c_{\vec{sup}_i} = c_{\vec{sup}}$ y exploran $IPT(n_i, c_{\vec{sup}}, SOL)$, devolviendo $IPF(n_i, c_{\vec{sup}}, SOL)$. Pero tenemos que el punto ideal $\bigcup_{n_i} IPF(n_i, c_{\vec{sup}}, SOL), n_i \in Sucessors(n)$ es, por definición, el vector $IPF(n, c_{\vec{sup}}, SOL)$.

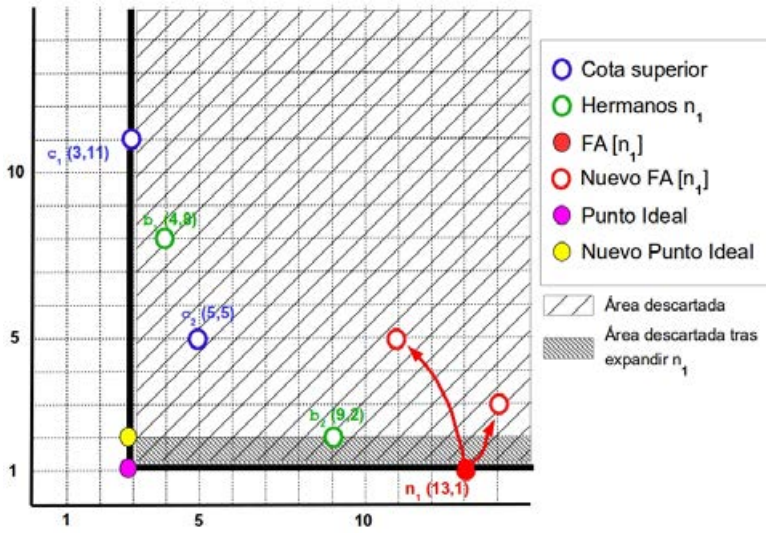
□

Lema 7.7.3. *Dada una llamada IP-MO-RBFS* (n, FA_n, $c_{\vec{sup}}$, *SOL), donde entre los nodos interiores del árbol* $IPT(n, c_{\vec{sup}}, SOL)$ *existe un conjunto de soluciones* $SOL2$*, entonces la llamada devuelve* $IPF(n, FA_n, c_{\vec{sup}}, SOL_n)$*, donde* SOL_n *es el conjunto de soluciones no dominadas extraído de* $SOL \cup SOL2$*.*

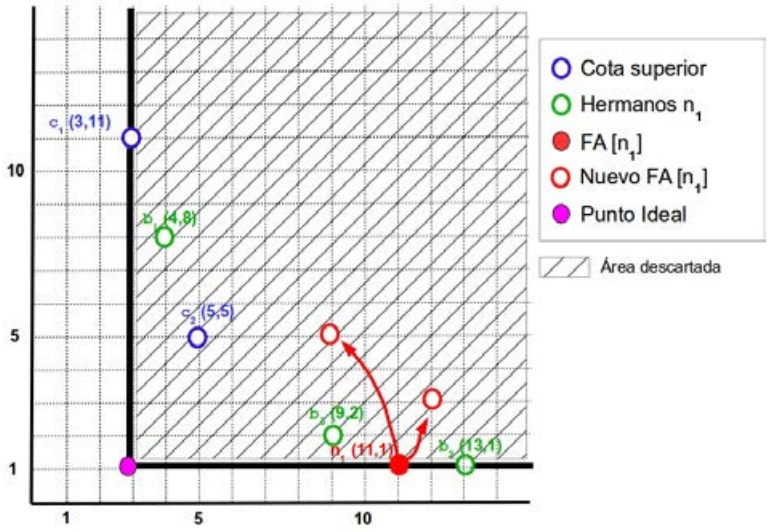
Demostración. Por el lema 7.7.2 sabemos que el algoritmo explorará el árbol definido por $IPT(n, c_{\vec{sup}}, SOL)$, encontrando una primera solución de $SOL2$, de coste $\vec{c}$. A partir de ese momento, la solución se añade al conjunto SOL, por lo que el resto de la exploración será análoga a la realizada hasta el momento, salvo que se descartarán adicionalmente los caminos dominados por $\vec{c}$. Si alguna de estas soluciones domina a alguna de las ya localizadas, éstas últimas se eliminan del conjunto SOL. Análogamente, si durante la exploración se localiza una solución dominada por otra localizada previamente, la primera no se añade al conjunto SOL. Esto no impedirá que se encuentren el resto de soluciones de $SOL2$, devolviendo la llamada finalmente el valor $IPF(n, c_{\vec{sup}}, nodomset(SOL \cup SOL2))$.

□

Teorema 7.7.1. *Si existe al menos una solución, IP-MO-RBFS encuentra todas las soluciones no dominadas, con* $SOL = C^*$*.*



(a) Actualización del conjunto FA (caso 1)



(b) Actualización del conjunto FA (caso 2)

Figura 7.3: Actualización del conjunto FA en *IP-MO-RBFS*.

Demostración. Supongamos que no se encuentra una solución no dominada de coste $\vec{c^*}$. Por los lemas 7.7.1 y 7.7.2 el valor de los diferentes FA_n, para todo nodo n sucesor de la raíz, se va incrementando, por lo que en un número finito de pasos se alcanzará $\vec{c^*}$ (dado que la cota inicial sobre el nodo raíz es $\vec{\infty}$), y el nodo asociado a dicha solución óptima se expandirá. □

Teorema 7.7.2. *Si existe al menos una solución, entonces el algoritmo IP-MO-RBFS finaliza, devolviendo todas las soluciones no dominadas.*

Demostración. Sabemos por el teorema 7.7.1 que el algoritmo encuentra todas las soluciones no dominadas, es decir, $SOL = C^*$. Dado que todas las componentes de coste tienen un valor mínimo, ϵ_i, el árbol $IPT(s, \vec{c_{sup}}, SOL)$ va progresando, al igual que el vector $\vec{ip}_{suc}$, hasta que esté dominado por el coste de alguna solución en un número finito de pasos.

□

Capítulo 8

Evaluación empírica

8.1. Introducción

En los Capítulos 4 y 5 introdujimos nuevos algoritmos multiobjetivo basados en profundización iterativa ($PIDMOA^*$, $LEXIDMOA^*$ e $IPID^*$) y búsqueda recursiva *best-first* ($Pareto\text{-}MO\text{-}RBFS$ e $IP\text{-}MO\text{-}RBFS$). Todos ellos presentan una complejidad espacial lineal durante la exploración del árbol, ya que únicamente mantienen en memoria la rama que está siendo explorada en el instante actual, aunque como ya hemos comentado previamente, esta complejidad puede ser exponencial en lo relativo al almacenamiento del conjunto de soluciones óptimas. Se hace necesario realizar un análisis detallado del rendimiento temporal de estos algoritmos frente a otros algoritmos de tipo *retroceso* como *DF-BnB*, $MOMA^*0$ ó $IDMOA^*$ para determinar cuál es la mejor opción a utilizar según el tipo de problema planteado. Hemos optado por realizar una evaluación escalonada, de modo que se identificarán los mejores algoritmos para cada uno de los tres grandes grupos de algoritmos con retroceso definidos: profundización iterativa, búsqueda recursiva *best-first* y algoritmos secuenciales. Una última ronda de pruebas servirá para determinar qué algoritmos multiobjetivo *depth-first* presentan mejor rendimiento global.

Aunque hay diversos problemas multiobjetivo que podrían servir de campo de pruebas para la evaluación que nos atañe, en este trabajo se ha optado por emplear un marco más teórico sobre el cual realizar el estudio de rendimiento: árboles binarios finitos e infinitos, éstos últimos generados de acuerdo con el procedimiento elaborado por (Korf y Chickering, 1996). Esto nos permite un alto grado de control de los parámetros de los problemas, concretamente en lo que respecta a profundidad del árbol, cantidad de soluciones o correlación entre los objetivos entre otros parámetros.

Parte de los resultados presentados en este Capítulo han sido publicados en la revista *Journal of Intelligent Manufacturing* (Coego et al., 2010) y en diferentes conferencias internacionales de Inteligencia Artificial (Coego et al., 2009) (Coego et al., 2012).

Este capítulo se organiza de la siguiente forma. En primer lugar, en la Sección 8.2

comentaremos brevemente la configuración software y hardware del entorno de pruebas utilizado para la evaluación de los algoritmos. A continuación, en la sección 8.3 se describirán los diferentes tipos de problemas generados para dicha evaluación. La sección 8.4 explicará por qué se ha utilizado el tiempo de procesamiento como principal parámetro para medir el rendimiento, en detrimento de otros como el número de nodos, usado más habitualmente en este tipo de análisis. En las secciones 8.5 y 8.6 se incluyen los análisis de rendimiento realizados para los diferentes algoritmos multiobjetivo basados en profundización iterativa y búsqueda recursiva *best-first* respectivamente. La sección 8.7 determinará el mejor algoritmo de la categoría de algoritmos secuenciales, centrando la comparativa en las variantes multiobjetivo de *DF-BnB* sobre espacios de estados infinitos con cota inicial. Asimismo se realiza una evaluación del impacto que tienen las cotas iniciales en el rendimiento del algoritmo *DF-BnB* multiobjetivo. La sección 8.8 realizará una última evaluación de los mejores algoritmos obtenidos de las secciones anteriores con el fin de determinar la mejor opción multiobjetivo *best-first*. Por último, en la sección 8.9 se enumerarán una serie de conclusiones globales obtenidas de los diferentes experimentos realizados.

8.2. Entorno de pruebas

La práctica totalidad de los experimentos contemplados en este Capítulo fueron realizados en una máquina con dos procesadores Six-Core AMD Opteron 2435 2600MHz, 64GB de memoria RAM y sistema operativo Windows Server 2008 R2 Enterprise 64bits. De modo auxiliar se ha empleado otra máquina HP Proliant DL160 G5 server con dos procesadores Intel Xeon QuadCore 4572 @ 3GHz, 18 GB de memoria RAM y sistema operativo Windows Server 2008 32bits. Todos los algoritmos fueron implementados en LispWorks Enterprise Edition 5.0 y 6.0, y se ejecutaron sin necesidad del entorno Lisp, ya que se generaron los correspondientes ejecutables de 32 (versión 5.0) y 64 bits (versión 6.0) para las máquinas mencionadas. La ejecución de los algoritmos se realizó en una sola hebra, es decir, utilizando un único procesador en cada momento.

La gran mayoría de los experimentos se ejecutaron dos veces:

- una para medir el tiempo de resolución de los problemas,
- otra para obtener otras medidas adicionales (tales como número de nodos expandidos, cantidad de tests de dominancia vectoriales realizados o tamaño del conjunto C^* durante las diferentes iteraciones) y así evitar distorsionar las medidas de tiempo con esta recopilación secundaria de datos.

Todas las mediciones de tiempos se realizaron sobre la primera de las máquinas anteriormente descritas. Parte del resto de pruebas para la recopilación de las medidas adicionales (nodos, tests de dominancia, tamaños de conjuntos umbral y C^*) se realizaron sobre la segunda máquina, HP Proliant. Dado que estas mediciones no están

relacionadas con el rendimiento de la máquina empleada, la diferente configuración del equipo usado no afecta al resultado de los análisis mostrados.

8.3. Generación de problemas

De modo general, los problemas de búsqueda se pueden categorizar en dos grandes grupos: los grafos generales y los árboles. En el caso de los primeros, una característica habitual es la posibilidad de alcanzar un mismo nodo a través de diferentes caminos. Es lo que sucede, por ejemplo, en los problemas de búsqueda del camino más corto en mapas de carreteras. Esto provoca la aparición de estados repetidos en ramas del árbol de búsqueda y, posiblemente, ciclos. Los algoritmos de tipo *best-first* se adaptan mejor a estos problemas, puesto que, al mantener el conjunto de nodos explorados en memoria, permiten la detección de estados repetidos durante la generación del árbol de búsqueda, a costa de un mayor consumo de memoria para su almacenamiento.

Por otro lado, los algoritmos de tipo *depth-first* no soportan en general esta detección, puesto que únicamente mantienen en memoria la rama que está siendo explorada en cada momento. Aunque se han diseñado mecanismos para paliar este problema, como el uso de tablas de transposición (Reinefeld y Marsland, 1994), estas mejoras tienen un alcance limitado.

Los problemas que mejor se adaptan a los algoritmos *depth-first* son aquellos en los que el grafo o espacio a explorar tiene forma de árbol. Algunos problemas, como las subastas combinatorias, el problema del viajante u otros ya mencionados en el Capítulo 3, se pueden formular fácilmente como búsqueda en un árbol. Para la evaluación de los algoritmos multiobjetivo incluídos en esta tesis hemos optado por realizar las pruebas de rendimiento con baterías de árboles aleatorios, tanto finitos como infinitos, debido a que nos proporcionan un mayor control sobre sus parámetros, tales como la profundidad del árbol, profundidad de las soluciones, ubicación de las mismas dentro de cada nivel, rango de costes para cada uno de los objetivos o la correlación entre los diferentes objetivos.

En la totalidad de problemas usados para el análisis se han tenido en cuenta dos objetivos, dado que la gran mayoría de problemas prácticos multiobjetivo pertenecen a esta categoría. Además, los valores para estos objetivos se han generado casi siempre siguiendo una distribución uniforme, de tal modo que la correlación entre ambos objetivos es nula. Aunque se han realizado pruebas con correlación tanto positiva como negativa entre los objetivos, los resultados no han reflejado una variación cualitativa importante con respecto a los obtenidos con correlación cero, tal y como se verá en la sección 8.8.2. Asimismo los árboles usados para las pruebas son binarios, es decir, el factor de ramificación (*branching*) es 2.

8.3.1. Árboles finitos

Los árboles finitos se han usado fundamentalmente para evaluar el rendimiento de los algoritmos *DF-BnB* multiobjetivo, los cuales, en caso de no disponer de una cota inicial, precisan de la finitud del espacio de búsqueda para poder asegurar su finalización. En este tipo de problemas, los parámetros básicos empleados para generar los bancos de prueba han sido los siguientes:

- *Profundidad máxima del árbol.* Esta profundidad es única, todas las ramas del árbol terminan en un nodo hoja ubicado en este nivel.
- *Profundidad de las soluciones.* Todos los nodos finales del problema se encuentran ubicados en un nivel de profundidad prefijado del árbol y distribuidos de manera aleatoria. Se considera que el nodo raíz está en el nivel de profundidad cero.
- *Cantidad de nodos finales.* Este parámetro se ha establecido de dos modos diferentes: en algunos casos se ha asignado una cantidad fija de nodos finales y en otros casos la cantidad de nodos finales ha sido un porcentaje sobre la cantidad de nodos que pertenecen al nivel prefijado para las soluciones. En cualquiera de los dos casos, esta cantidad incluye tanto las soluciones óptimas como las subóptimas. No se ha parametrizado por tanto de ningún modo directo el tamaño de la frontera de Pareto.
- *Rango de costes.* Cada uno de los costes del problema toma valores enteros en un rango $[n, m]$ para cada uno de los arcos que unen los nodos del árbol, generalmente $n = 1$. En cualquier caso estos costes siempre son enteros positivos. El rango es el mismo para ambos objetivos.

La generación de estos árboles se ha realizado generalmente de modo *offline*, guardando por un lado los nodos junto con los vectores de coste asociados, y por otro lado los conjuntos de nodos finales. En el caso de árboles con el 100 % de nodos finales a una profundidad fijada, estos árboles se han generado de modo similar al que comentaremos en la sección 8.3.2. El único nodo raíz del problema está etiquetado como **1** y el resto de nodos se etiqueta consecutivamente por niveles de izquierda a derecha, de modo similar al de la figura 8.1. Al ser un árbol binario, los sucesores del nodo **n** son los etiquetados como $n * 2$ y $n * 2 + 1$. Suponiendo que el nivel del nodo raíz es cero, los nodos de un nivel n se etiquetarán consecutivamente como $2^n, \ldots, 2^{n+1} - 1$.

Al tratar con problemas de tipo aleatorio, no disponemos de un heurístico específico para resolver los problemas, así que se ha implementado un heurístico genérico que se calcula como el valor absoluto de la diferencia entre la profundidad a la cual se encuentran las soluciones y la profundidad del nodo considerado ($\mid p_\gamma - p_n \mid$). Dado que cada objetivo tiene un coste entero positivo mínimo (generalmente 1), el heurístico es admisible de modo trivial. Este heurístico ha sido empleado en un número reducido de pruebas, dado que el objetivo de este trabajo no es evaluar la utilidad del heurístico,

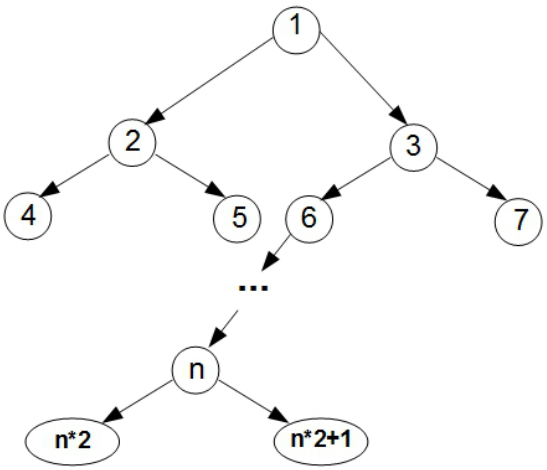


Figura 8.1: Árbol binario

sino evaluar la estrategia *depth-first* multiobjetivo más competitiva. Es de esperar que el uso de un heurístico mejorará en general el rendimiento de todos los algoritmos considerados.

8.3.2. Árboles infinitos

De modo complementario a los árboles comentados en 8.3.1 se han generado también bancos de pruebas con árboles binarios infinitos. Este tipo de árboles permiten representar problemas con espacios de estados infinitos o muy grandes y con soluciones *superficiales*, es decir, ubicadas en niveles del árbol muy próximos a la raíz.

Para la generación de este tipo de problemas se han tomado como referencia los árboles infinitos aleatorios definidos en (Korf y Chickering, 1996). La estructura del árbol binario es similar a la descrita en 8.3.1 para los árboles finitos. La gran diferencia es que este árbol infinito no se puede generar de modo estático, y por lo tanto tampoco podemos prealmacenar sus vectores de coste. Debido a ello se hace necesario un mecanismo para generar los vectores de coste aleatorios en tiempo de ejecución de un modo eficiente en tiempo de ejecución, de modo que se obtengan los mismos valores en las diferentes reexpansiones que va a tener un nodo durante la aplicación de un algoritmo de profundización iterativa. Obviamente los valores también deben mantenerse entre diferentes ejecuciones, para poder resolver de modo coherente el mismo problema con diferentes algoritmos.

Consideremos primero el caso de árboles con costes aleatorios y un único objetivo. El generador de números pseudo-aleatorio presentado en (Korf y Chickering, 1996) está basado en el mecanismo de la función *rand* de la librería estándar C, en la cual, a partir de una semilla inicial (usada para generar el primer número aleatorio) se genera la siguiente semilla de acuerdo a la fórmula $s_{i+1} = a * s_i + c$, donde $a = 1{,}103{,}515{,}245$ y $c = 12{,}345$. Desarrollando la fórmula anterior, tenemos lo siguiente:

- $s_1 = a * s_0 + c$
- $s_2 = a * s_1 + c = a^2 * s_0 + a * c + c$
- *etc.*

Esto nos lleva a la fórmula para el cálculo de la semilla s_n como $s_n = a^n * s_0 + c * \sum_{j=0}^{n-1} a^j$, pudiendo precalcularse los valores de a^n.

Por lo tanto, para generar j números aleatorios se usarán una semilla inicial y $j-1$ semillas adicionales calculadas según lo indicado anteriormente. Supongamos que tenemos un problema con un único objetivo. Precisaremos generar tantos costes como nodos menos uno tengan el árbol, ya que a cada nodo, excepto el raíz, llega un arco a etiquetar. Dado que cada nodo tiene asignado un número de orden (ver figura 8.1), asignamos a cada arco el número de orden de su nodo destino. Para calcular el coste asociado a un arco n no es necesario, por lo tanto, generar una secuencia de n números aleatorios y quedarnos con el último; es suficiente con realimentar el generador de números aleatorios con la semilla s_n y generar el número aleatorio correspondiente.

De este modo es inmediato replicar cualquier coste sin otros datos que la semilla inicial (parámetro que irá asociado a cada uno de los árboles problema) y el identificador del nodo al que llega dicho arco. En (Korf y Chickering, 1996) se detallan los cálculos necesarios para computar la semilla s_n (y por tanto cualquier coste) en un tiempo $O(\log n)$.

En el caso biobjetivo, si el árbol tiene n nodos, es necesario generar $2 \times n$ costes, es decir, 2 valores por cada vector asociado a un arco. Para ello seguimos un esquema similar al indicado en la figura 8.2. A cada arco que llega a un nodo n se le asocian dos etiquetas con los identificadores de los sucesores de n, $n \times 2$ y $n \times 2 + 1$. Estos valores se usan para calcular, a partir también de la semilla inicial asociada al problema, sendas semillas que se utilizaran para alimentar de modo consecutivo al generador de números aleatorios y obtener las dos componentes de coste del vector correspondiente. Cualquiera de estos valores aleatorios es luego fácilmente trasladable al rango de costes del problema $[1, r]$ sin más que utilizar la fórmula $nuevocoste = \lfloor mod(coste, r) \rfloor$.

De este modo es posible realizar consultas sobre costes de árboles aleatorios infinitos sin otro requisito que el almacenamiento en memoria de su semilla inicial.

8.3.3. Correlación de los objetivos

En algunos casos se han generado árboles variando la correlación entre las componentes del vector de coste generadas aleatoriamente. Aunque generalmente la correlación que se ha empleado para generar los costes es cero, siendo los valores independientes para todos los objetivos, se ha jugado también tanto con correlaciones positivas como negativas en el rango $[-1, 1]$. Para el caso de un vector de coste bidimensional (c_1, c_2), tras generar aleatoriamente el valor de coste de la primera componente c_1, usando una distribución uniforme en el rango de valores indicado, el valor de la segunda componente c_2 se ajusta usando las siguientes fórmulas:

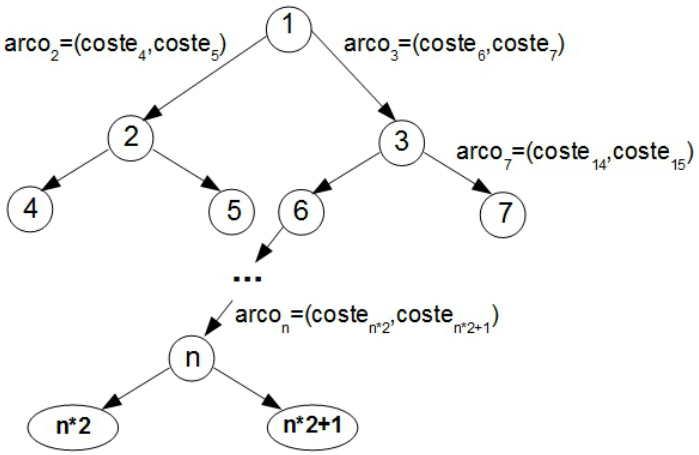


Figura 8.2: Costes aleatorios en un árbol binario aleatorio infinito

- Correlación ρ positiva ($\rho > 0$): $c_2 = (c_1 * \rho) + (c' * (1 - \rho))$
- Correlación ρ negativa ó cero ($\rho \leq 0$): $c_2 = 1 + (c_{max} - ((c_1 * (-\rho)) + (c' * (1 + \rho))))$

dónde ρ es la correlación establecida, c_1 es el coste aleatorio generado para la primera componente del vector de coste, c' es el coste aleatorio auxiliar generado para la segunda componente del vector de coste y c_{max} es el coste máximo para cualquiera de las dos componentes. Los valores de c_2 se redondean al entero más cercano.

Una asociación positiva entre los costes de los arcos se establece usando un factor de correlación $0 \leq \rho \leq 1$. La correlación positiva puede tener un impacto muy fuerte en la naturaleza del problema multiobjetivo. Cuanto mayor sea la correlación, mayor similitud tendremos en los valores de los costes. En el caso extremo, $\rho = 1$, en la práctica convertimos nuestro problema multiobjetivo en otro con un único objetivo, pues con esta correlación, dentro de un vector de coste, los valores para todos los objetivos serán idénticos. Por ello la frontera de Pareto estaría formada por un único vector de coste.

Por el contrario, una asociación negativa entre los costes de los arcos se crea con una correlación negativa $-1 \leq \rho \leq 0$. Una correlación negativa de magnitud alta incrementa enormemente la dificultad del problema. En el caso extremo, $\rho = -1$, los costes de los objetivos en un mismo vector tendrán valores opuestos (si uno es grande, el otro es pequeño, y viceversa), con lo cual la cantidad de vectores de coste no dominados se incrementan sustancialmente, incrementando asimismo el número de tests de dominancia a realizar.

8.4. Parámetros de rendimiento

Los algoritmos fruto de la comparativa de esta tesis son generalizaciones de algoritmos escalares del tipo *linear space*. Muchos trabajos que incluyen evaluaciones de

algoritmos de este tipo (p.ej. (Zhang y Korf, 1995) ó (Zhang, 1999)) se centran casi exclusivamente en medir la cantidad de nodos expandidos durante el proceso de búsqueda. Para el caso de la búsqueda con un único objetivo, éste puede ser un enfoque adecuado. El tiempo de procesamiento de un nodo en estos algoritmos se puede considerar constante, ya que este procesamiento incluye la comparación de la función de coste del nodo con el umbral vigente en ese momento (ambos valores escalares), para determinar si el nodo se expande o no. Por ello podemos asegurar que los requisitos de tiempo de estos algoritmos son directamente proporcionales a la cantidad de nodos generados.

Sin embargo, en el caso multiobjetivo, el cálculo de nodos generados no es una medida fiable del rendimiento del algoritmo, ya que la naturaleza de las comparaciones que se realizan en los algoritmos multiobjetivo es diferente del caso escalar. Veamos a continuación qué implica el procesamiento de un nodo cualquiera en los algoritmos de profundización multiobjetivo considerados en los capítulos precedentes.

En el caso de $PIDMOA^*$, cada vez que se genera un nodo n, en primer lugar el algoritmo debe comparar su vector de coste contra las soluciones que ya ha localizado el algoritmo, para determinar si tiene sentido seguir explorando el camino que pasa a través de n. El tiempo necesario para esta operación dependerá de la cantidad de soluciones que se hayan localizado hasta ese momento. Obviamente, este tiempo de procesamiento no es el mismo para todos los nodos, pues a medida que avanza la búsqueda, el conjunto que almacena los óptimos de Pareto se encuentra más poblado.

De modo análogo, para los nodos que no están dominados por ninguna solución ya localizada se debe comparar su vector de coste con los vectores que sirven de umbral para la iteración actual. Este umbral también tiene tamaño variable durante el proceso de búsqueda, con lo cual los tiempos de comparación pueden diferir mucho de un nodo a otro, incluso dentro de una misma iteración, debido a los óptimos de Pareto que se van localizando y añadiendo al conjunto de soluciones. Por ejemplo, para un conjunto umbral $\{(5,7),(6,6),(7,5)\}$, un nodo m cuyo vector de coste es $\vec{f}(m) = (8,8)$ podría ser descartado tras la primera comparación, pero no así un nodo n con $\vec{f}(n) = (4,4)$, el cual tiene que ser comparado con todos los vectores del umbral para poder continuar su expansión.

En resumen, el tiempo de procesamiento de un nodo cualquiera en $PIDMOA^*$ tiene una cierta variabilidad dependiendo del tamaño de los conjuntos *Threshold* y *SOLUTION*.

Analicemos ahora qué sucede con $IDMOA^*$. Este algoritmo usa umbrales de tipo escalar, al procesar los objetivos secuencialmente, con lo cual el tiempo para detectar si un nodo debe expandirse o no sí se puede considerar constante, a diferencia de lo que ocurría con $PIDMOA^*$. Por otro lado $IDMOA^*$ añade un tiempo de procesamiento extra para aquellos nodos que sí son nodos finales, ya que es posible que localice soluciones finales subóptimas, las cuales deben ser descartadas con los correspondientes tests de dominancia dentro del conjunto *SOLUTION*. En consecuencia, cuanto más poblado esté el conjunto de soluciones finales, más lento será el procesamiento de un

nodo final. De todos modos, quizá sea $IDMOA^*$, de entre todos los algoritmos, el que presente una mayor linealidad entre el número de nodos y tiempo de procesamiento, ya que se trata en la práctica de una secuencia de aplicaciones de IDA^*.

En el caso de $IPID^*$, al igual que en $IDMOA^*$, el tiempo de comparación de un nodo contra el umbral se puede considerar constante. Esto se debe a que dicho umbral está formado siempre por un único vector. El tiempo de procesamiento de los nodos finales presenta mayor variabilidad, pues depende también de la cantidad de soluciones ya localizadas.

Los razonamientos anteriores se pueden extender a las variantes multiobjetivo *RBFS* vistas en el Capítulo 5, resultando del mismo modo una gran variabilidad en los tiempos de procesamiento de los nodos a expandir.

Veamos qué ocurre en el caso de *MO-DF-BnB*. En este algoritmo el umbral está formado por los vectores de coste no dominados entre sí pertenecientes a soluciones ya localizadas, óptimas o subóptimas. Obviamente, a mayor número de soluciones localizadas, mayor número de tests de dominancia vectoriales que tendremos que realizar. Además, al igual que en $IDMOA^*$, también es necesario purgar soluciones dominadas cuando localizamos una que *mejora* alguna de las ya existentes.

La conclusión de estos comportamientos es que en problemas multiobjetivo la medida de nodos explorados durante la búsqueda no proporciona información fiable acerca de la calidad de los algoritmos, puesto que los tiempos de procesamiento de los nodos varía mucho entre algoritmos e incluso de un nodo a otro para el mismo algoritmo. Por ejemplo, en el caso de profundización iterativa un análisis basado en nodos estaría sesgado hacia el algoritmo que realice menor número de reexpansiones. Este comportamiento es generalizable a cualquier algoritmo multiobjetivo. Es por ello que el análisis de rendimiento empírico de este trabajo se ha basado fundamentalmente en medir tiempos de procesamiento. Adicionalmente se han realizado mediciones de otros parámetros tales como el tamaño de los conjuntos *Threshold* y $SOLUTION$, que nos darán las claves para interpretar el comportamiento de los algoritmos y sus posibles mejoras.

Por último cabe resaltar que los algoritmos multiobjetivo de tipo *depth-first* mostrados en este Capítulo, así como los vistos en el Capítulo 3, se limitan a devolver el conjunto de costes óptimos C^*, no incluyendo el conjunto de caminos asociados a estos costes óptimos. La generalización de los algoritmos para almacenar los caminos y operadores es trivial.

8.5. Comparativa de algoritmos de profundización iterativa

En esta sección presentamos la evaluación realizada sobre las diferentes variantes multiobjetivo basadas en profundización iterativa analizadas en este trabajo: $IDMOA^*$, $PIDMOA^*$, $LEXIDMOA^*$ e $IPID^*$.

En primer lugar, en la sección 8.5.1 analizaremos con detalle $PIDMOA^*$, un en-

foque multiobjetivo puro con uso de cotas multivectoriales para las diferentes profundizaciones, e $IDMOA^*$, consistente en la aplicación secuencial del algoritmo escalar IDA^*. La sección 8.5.2 incorpora a las pruebas $IPID^*$, una opción intermedia que emplea una cota formada por un solo vector. Finalmente, la sección 8.5.3 incluye una evaluación de los algoritmos con umbral de tamaño constante, tanto en su versión escalar como en su versión vectorial: $IDMOA^*$, $LEXIDMOA^*$ e $IPID^*$.

8.5.1. Comparativa de cotas escalares y multivectoriales: algoritmos $IDMOA^*$ y $PIDMOA^*$

En la sección 3.6.2 se explicaba el algoritmo $IDMOA^*$, una variante de la profundización iterativa para el caso multiobjetivo, pero que empleaba un esquema escalar, objetivo a objetivo, en el tratamiento de los vectores de coste. Los principales problemas de diseño de $IDMOA^*$, comentados detalladamente en la sección 4.1, son el procesamiento secuencial de objetivos y la inclusión temporal de soluciones subóptimas en el conjunto de soluciones a devolver. Esto conlleva por un lado importantes reexpansiones de nodos y por otro lado tests de dominancia extra para eliminar esas soluciones que no pertenecen a la frontera de Pareto.

Tal y como se comenta en la sección 4.2, $PIDMOA^*$ trata de solventar estos problemas empleando como cota para la profundización conjuntos de vectores de coste. Además localiza únicamente soluciones óptimas. Las primeras baterías de problemas empleadas para evaluar el rendimiento de $IDMOA^*$ y $PIDMOA^*$, publicadas en (Coego et al., 2009) y basadas en árboles aleatorios finitos, consideraron una versión simplificada y menos eficiente que la versión completa de $IDMOA^*$ publicada en (Harikumar y Kumar, 1996). La evaluación presentada en esta tesis emplea la versión completa de $IDMOA^*$.

Experimentos

Para la evaluación de los algoritmos $IDMOA^*$ y $PIDMOA^*$ se resolvieron una serie de árboles infinitos aleatorios (ver sección 8.3.2) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: 14.
- Rango de costes de ambos objetivos: [1,5].
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Cantidad de nodos finales: 1 único nodo final, 10 % de los nodos a la profundidad fijada y 100 % de los nodos a la profundidad fijada para las soluciones (nivel 14). Este último caso es equivalente a un árbol finito de profundidad igual a la de

las soluciones. No está prefijado cuales de estos nodos finales serán óptimos de Pareto, ya que sus costes son aleatorios.

- Correlación entre los dos objetivos: -1, -0.5, 0, 0.5, 1.
- Por cada combinación (cantidad soluciones, correlación) se resolvieron 30 problemas aleatorios diferentes (se presentan las medias).

Resultados

En las figuras 8.3 y 8.4 puede verse el ratio del tiempo de procesamiento (medido en segundos) de $IDMOA^*$ y $PIDMOA^*$ para las correlaciones $-0{,}5$, 0,5 y 0, y para todas las variantes de cantidad de nodos finales ya comentadas. Los resultados para las correlaciones -1 y 1, no incluídos en estas gráficas, son similares a los obtenidos para las correlaciones $-0{,}5$ y 0,5 respectivamente.

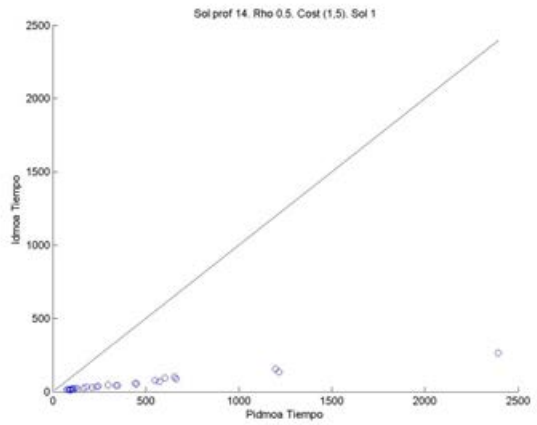
Todos los resultados mostrados bajo la diagonal $y = x$ de la gráfica representan problemas en los cuales el tiempo de resolución de $PIDMOA^*$ es mayor que el de $IDMOA^*$. Cuánto más cerca está el punto del eje X, mayor diferencia hay entre ambos algoritmos a favor de $IDMOA^*$.

Las gráficas de las figuras 8.5 y 8.6 muestran el ratio de la cantidad de tests de dominancia realizados por $IDMOA^*$ y $PIDMOA^*$ para las mismas correlaciones y cantidades de nodos finales de las figuras 8.3 y 8.4.

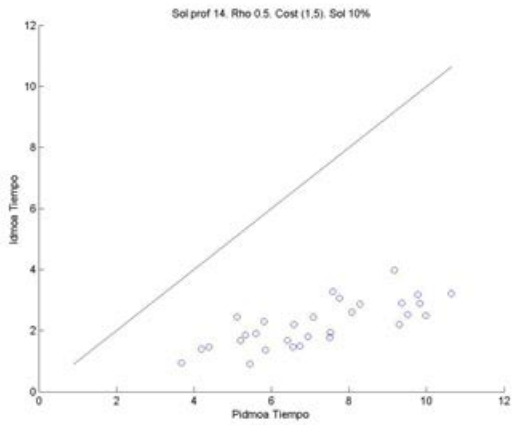
Análisis

En las figuras de las gráficas 8.3 y 8.4 se puede observar claramente que el tiempo empleado en resolver los problemas aumenta considerablemente al disminuir el número de soluciones. Este comportamiento, que será una constante en todos los algoritmos multiobjetivo *depth-first*, se debe a que las soluciones ya localizadas por cualquier algoritmo sirven como una cota superior muy eficiente, pues descartan de modo definitivo cualquier rama cuyo coste esté dominado por dichas soluciones. A mayor número de soluciones, mayor probabilidad tiene el algoritmo de realizar podas del árbol de búsqueda, mejorando la eficiencia. El resultado general es que $PIDMOA^*$ es más ineficiente que $IDMOA^*$. La respuesta a este comportamiento la obtenemos analizando la cantidad de tests de dominancia que realiza cada uno de los algoritmos.

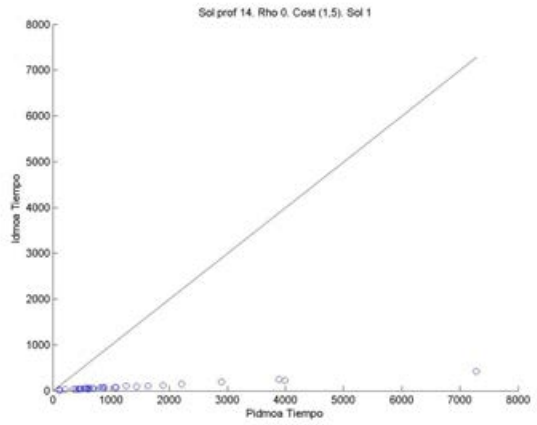
En las gráficas de las figuras 8.5 y 8.6 se muestra la proporción de tests de dominancia **vectoriales** que realizan $IDMOA^*$ y $PIDMOA^*$. Se puede apreciar que los tests que realiza $PIDMOA^*$ (comparar cada nodo expandido con el umbral y/o con el conjunto de soluciones localizadas) son muchos más que los que necesita $IDMOA^*$ (comparar las soluciones localizadas entre sí para descartar aquellas subóptimas). Aunque $IDMOA^*$ se ve obligado a realizar muchos más tests contra el umbral (al realizar, como veremos en la sección 8.5.2, más reexpansiones de nodos), éstos son de naturaleza **escalar** y su coste temporal no es significativo a la vista de lo que apreciamos en los tiempos de procesamiento indicados en las gráficas de rendimiento. De hecho se puede



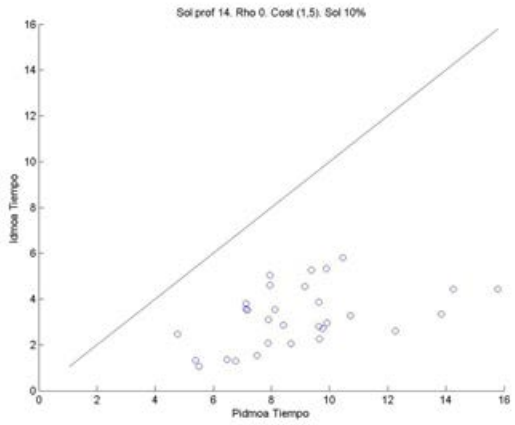
(a) $\rho = 0{,}5$ y 1 solución



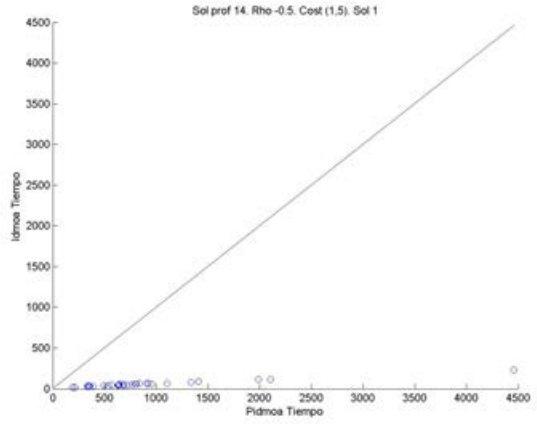
(b) $\rho = 0{,}5$ y 10 % soluciones



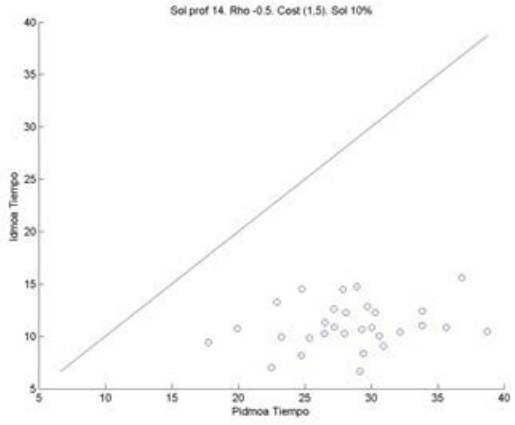
(c) $\rho = 0$ y 1 solución



(d) $\rho = 5$ y 10 % soluciones

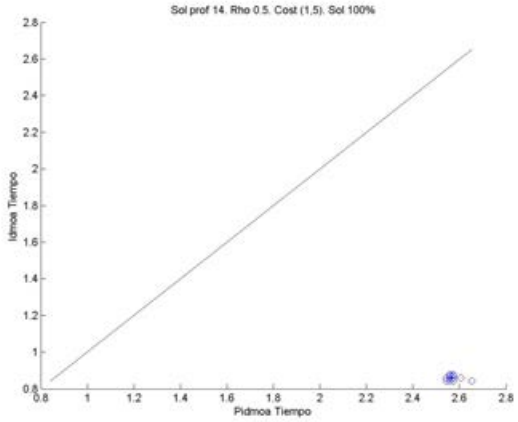


(e) $\rho = -0{,}5$ y 1 solución

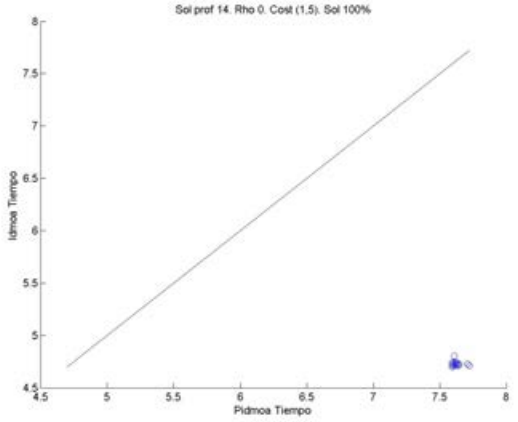


(f) $\rho = -0{,}5$ y 10 % soluciones

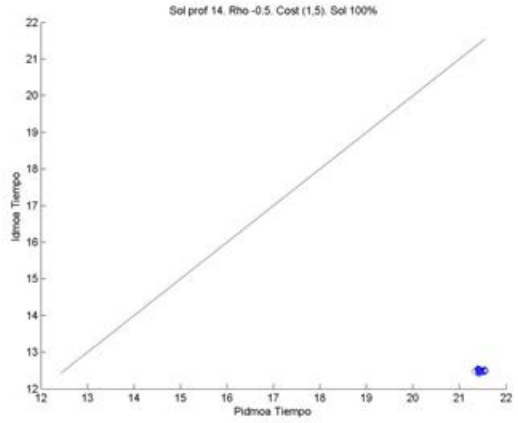
Figura 8.3: Comparativa $IDMOA^*$ vs. $PIDMOA^*$. Árboles infinitos binarios. Tiempo medido en segundos. Profundidad de las soluciones: 14. 1 solución y 10 % de soluciones.



(a) $\rho = 0{,}5$ y 100 % soluciones

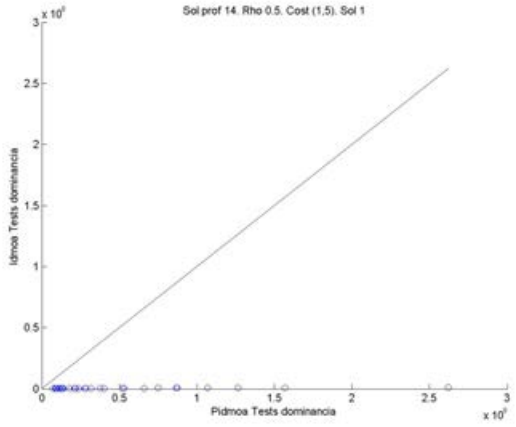


(b) $\rho = 0$ y 100 % soluciones

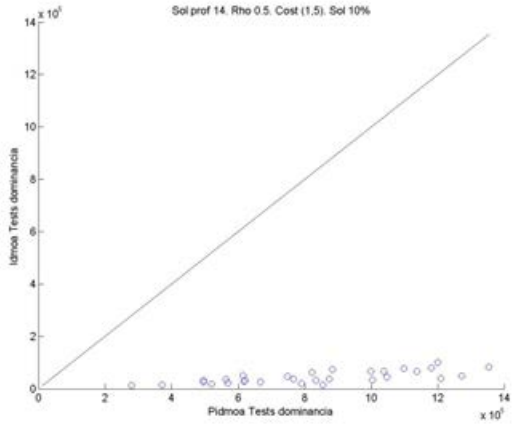


(c) $\rho = -0{,}5$ y 100 % soluciones

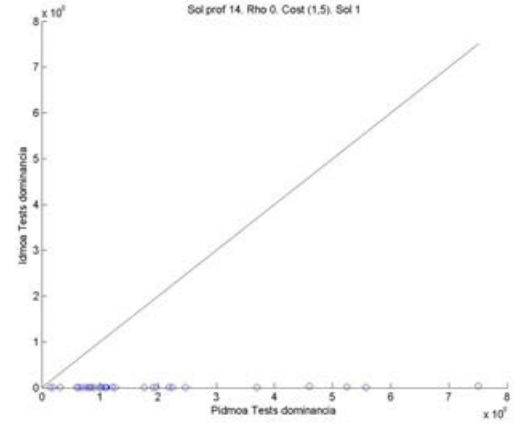
Figura 8.4: Comparativa $IDMOA^*$ vs. $PIDMOA^*$. Árboles infinitos binarios. Tiempo medido en segundos. Profundidad de las soluciones: 14. 100 % de soluciones.



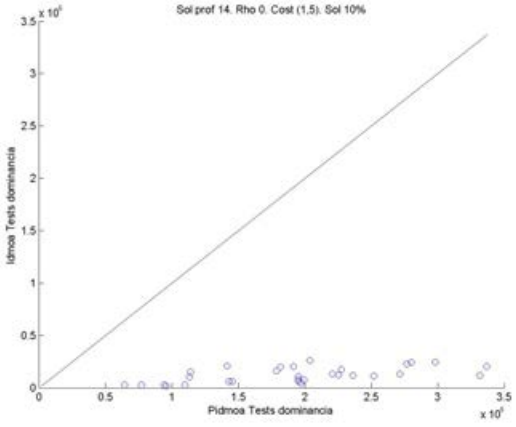
(a) $\rho = 0{,}5$ y 1 solución



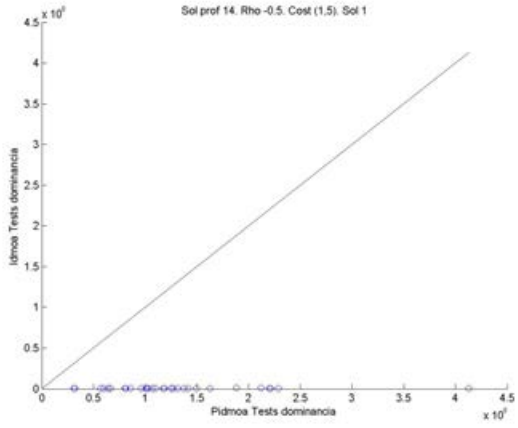
(b) $\rho = 0{,}5$ y 10 % soluciones



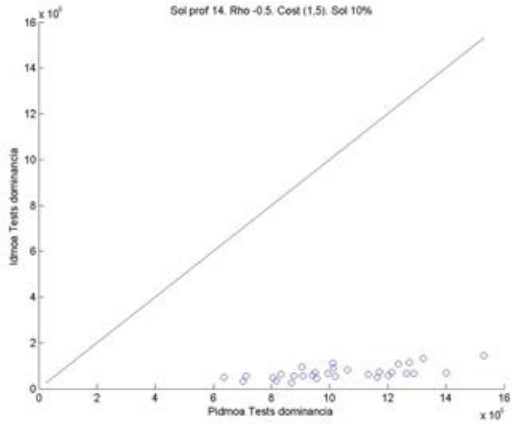
(c) $\rho = 0$ y 1 solución



(d) $\rho = 0$ y 10 % soluciones

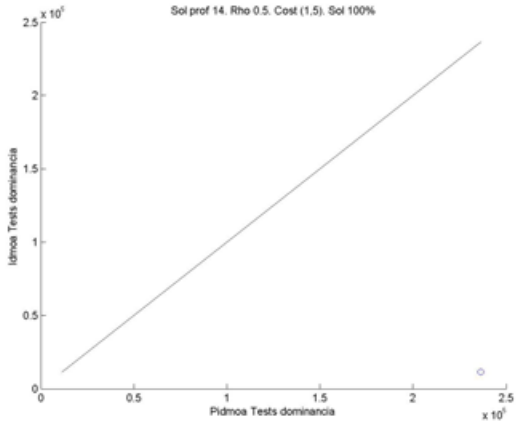


(e) $\rho = -0{,}5$ y 1 solución

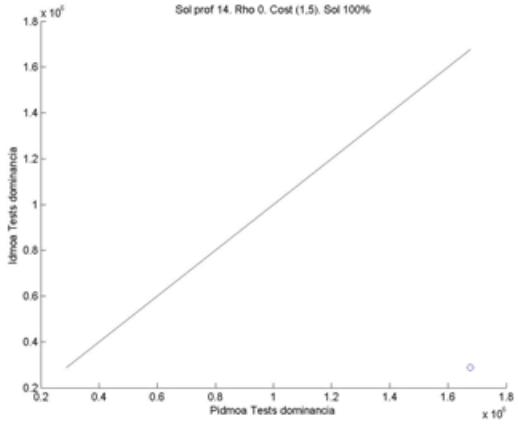


(f) $\rho = -0{,}5$ y 10 % soluciones

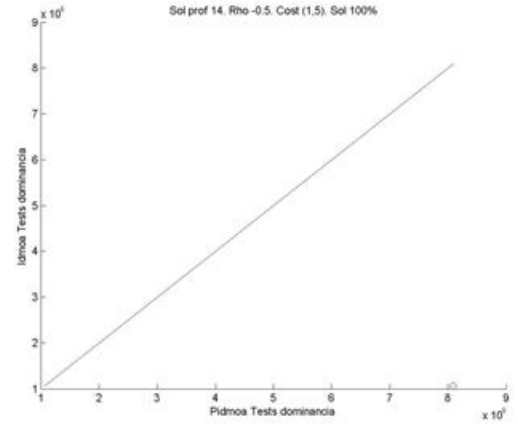
Figura 8.5: Comparativa $IDMOA^*$ vs. $PIDMOA^*$. Árboles infinitos binarios. Tests de dominancia. Profundidad de las soluciones: 14. 1 solución y 10 % de soluciones.



(a) $\rho = 0{,}5$ y 100 % soluciones



(b) $\rho = 0$ y 100 % soluciones



(c) $\rho = -0{,}5$ y 100 % soluciones

Figura 8.6: Comparativa $IDMOA^*$ vs. $PIDMOA^*$. Árboles infinitos binarios. Tests de dominancia. Profundidad de las soluciones: 14. 100 % de soluciones.

apreciar que para el caso de problemas con una única solución (la cual es, obviamente, el único óptimo de Pareto existente), los tests de dominancia realizados por $IDMOA^*$ son inexistentes.

Analizamos a continuación el efecto de la correlación en los resultados obtenidos. Tal y como se comentó en la sección 8.3.3, una correlación negativa fomenta la existencia de un mayor número de vectores de coste no dominados entre sí, independientemente de la cantidad de soluciones existentes. De modo análogo, una correlación positiva reduce la variabilidad de costes entre objetivos, disminuyendo por tanto la cantidad de vectores no dominados entre sí.

En el caso de $IDMOA^*$, para los problemas con una única solución esta correlación no tiene impacto apreciable en su rendimiento, al no haber tests de dominancia asociados a los nodos no finales. A medida que se incrementa el número de soluciones, también lo hará la proporción de estas soluciones que sean óptimos de Pareto. Cuanto menor sea la correlación, mayor será el tamaño de C^*, incrementando la cantidad de tests de dominancia a realizar y, por tanto, el tiempo de procesamiento de $IDMOA^*$. De modo similar, para una correlación positiva, el problema multiobjetivo ve reducida la cantidad de óptimos de Pareto, disminuyendo los tests de dominancia y con ello el tiempo de procesamiento. En cualquier caso, para un algoritmo como $IDMOA^*$, con un comportamiento *cuasi-escalar*, la correlación no es un factor de tanta influencia en el rendimiento del algoritmo como lo puede ser para un enfoque puramente multiobjetivo.

$PIDMOA^*$, al usar una cota de expansión multivectorial, sí se ve más afectado por la correlación entre objetivos. Una correlación negativa provoca un incremento en la cantidad de vectores en dichas cotas, y por lo tanto en la cantidad de tests de dominancia a realizar por cada nodo expandido. Esto se puede apreciar analizando el comportamiento al disminuir la correlación: al pasar de $\rho = 0{,}5$ a $\rho = -0{,}5$) se aprecia un desplazamiento hacia la derecha sobre el eje X de las gráficas de las figuras 8.5 y 8.6, que se corresponde con un mayor número de tests de dominancia. Esto se traduce en mayor tiempo de procesamiento, reflejado en un desplazamiento similar en las gráficas de las figuras 8.3 y 8.4.

8.5.2. Comparativa de algoritmos $IDMOA^*$, $PIDMOA^*$ e $IPID^*$

De los resultados obtenidos en la sección anterior podemos deducir que el cuello de botella de $PIDMOA^*$ es claramente el umbral multivectorial sobre el cual se realizan los tests de dominancia. En las pruebas presentadas en esta sección introducimos $IPID^*$, el cual reduce el umbral a un único vector, disminuyendo drásticamente los tests de dominancia, a costa de un pequeño incremento en la cantidad de nodos reexpandidos.

Experimentos

Una vez definido el algoritmo $IPID^*$, se realizaron una serie de pruebas involucrando a los algoritmos multiobjetivo de profundización iterativa $IDMOA^*$, $PIDMOA^*$ e $IPID^*$. En este caso las pruebas estaban formadas por árboles binarios infinitos (ver sección 8.3.2) con las siguientes características:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 8, 10, 12, 14, 16, 18, 20 y 22 del árbol de búsqueda.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos ($\rho = 0$).
- Heurístico: $\vec{h}(n) = 0, \forall n$, es decir, se realiza una búsqueda ciega. Dado que en pruebas previas no se apreciaron diferencias significativas entre los resultados obtenidos con y sin el heurístico mencionado en el apartado 8.3, no se ha considerado necesario realizar pruebas con información heurística. Nótese que la calidad del heurístico propuesto en 8.3 empeora cuanto mayor es el rango de costes de los objetivos.
- Porcentaje de la cantidad de nodos solución en la profundidad fijada para los mismos. Estos porcentajes son 1 %, 4 %, 7 %, 10 %, 25 %, 40 %, 60 % y 80 %. Tal y como comentamos en la sección 8.3, estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 5 problemas aleatorios diferentes (se presentan las medias).

La principal diferencia con respecto a las pruebas presentadas en la sección 8.5.1 es la ampliación de la cantidad de niveles de profundidad de los nodos solución y los porcentajes de nodos solución al nivel prefijado. Estas ampliaciones permiten estudiar con más detalle el comportamiento de los algoritmos ante una casuística mucho mayor. Por otro lado el rango de costes se ha aumentado, lo que conlleva una mayor variabilidad en los vectores de coste. Esto se traduce en una mayor posibilidad de no dominancia entre vectores y un posible mayor número de óptimos de Pareto, incrementando la dificultad del problema. Por último se ha fijado un factor de correlación cero, al no afectar significativamente a la comparativa entre algoritmos.

Resultados

Los resultados mostrados en esta sección aparecen en (Coego et al., 2012). En las figuras 8.7 y 8.8 se pueden ver los tiempos de procesamiento (medidos en segundos) de

los algoritmos para cada una de las profundidades fijadas para los nodos solución. En el eje X se representa el tanto por ciento de soluciones en el nivel fijado, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo. Nótese que la escala del eje Y varía ligeramente de unas gráficas a otras.

En las gráficas de las figuras 8.9 y 8.10 se representa el tiempo de procesamiento (medido en segundos) de los algoritmos en función de la profundidad de las soluciones para cada porcentaje de soluciones al nivel establecido. En el eje X se representa la profundidad a la cual se encuentran las soluciones, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.

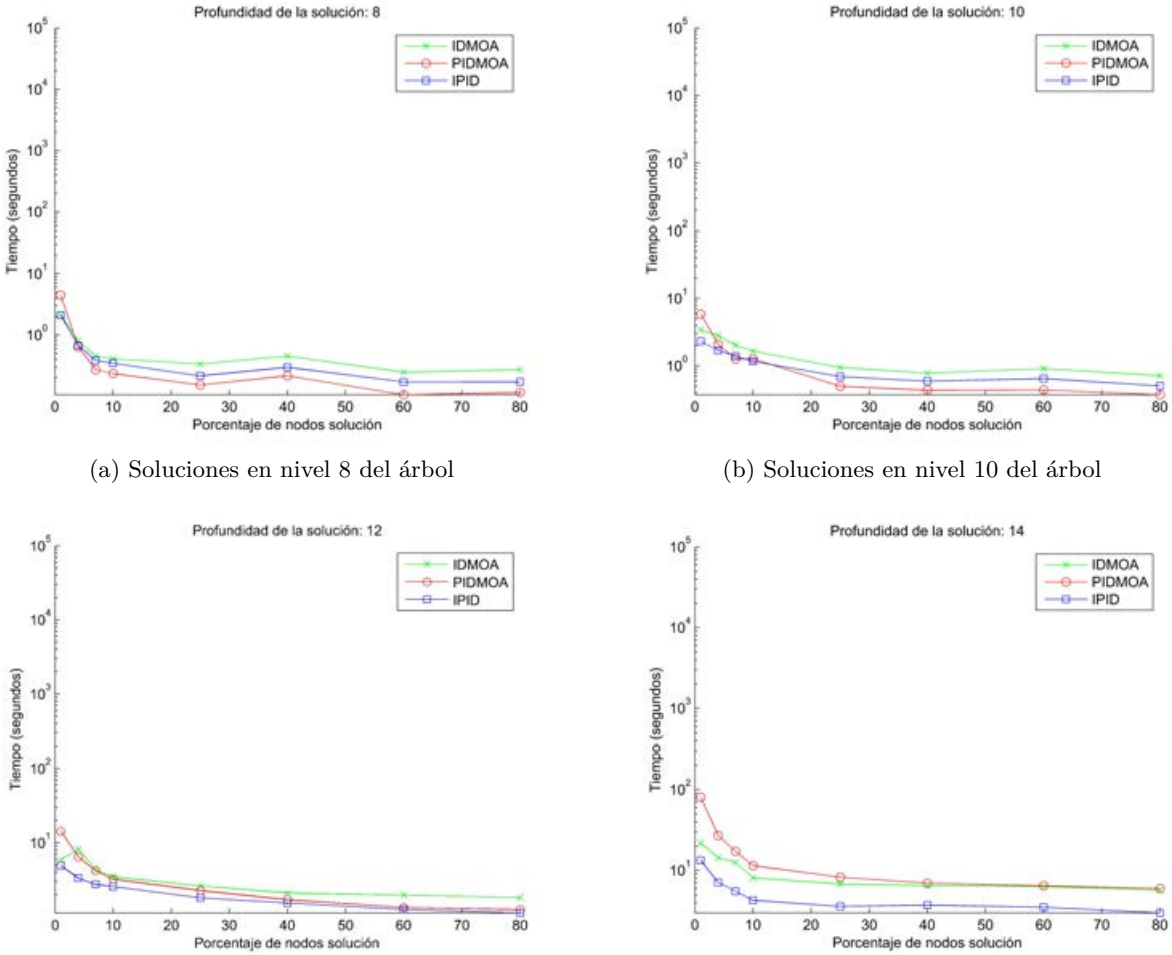
En las gráficas de las figuras 8.11 y 8.12 hemos representado la cantidad de nodos expandidos por cada algoritmo en cuatro instancias de problemas diferentes. Se han seleccionado dos problemas en los cuales las soluciones estaban a una profundidad intermedia (nivel 16), y para cada uno de ellos hemos seleccionado sendos porcentajes de soluciones a dicho nivel 16: porcentaje bajo (4 %) y porcentaje alto (80 %). Se han generado otro par de problemas similares, pero con las soluciones a una mayor profundidad, nivel 22. En el eje X hemos representado las diferentes iteraciones o profundizaciones que realiza el algoritmo, mientras que en el eje Y está representada la cantidad de nodos que el algoritmo ha expandido en la iteración correspondiente. Esta cantidad es acumulativa, siendo el número de nodos representado en la iteración i la cantidad total de nodos expandidos entre las iteraciones 0 e i.

En las gráficas de la figura 8.13 podemos ver los tests de dominancia realizados en cada iteración y la evolución del tamaño de los conjuntos *Threshold* y C^* para los tres algoritmos con las soluciones ubicadas a profundidad 16 y con un 4 % de nodos finales a dicha profundidad. Los datos están promediados sobre 3 instancias de problemas con las características mencionadas.

En la gráfica de tests de dominancia, en el eje X se representan cada una de las iteraciones realizadas por el algoritmo, mientras que en el eje Y se representan la cantidad de tests de dominancia que se han realizado exclusivamente en esa iteración.

En la gráfica de tamaño de los conjuntos *Threshold* y C^*, en el eje X se representan las iteraciones realizadas por cada algoritmo, mientras que en el eje Y se representan el número de vectores que contiene cada uno de los conjuntos mencionados exclusivamente en dicha iteración.

En las figuras 8.14 y 8.15 se muestran los tests de dominancia y tamaños de los conjuntos *Threshold* y C^* para resultados obtenidos sobre instancias de problemas con un 80 % de nodos finales a profundidad 16 y un 80 % de nodos finales a profundidad 22 respectivamente.



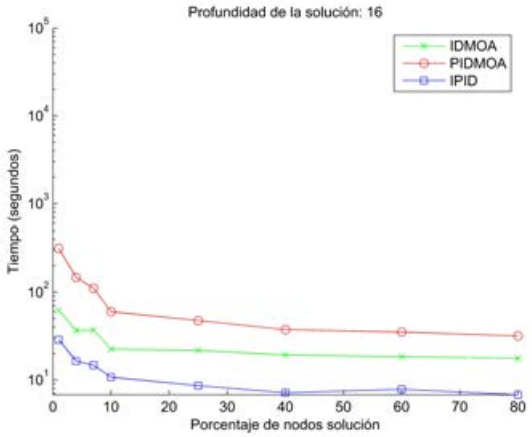
(a) Soluciones en nivel 8 del árbol

(b) Soluciones en nivel 10 del árbol

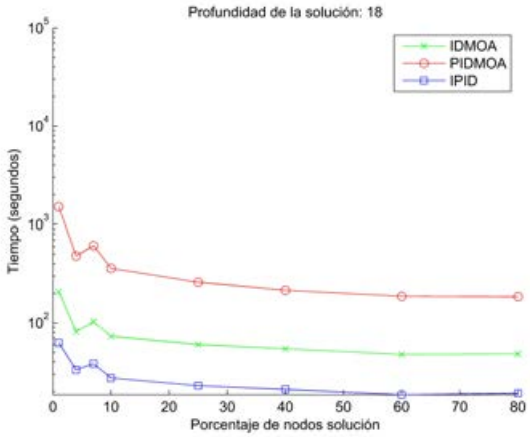
(c) Soluciones en nivel 12 del árbol

(d) Soluciones en nivel 14 del árbol

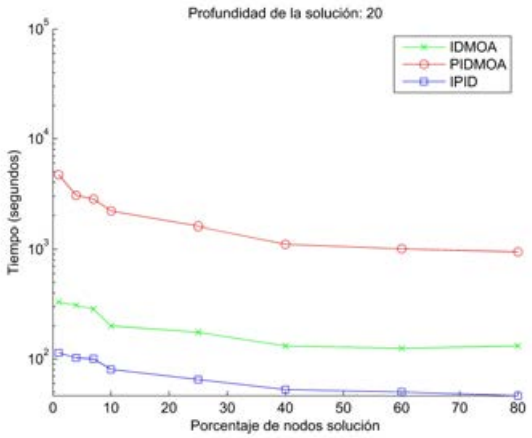
Figura 8.7: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones. $\rho = 0$. Niveles de profundidad de las soluciones: 8, 10, 12, 14.



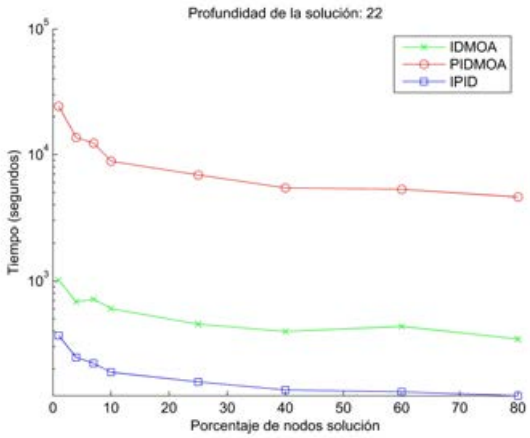
(a) Soluciones en nivel 16 del árbol



(b) Soluciones en nivel 18 del árbol

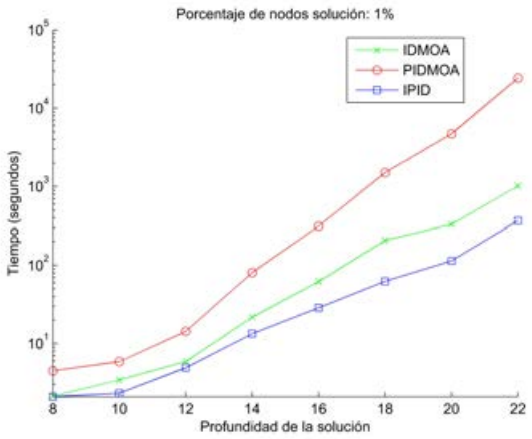


(c) Soluciones en nivel 20 del árbol

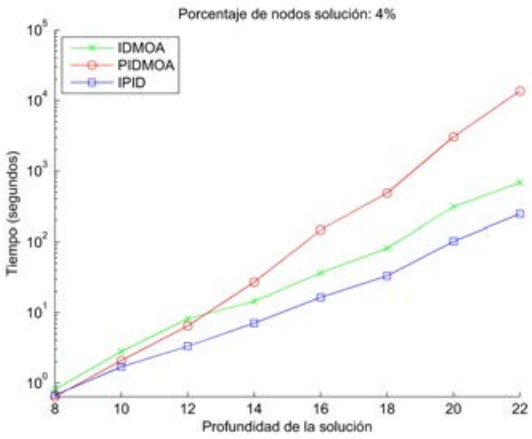


(d) Soluciones en nivel 22 del árbol

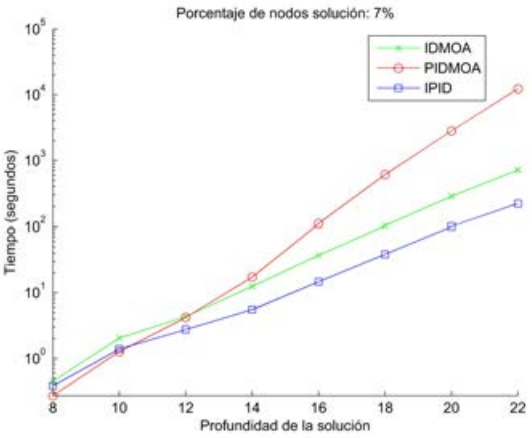
Figura 8.8: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones. $\rho = 0$. Niveles de profundidad de las soluciones: 16, 18, 20, 22.



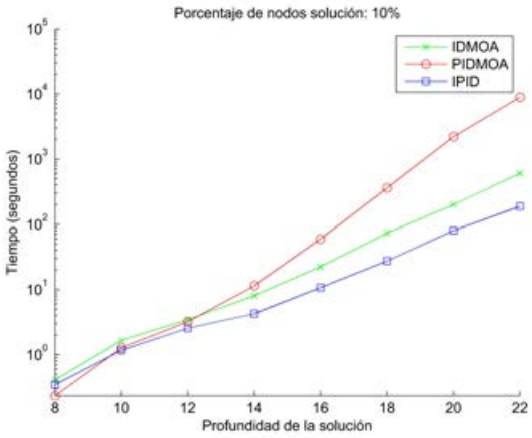
(a) 1 % de soluciones en nivel fijado



(b) 4 % de soluciones en nivel fijado

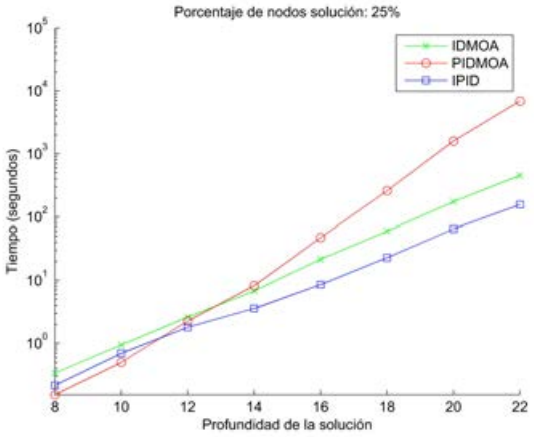


(c) 7 % de soluciones en nivel fijado

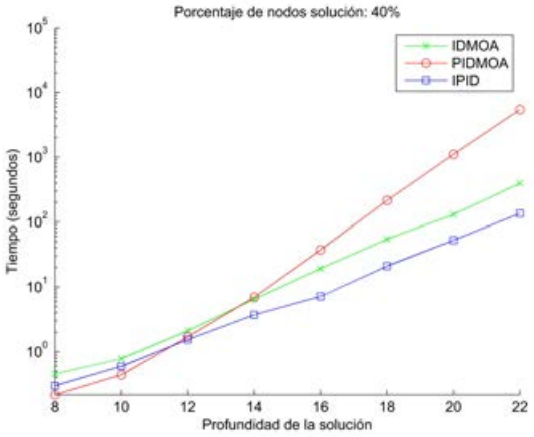


(d) 10 % de soluciones en nivel fijado

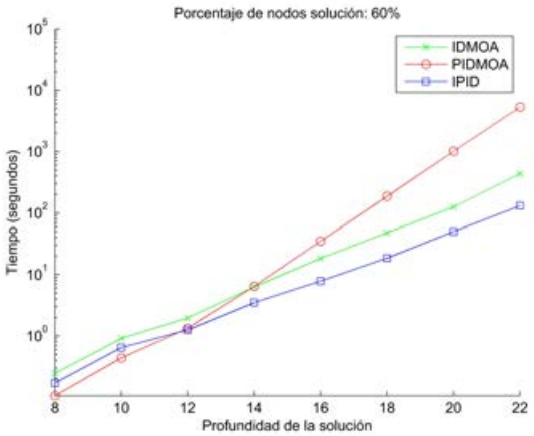
Figura 8.9: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de la profundidad de las soluciones. Porcentajes de soluciones: 1 %, 4 %, 7 %, 10 %.



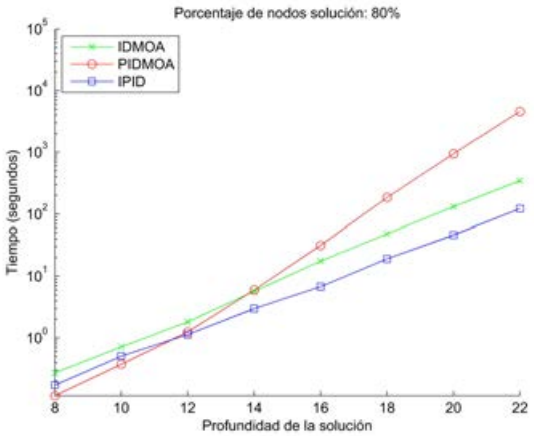
(a) 25 % de soluciones en nivel fijado



(b) 40 % de soluciones en nivel fijado

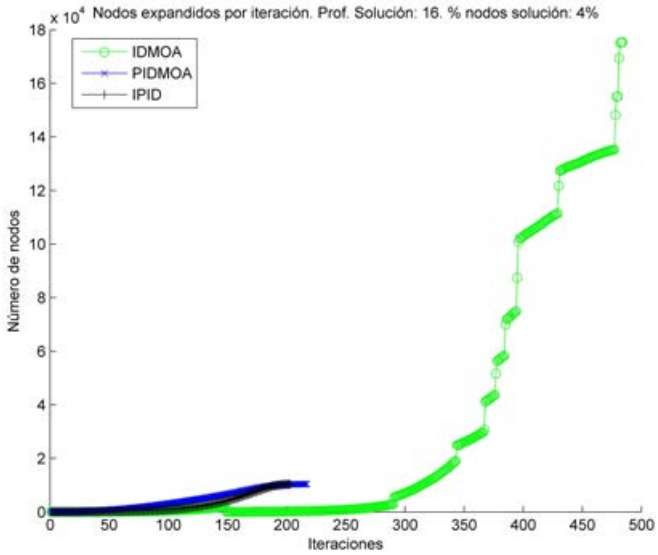


(c) 60 % de soluciones en nivel fijado

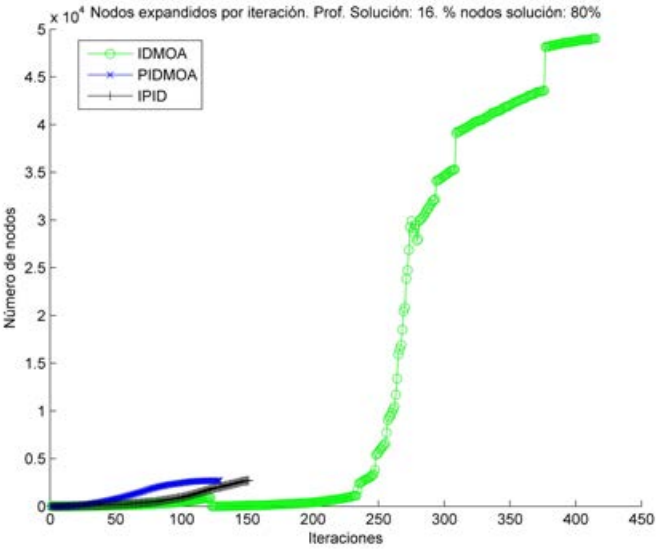


(d) 80 % de soluciones en nivel fijado

Figura 8.10: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de la profundidad de las soluciones. Porcentajes de soluciones: 25 %, 40 %, 60 %, 80 %.

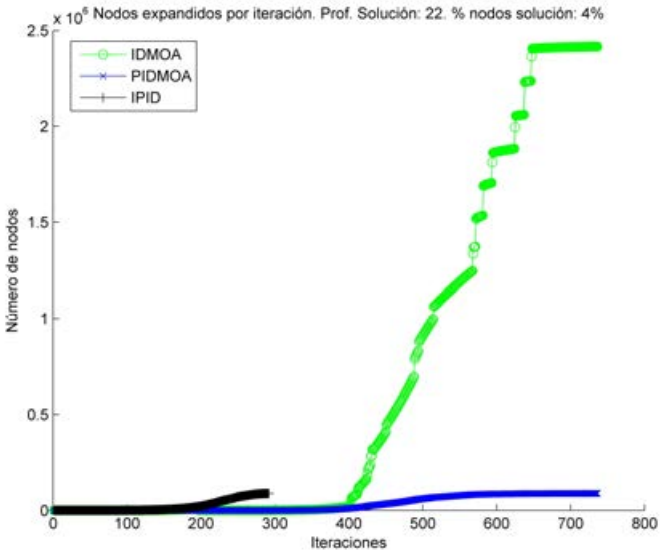


(a) Profundidad soluciones 16. 4 % nodos finales

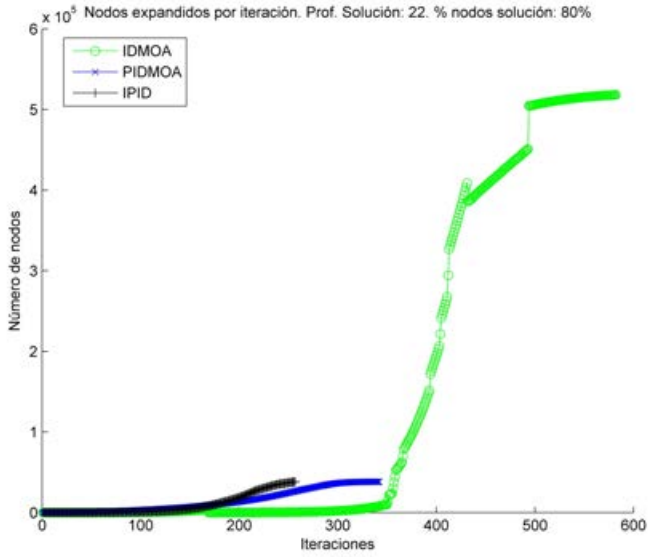


(b) Profundidad soluciones 16. 80 % nodos finales

Figura 8.11: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Nodos expandidos en función de las iteraciones realizadas en 4 instancias de problemas. Soluciones a profundidad 16.

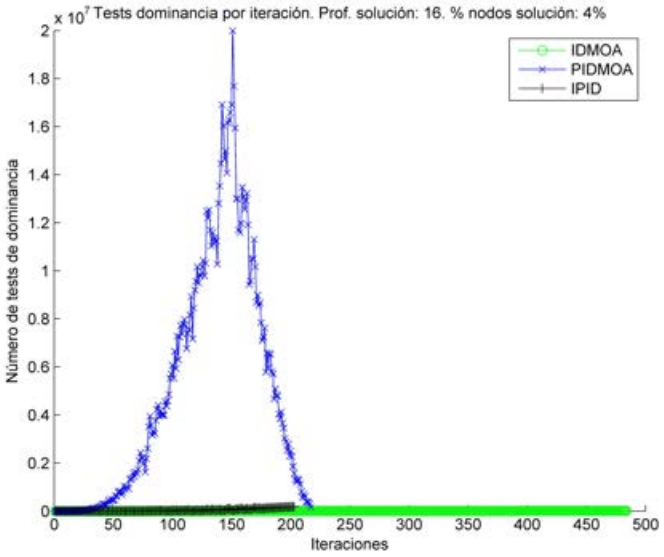


(a) Profundidad soluciones 22. 4 % nodos finales

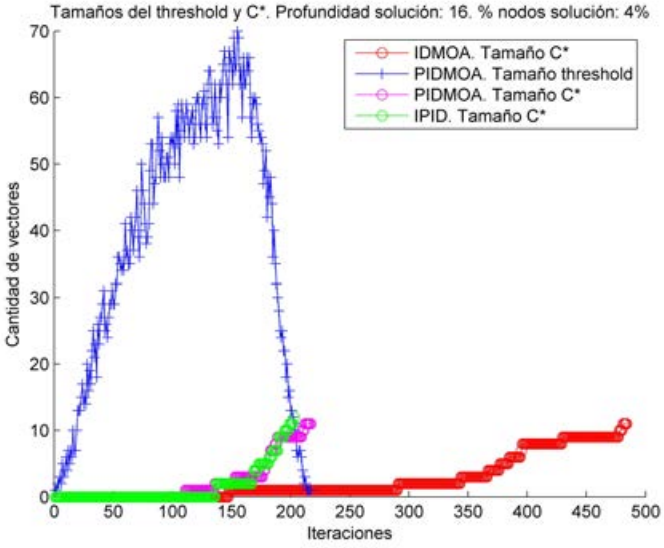


(b) Profundidad soluciones 22. 80 % nodos finales

Figura 8.12: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Nodos expandidos en función de las iteraciones realizadas en 4 instancias de problemas. Soluciones a profundidad 22.

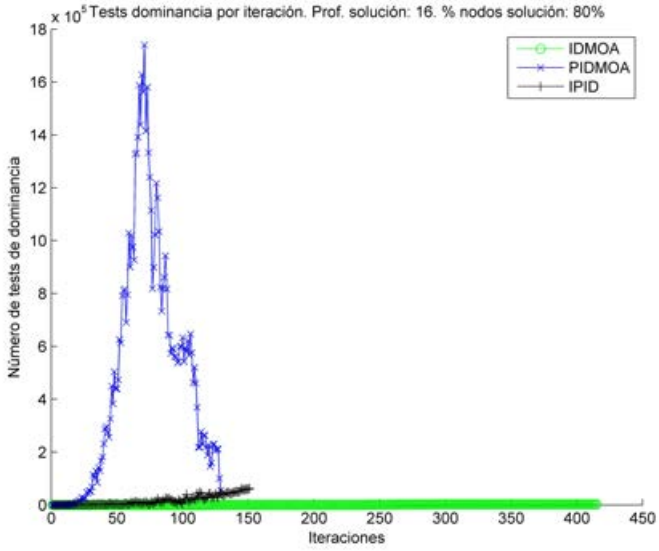


(a) Tests de dominancia por iteración.

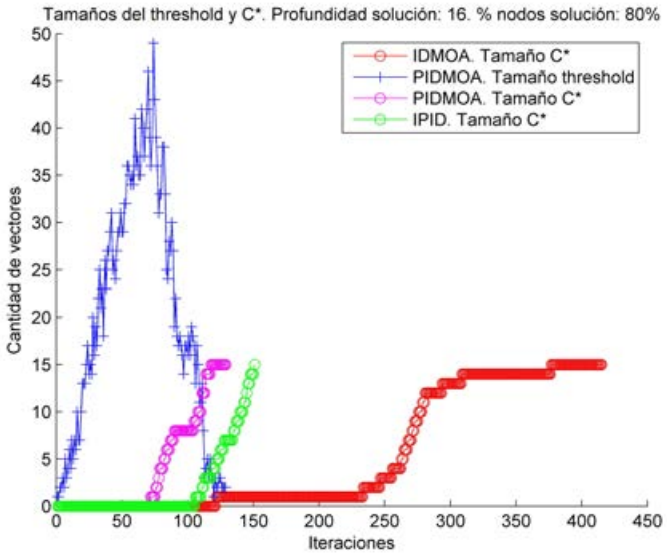


(b) Tamaño de los conjuntos *Threshold* y C^*.

Figura 8.13: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tests de dominancia y tamaño de *Threshold* y C^*. 4 % de nodos finales a profundidad 16.

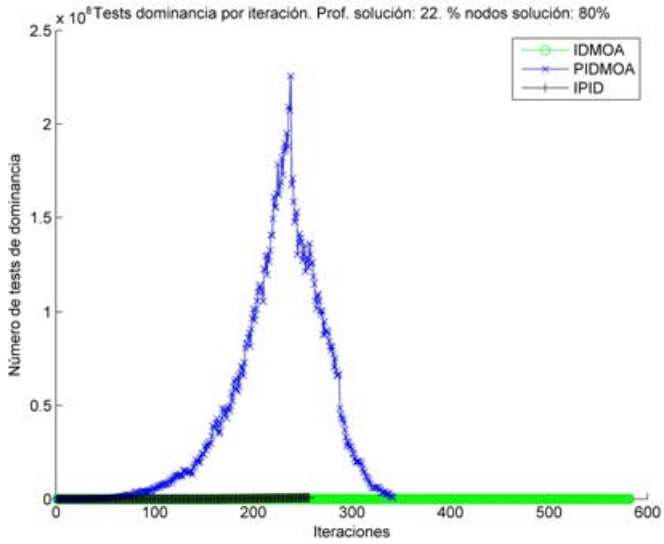


(a) Tests de dominancia por iteración.

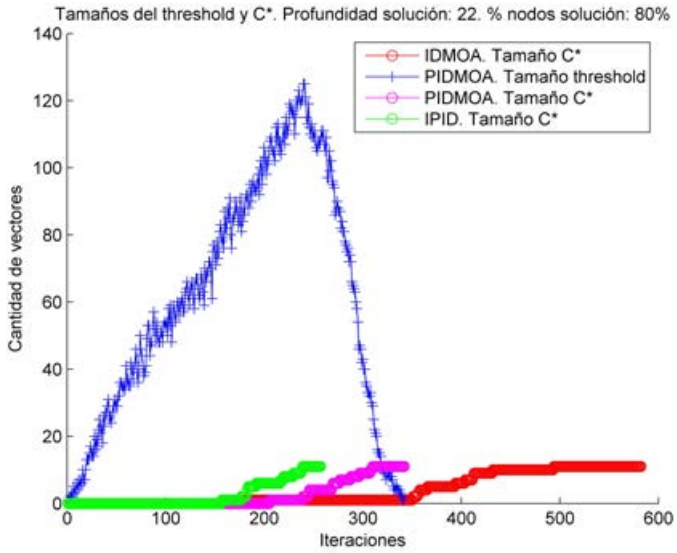


(b) Tamaño de los conjuntos *Threshold* y C^*.

Figura 8.14: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tests de dominancia y tamaño de *Threshold* y C^*. 80 % de nodos finales a profundidad 16.



(a) Tests de dominancia por iteración.



(b) Tamaño de los conjuntos *Threshold* y C^*.

Figura 8.15: Comparativa $IDMOA^*$ vs. $PIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tests de dominancia y tamaño de *Threshold* y C^*. 80 % de nodos finales a profundidad 22.

Análisis

Como podemos apreciar en las gráficas de las figuras 8.7 y 8.8, todos los algoritmos se comportan progresivamente mejor cuando aumenta el número de soluciones. En estos casos es más probable que localicen una solución en las etapas iniciales, la cual generalmente les servirá de cota eficiente para ir podando e incluso descartando ramas.

A profundidades bajas (figuras 8.7a, 8.7b), el algoritmo $PIDMOA^*$ es el más eficiente en cuanto el porcentaje de soluciones supera el $10 - 20\,\%$ de nodos finales. Esto se debe a que, aunque todos los algoritmos localizan soluciones rápidamente, las que localiza $PIDMOA^*$ son soluciones óptimas, con lo cual las podas que podrá realizar en el árbol de búsqueda serán más numerosas que las que puedan realizar $IDMOA^*$ o $IPID^*$.

De todos modos esas profundidades no conllevan un coste computacional alto. A medida que vamos ubicando las soluciones en niveles más alejados del nodo raíz, el comportamiento de los algoritmos se mantiene en lo que respecta al porcentaje de soluciones. A menor número de soluciones, peor rendimiento de los tres algoritmos, debido a que les cuesta más localizar esas cotas efectivas (soluciones, tanto óptimas como subóptimas) que generen gran cantidad de podas en el proceso de búsqueda. Además la situación tiende a estabilizarse entre los tres algoritmos, presentándose $IPID^*$ como el más eficiente, $PIDMOA^*$ como el más ineficiente e $IDMOA^*$ en un término medio.

En las gráficas de las figuras 8.7 y 8.8 se observa, como es obvio, que a mayor profundidad de las soluciones, mayor dificultad del problema y por lo tanto mayor tiempo de computación. Sin embargo podemos apreciar que el crecimiento del tiempo necesario para $PIDMOA^*$ es mayor que el de $IDMOA^*$ ó $IPID^*$ para todos los porcentajes de soluciones. Con problemas que tengan soluciones ubicadas a un nivel de profundidad igual o superior a 14, $PIDMOA^*$ es ya el peor de los tres algoritmos. En cuanto a $IDMOA^*$ e $IPID^*$, se observa que la pendiente de crecimiento de tiempo es similar, con una clara ventaja a favor de $IPID^*$ para todos los problemas con soluciones a profundidad mayor de 14. El *ranking* de algoritmos en cuanto a coste temporal es $IPID^*$, $IDMOA^*$, $PIDMOA^*$, excepto en el caso ya mencionado de baja profundidad y alto porcentaje de soluciones. El mejor comportamiento de $PIDMOA^*$ frente a $IDMOA^*$ que se aprecia en la figura 8.7d, en problemas similares a los presentados en la sección 8.5.1 y donde $IDMOA^*$ tenía mejor rendimiento, tiene su explicación en el rango de costes empleado. Los problemas presentados en esta sección emplean el rango de costes $[1, 50]$, frente al rango $[1, 5]$ usado en la sección 8.5.1. Este rango propicia la aparición de mayor variabilidad de costes, influyendo negativamente en la cantidad de reexpansiones realizadas por $IDMOA^*$.

Para poder analizar con mayor detalle el motivo de los comportamientos anteriores, se realizaron una serie de medidas adicionales en varios problemas más o menos representativos de las diferentes configuraciones. No se ha realizado un estudio detallado para profundidades bajas, dado que no son problemas de una dificultad significativa y

la variabilidad en los resultados obtenidos es en general muy alta.

Las gráficas de las figuras 8.11 y 8.12 muestran los nodos expandidos por los algoritmos en diferentes instancias de problemas. Aunque, como ya se mencionó en la sección 8.4, la cantidad de nodos expandidos no es por sí sola una medida completa de rendimiento en el caso multiobjetivo, se representan para apreciar el efecto que tiene el uso de umbrales vectoriales en $PIDMOA^*$ e $IPID^*$, frente al uso de umbrales escalares de $IDMOA^*$. En cualquiera de los cuatro casos analizados, la cantidad de iteraciones (o profundizaciones) realizadas por $IDMOA^*$ es claramente superior a la de los otros dos algoritmos, excepto para profundidad 22 y 4 % de nodos solución, donde $IDMOA^*$ y $PIDMOA^*$ tienen una cantidad de iteraciones similar.

El valor más significativo de estas gráficas es el número de nodos procesados. Mientras $PIDMOA^*$ e $IPID^*$, con más o menos iteraciones, mantienen un perfil bajo de expansión de nodos, los valores de $IDMOA^*$ se disparan aproximadamente a partir de la mitad de las iteraciones realizadas. Este comportamiento se debe a ese procesamiento secuencial de los objetivos por parte de $IDMOA^*$. Cuando está procesando el primer objetivo no realiza ningún tipo de comparaciones relacionadas con el segundo objetivo. Esto hace que en esa primera etapa se limite a buscar un solución óptima en lo que respecta al primer objetivo.

Supongamos que tenemos la siguiente frontera de Pareto: $C^* = \{(1,n),(2,n-1),(3,n-2),\ldots,(n-2,3),(n-1,2),(n,1)\}$. El comportamiento de $IDMOA^*$ cuando procesa el primer objetivo es similar al que tendría el algoritmo IDA^*, es decir, localiza una solución con el coste más bajo posible para ese primer objetivo. En nuestro ejemplo, la solución obtenida sería la asociada al vector de coste $(1,n)$. Dado que esta solución no tendrá un coste relativamente muy elevado para ese objetivo, al ser óptima para el mismo, se localizará en niveles del árbol de búsqueda cercanos al nodo raíz, con lo cual la cantidad de nodos expandidos no será muy alta.

Una vez localizada esta solución, $IDMOA^*$ procede a trabajar con el segundo objetivo, usando un umbral escalar como cota para dicho objetivo, pero tomando también el valor n como cota superior global, superada la cual, el procesamiento de este objetivo finaliza. Como se puede apreciar en la frontera de Pareto de ejemplo indicada en el párrafo anterior, la cantidad de soluciones óptimas a procesar a partir de ahora es mucho mayor (junto con las soluciones subóptimas que también puedan aparecer durante la búsqueda). Como los valores de ambos objetivos se van incrementando, $IDMOA^*$ profundiza más en el árbol de búsqueda y la cantidad de nodos a explorar se dispara, agravado todo esto por el hecho de no haber aprovechado la información del segundo objetivo cuando se estaba procesando el primero.

Pero aunque el número de iteraciones y de nodos expandidos de $IDMOA^*$ es mucho mayor que el de $PIDMOA^*$, en las gráficas de tiempo (figuras 8.8 y 8.10) hemos constatado que el rendimiento de $PIDMOA^*$ es ostensiblemente inferior. El motivo, como ya avanzamos, es la naturaleza de los tests de dominancia: vectoriales para $PIDMOA^*$ y escalares para $IDMOA^*$. El tiempo de procesamiento de un nodo para $IDMOA^*$ es prácticamente constante y bajo, al tener que comparar el valor en el

vector de coste para el objetivo procesado con el valor del umbral. En el caso de $IPID^*$ podemos observar que el número de nodos expandidos es similar al de $PIDMOA^*$, aunque en un número de iteraciones menor.

En las figuras 8.13, 8.14 y 8.15 se muestran los test de dominancia realizados por los algoritmos en diferentes instancias de problemas. Estas figuras únicamente reflejan los tests de tipo vectorial. En el caso de $IDMOA^*$, estos tests son inexistentes hasta que el algoritmo encuentra la primera solución (un óptimo de Pareto para el primer objetivo). A partir de entonces, el vector de coste de cualquier solución que se localice durante la búsqueda tiene que ser comparado con los vectores de coste de las soluciones ya localizadas. Estos son los únicos tests de dominancia vectoriales que realiza $IDMOA^*$.

En el caso de $IPID^*$, la convergencia del algoritmo a la frontera de Pareto es más rápida. Expande muchos menos nodos que $IDMOA^*$ para localizar todas las soluciones óptimas, como se reflejaba en las figuras 8.11 y 8.12, y además cada nodo expandido únicamente tiene que ser comparado contra un único vector umbral, reduciendo considerablemente los tests de dominancia. Por otro lado, aunque puede localizar temporalmente soluciones subóptimas, la aproximación que hace inicialmente de la frontera de Pareto es más exacta, puesto que tiene en cuenta todos los objetivos simultáneamente, con lo cual el algoritmo finaliza en un menor número de iteraciones.

En el polo opuesto a $IDMOA^*$ e $IPID^*$, en lo que a tests de dominancia se refiere, se encuentra $PIDMOA^*$. Como se puede apreciar en las figuras 8.13, 8.14 y 8.15, este algoritmo presenta dos comportamientos muy diferentes durante el proceso de búsqueda: en primer lugar hay un crecimiento muy acusado de los tests de dominancia en las primeras iteraciones, hasta que localiza una solución óptima. Mientras no sucede esto, el tamaño del conjunto *Threshold* crece considerablemente, y con ello los tests de dominancia que son necesarios para procesar cualquier nodo que se haya generado en el árbol de búsqueda.

Una vez que se localiza la primera solución óptima, el número de podas realizadas en el árbol aumenta, disminuyendo por lo tanto la cantidad de tests de dominancia realizada contra el umbral. Este decrecimiento se acentúa más a medida que se localizan más óptimos de Pareto.

Este comportamiento queda ratificado por las gráficas de esas mismas figuras, donde se muestra la evolución del tamaño de los conjuntos *Threshold* y C^* en diferentes instancias de problemas.

En cuanto al tamaño del conjunto C^*, podemos apreciar en la gráfica que $IDMOA^*$ localiza las soluciones óptimas más gradualmente que los otros dos algoritmos, los cuales, una vez que han localizado la primera solución (óptima en el caso de $PIDMOA^*$, no necesariamente óptima en el caso de $IPID^*$), convergen mucho mas rápidamente al conjunto final.

En la evolución del tamaño del umbral de $PIDMOA^*$ se puede apreciar, comparando estas gráficas con las equivalentes de tests de dominancia, que el número de tests crece proporcionalmente al tamaño del umbral. El inicio del decrecimiento coincide con la localización de la primera solución óptima, momento a partir del cual se podarán

muchas ramas de modo definitivo al estar dominadas por soluciones ya encontradas, disminuyendo asimismo el tamaño del *Threshold*.

El compromiso que mantiene $IPID^*$ entre el procesamiento simultáneo de objetivos y el mantenimiento de una cota fácilmente manejable en cuanto a tests de dominancia lo convierte en el algoritmo más eficiente. Este comportamiento se mantiene en la gran mayoría de problemas generados, a excepción de algunos de dificultad baja poco representativos.

8.5.3. Comparativa de algoritmos $IDMOA^*$, $LEXIDMOA^*$ e $IPID^*$

En esta sección se evaluarán los algoritmos de profundización iterativa con cota de expansión más eficiente. Descartado $PIDMOA^*$ por su bajo rendimiento, al tener que realizar un gran número de tests de dominancia, se evalúan $IDMOA^*$, $LEXIDMOA^*$ e $IPID^*$. El primero mantiene un umbral escalar, mientras que los otros dos usan un umbral formado por un único vector y calculado como el mejor léxicográfico y el punto ideal, respectivamente, de la frontera de Pareto de la iteración actual.

Experimentos

Para la evaluación de los algoritmos $IDMOA^*$, $LEXIDMOA^*$ e $IPID^*$ se emplearon una serie de árboles infinitos aleatorios (ver sección 8.3.2) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 16, 18, 20 y 22 del árbol de búsqueda.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos.
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Porcentaje de la cantidad de nodos solución en la profundidad fijada para los mismos. Estos porcentajes son 1 %, 10 %, 25 %, 40 %, 60 %, 80 % y 100 %. Este último caso representa un árbol finito. Tal y como comentamos en la sección 8.3, estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 10 problemas aleatorios diferentes (se presentan las medias).

En esta batería de problemas, a diferencia de los experimentos presentados en la sección 8.5.2, optamos por descartar los problemas con nodos solución ubicados a

profundidades más cercanas al nodo raíz, al tener en general unos tiempos de resolución más bajos y ser los resultados poco significativos. Asimismo se incluyen árboles finitos, al ser todos los nodos del nivel prefijado una solución del problema. Por último, se optó por establecer un límite temporal para cada uno de los experimentos, igual a $5 \times t_{mejor}$, donde t_{mejor} es, en este caso, el tiempo medio obtenido por $IPID^*$ para cada uno de los diferentes tipos de problemas (al ser $IPID^*$ el algoritmo más eficiente analizado hasta el momento).

Resultados

En la figura 8.16 se pueden ver los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada una de las profundidades fijadas para los nodos solución. En el eje X se representa el tanto por ciento de soluciones en el nivel fijado, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.

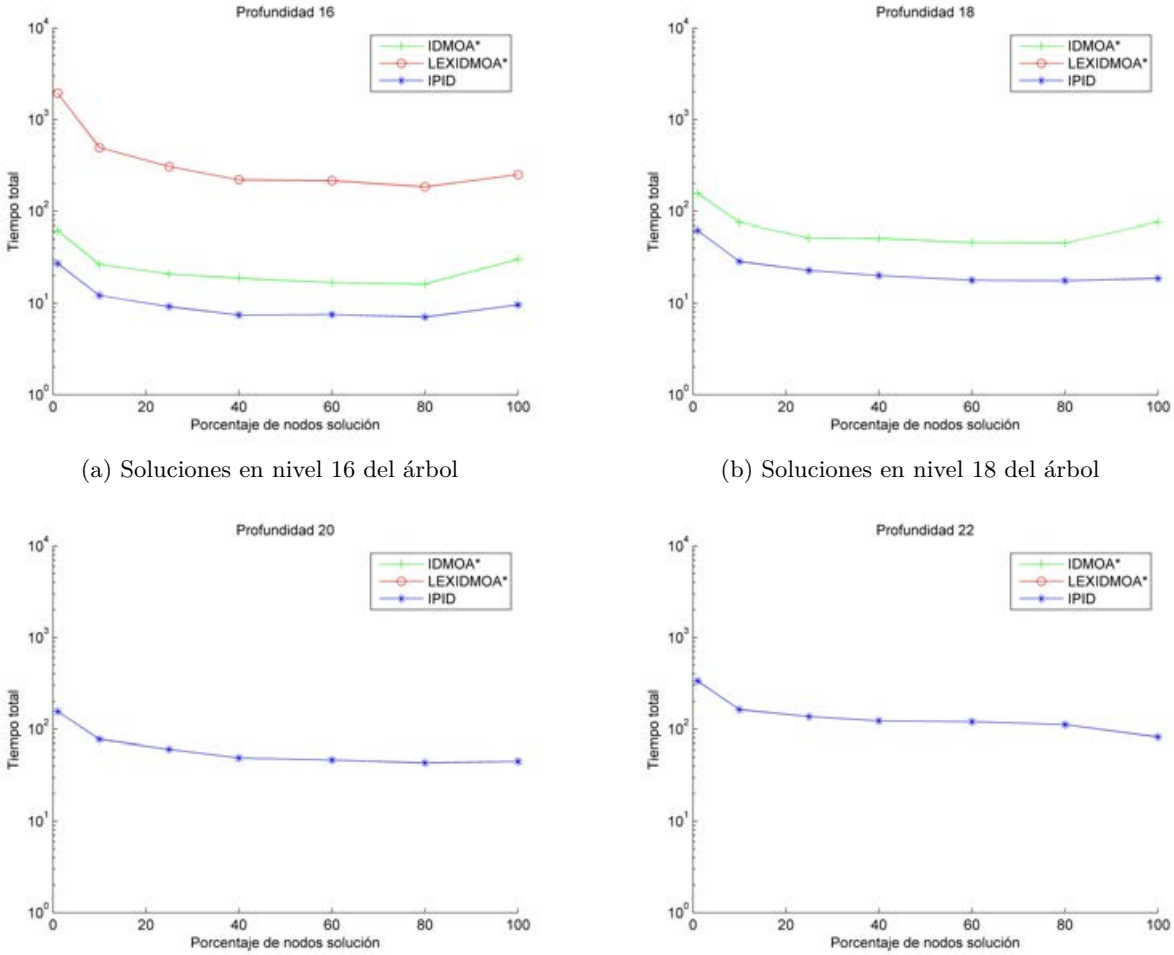
En la figura 8.17 se pueden ver los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada uno de los porcentajes de nodos solución fijados para los mismos experimentos. En el eje X se representa la profundidad a la cual se encuentran las soluciones, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.

Análisis

Al igual que en los experimentos planteados en la secciones anteriores, los algoritmos funcionan en general mejor a medida que aumenta el número de soluciones disponibles al nivel fijado. Esto se debe a que las soluciones sirven de cota superior a la expansión del árbol de búsqueda. La mejora es sustancial para los primeros porcentajes, estabilizándose a medida que poblamos el nivel fijado con más nodos meta. Esto se debe a que, con un porcentaje elevado de soluciones finales, el tiempo necesario para localizar la primera solución disminuye drásticamente. Esta mejora es incluso mayor en algoritmos que pueden localizar, aunque sea de modo temporal, soluciones subóptimas (como es el caso de $IPID^*$).

Como se puede observar en las gráficas de la figura 8.16, $LEXIDMOA^*$ únicamente ha sido capaz de finalizar los problemas con soluciones fijadas a nivel 16, mientras que $IDMOA^*$ ha conseguido completar también los de nivel 18 (debido a los límites de tiempo fijados, iguales a 5 veces el tiempo de resolución de $IPID^*$). $IPID^*$, por su parte, mantiene aproximadamente un incremento de medio orden de magnitud por cada incremento de dos niveles en la profundidad de las soluciones. Esto hace que haya solucionado en tiempos aceptables las baterías de problemas hasta nivel 24 (en la figura 8.16 se muestran únicamente los problemas hasta nivel 22, al ser $IPID^*$ el único que ha resuelto problemas de nivel 24).

En el caso de $LEXIDMOA^*$, el umbral para la iteración $i+1$ se calcula como el mejor vector lexicográfico de la frontera de Pareto para la iteración i. Este orden



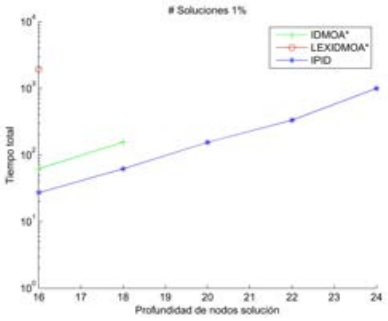
(a) Soluciones en nivel 16 del árbol

(b) Soluciones en nivel 18 del árbol

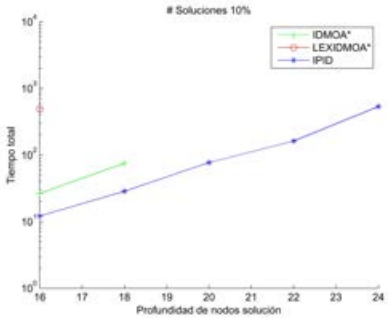
(c) Soluciones en nivel 20 del árbol

(d) Soluciones en nivel 22 del árbol

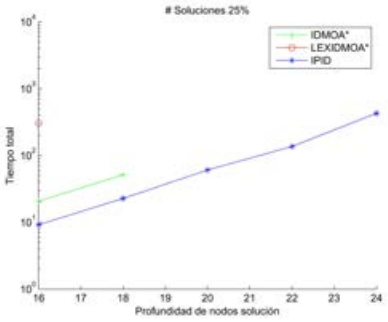
Figura 8.16: Comparativa $IDMOA^*$ vs. $LEXIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para distintos niveles de profundidad de las soluciones.



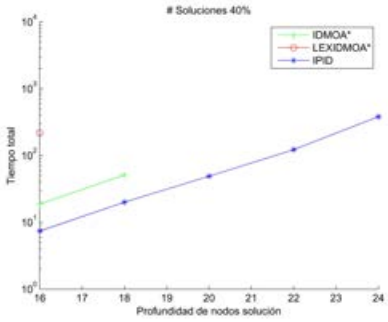
(a) 1 % de soluciones



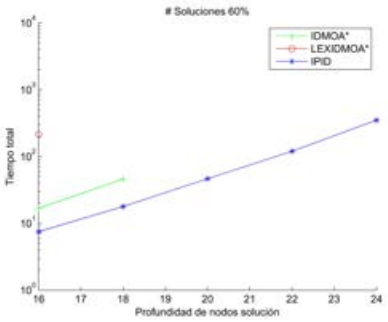
(b) 10 % de soluciones



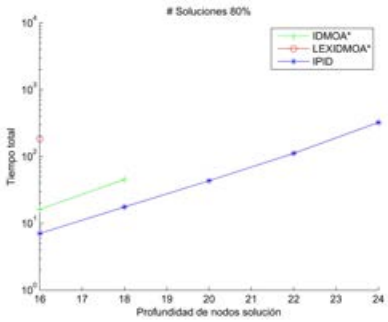
(c) 25 % de soluciones



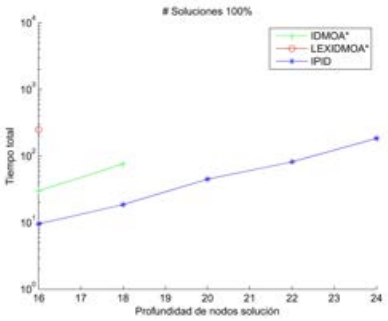
(d) 40 % de soluciones



(e) 60 % de soluciones



(f) 80 % de soluciones



(g) 100 % de soluciones

Figura 8.17: Comparativa $IDMOA^*$ vs. $LEXIDMOA^*$ vs. $IPID^*$. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de la profundidad de las soluciones para distintos porcentajes de soluciones.

lexicográfico propicia que inicialmente el algoritmo emplee como umbrales para la expansión vectores con valores de coste bajos para el primer objetivo. Sólo en caso de igualdad en el primer objetivo, empleará $LEXIDMOA^*$ como discriminante el coste del segundo objetivo.

En la práctica esto es similar al mecanismo empleado por $IDMOA^*$, el cual procesa inicialmente el primer objetivo y luego el segundo. Es decir, el orden de *avance* de las componentes durante la expansión del árbol es el mismo. Sin embargo, el hecho de que $LEXIDMOA^*$ mantenga un umbral vectorial hace que los tests de dominancia contra el umbral sean computacionalmente más costosos, motivo por el cual su rendimiento es peor que el de $IDMOA^*$, aún cuando elimina los tests de dominancia contra el conjunto de soluciones localizadas (al localizar únicamente soluciones óptimas).

Aunque tanto $LEXIDMOA^*$ como $IPID^*$ mantienen ambos un umbral formado por un único vector, el modo de calcularlo influye notablemente en la diferencia de rendimiento entre ambos. $IPID^*$, a diferencia de $LEXIDMOA^*$, mantiene un umbral cuyo cálculo tiene en cuenta simultáneamente los costes de todos los objetivos. Esto propicia que el avance en la expansión del árbol de búsqueda sea más rápido, al estar guiado por todos los objetivos. Sólo cuando el vector de coste de un nodo es estrictamente peor que el vector umbral en todas las componentes se discontinúa un nodo. $LEXIDMOA^*$, por el contrario, discontinúa la búsqueda cuando, por ejemplo, la primera componente del vector de coste sea peor que la primera componente del vector usado como umbral, aún cuando el resto de componentes sí puedan ser mejores.

Por otro lado, en las gráficas de la figura 8.17 se aprecia como $IPID^*$ mantiene la pendiente en el tiempo de procesamiento a medida que aumenta la profundidad a la cual se encuentran las soluciones en el árbol de búsqueda. Esto se debe a que, aunque la frontera de Pareto puede aumentar considerablemente con la profundidad del árbol, $IPID^*$ la transforma en un único vector. Por el contrario, $PIDMOA^*$, tal y como vimos en la sección 8.5.2, incrementa considerablemente el tamaño de su umbral multivectorial y, por consiguiente, la cantidad de tests de dominancia, a medida que aumentamos la profundidad de las soluciones.

8.5.4. Conclusiones de la comparativa

En esta sección hemos realizado una evaluación detallada de diversos algoritmos multiobjetivo exactos basados en profundización iterativa. De modo general podemos concluir que la cantidad de tests de dominancia a realizar durante la expansión del árbol de búsqueda es un factor determinante en el rendimiento del algoritmo. Esto hace que el empleo de umbrales multivectoriales en el caso de $PIDMOA^*$ lo convierta en una opción ineficiente en problemas con árboles de búsqueda relativamente profundos.

El algoritmo $IPID^*$ es, por tanto, la alternativa más eficiente entre las tres generalizaciones multiobjetivo directas ($IPID^*$, $LEXIDMOA^*$ y $PIDMOA^*$) de la profundización iterativa. Los análisis muestran además que $IPID^*$ supera también a $IDMOA^*$, un algoritmo secuencial basado en la aplicación repetida del algoritmo

IDA^*.

$LEXIDMOA^*$ e $IPID^*$ emplean ambos una cota de un solo vector. En el caso de $LEXIDMOA^*$, esta cota se calcula mediante un orden lexicográfico, lo cual asigna diferente importancia a los objetivos, priorizando, por ejemplo, la componente de coste asociada al primer objetivo antes que el resto. $IPID^*$, por el contrario, contempla de modo simultáneo todos los objetivos, consiguiendo un avance más rápido dentro del árbol de búsqueda.

8.6. Comparativa de algoritmos basados en *RBFS*

En esta sección se realiza una evaluación de los diferentes algoritmos multiobjetivo exactos basados en *RBFS*. Concretamente se utilizarán las dos aportaciones realizadas en esta tesis, *Pareto-MO-RBFS* e *IP-MO-RBFS*, así como el algoritmo $MOMA^*0$ y una variante del mismo, que hemos denominado $MOMA^*0$-*lineal*. Como ya vimos en el Capítulo 5, *Pareto-MO-RBFS* emplea una cota de expansión multivectorial, mientras que *IP-MO-RBFS* usa como cota el punto ideal de la frontera de Pareto del árbol explorado en el instante actual.

El algoritmo $MOMA^*0$, como vimos en la sección 3.6.3, es una variante de *RBFS* que calcula esta cota como el mejor lexicográfico de la frontera de Pareto. Un nodo se expande siempre y cuando sea mejor lexicográfico que su cota. El cuarto algoritmo contemplado en este banco de pruebas, $MOMA^*0$-*lineal*, es una adaptación de $MOMA^*0$ que emplea un orden lineal para la expansión.

Recordemos que un vector de coste $\vec{v_1} = (c_1^1, \ldots, c_1^k)$ es *mejor lineal* que otro vector $\vec{v_2} = (c_2^1, \ldots, c_2^k)$ si $\sum_{i=1}^{k} c_i^1 < \sum_{i=1}^{k} c_i^2$. En $MOMA^*0$-*lineal*, la cota de expansión es un único vector, al igual que en $MOMA^*0$, pero se calcula como el mejor lineal de la frontera de Pareto del árbol de búsqueda. Además, el algoritmo expande un nodo siempre y cuando sea mejor lineal que su cota.

Experimentos

Para la evaluación de los algoritmos $MOMA^*0$, $MOMA^*0$-*lineal*, *Pareto-MO-RBFS* e *IP-MO-RBFS* se emplearon una serie de árboles infinitos aleatorios (ver sección 8.3.2) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 16, 18, 20, 22 y 24 del árbol de búsqueda.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos.

- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Porcentaje de la cantidad de nodos solución en la profundidad fijada para las soluciones. Estos porcentajes son 1 %, 10 %, 25 %, 40 %, 60 %, 80 % y 100 %. Este último caso representa un árbol finito. Tal y como comentamos en la sección 8.3, estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 10 problemas aleatorios diferentes (se presentan las medias).

En este caso se optó por establecer un límite temporal para los experimentos de profundidades mayores que 16, igual a $5 \times t_{ip}$, donde t_{ip} es, en este caso el tiempo medio obtenido por *IP-MO-RBFS* para cada uno de los diferentes tipos de problemas. Esto se realizó por analogía con los experimentos de algoritmos basados en profundización iterativa, conjeturándose que en este caso la cota con *Punto Ideal* aplicada en *IP-MO-RBFS* sería la mejor opción (lo cual, como veremos, se confirmó con los experimentos).

Resultados

En las figuras 8.18 y 8.19 se pueden ver los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada una de las profundidades fijadas para los nodos solución. En el eje X se representa el tanto por ciento de soluciones en el nivel fijado, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.

En la figura 8.20 se pueden ver los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada uno de los porcentajes de nodos solución fijados para los mismos experimentos. En el eje X se representa la profundidad a la cual se encuentran las soluciones, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica.

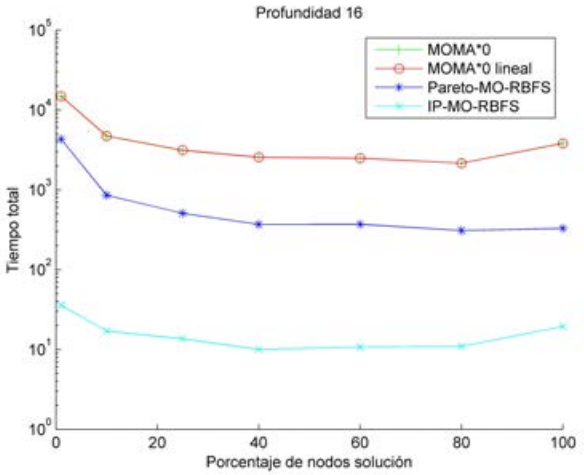
Las figuras 8.18, 8.19 y 8.20 no muestran los experimentos que hayan excedido del límite temporal fijado.

Análisis

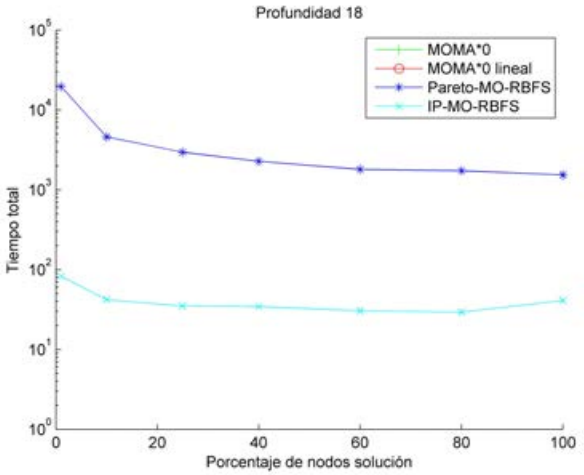
Tal y como se puede apreciar en las gráficas de las figuras 8.18 y 8.19, en las pruebas realizadas los algoritmos $MOMA^*0$ y $MOMA^*0$-*lineal* únicamente fueron capaces de solucionar los problemas en los cuales las soluciones se encontraban a un nivel de profundidad 16. De modo similar, *Pareto-MO-RBFS* únicamente consiguió completar la batería de pruebas hasta el nivel 18.

Al igual que para los análisis realizados en profundización iterativa, el rendimiento de los algoritmos *RBFS* mejora en general a medida que aumentamos la cantidad de soluciones disponibles.

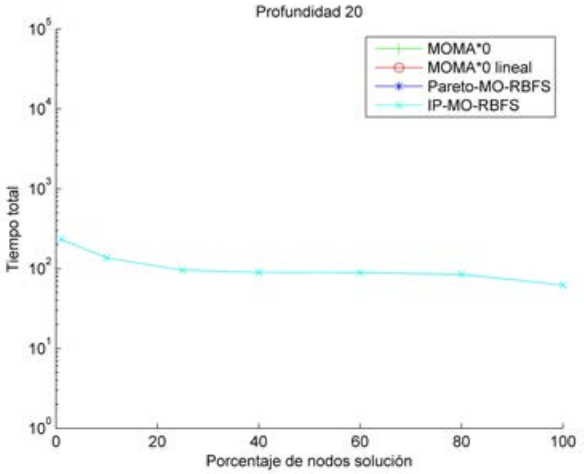
En lo que respecta a los algoritmos multiobjetivo *RBFS* incluídos en esta evaluación, su principal diferencia estriba en la naturaleza de la cota. En el caso de *Pareto-*



(a) Soluciones en nivel 16 del árbol

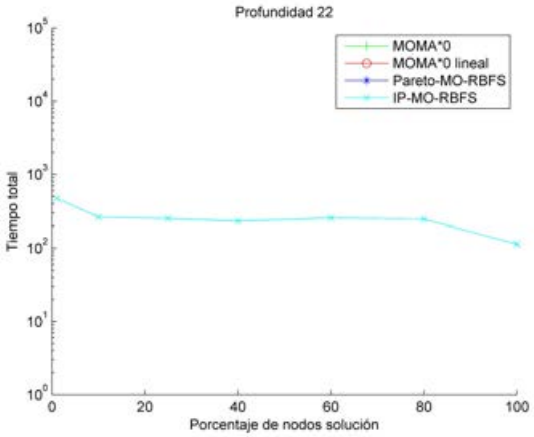


(b) Soluciones en nivel 18 del árbol

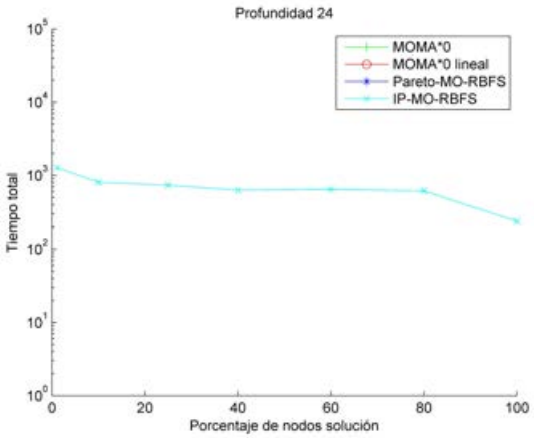


(c) Soluciones en nivel 20 del árbol

Figura 8.18: Comparativa de algoritmos multiobjetivo *RBFS*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para profundidades 16, 18 y 20 de las soluciones con límite temporal.

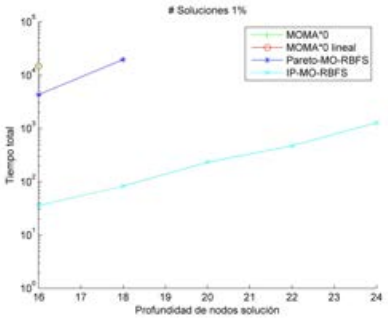


(a) Soluciones en nivel 22 del árbol

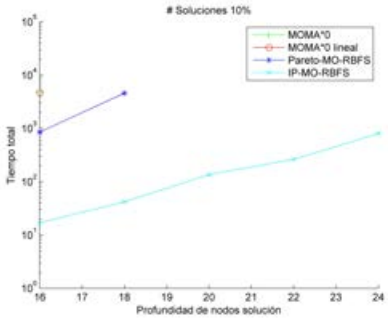


(b) Soluciones en nivel 24 del árbol

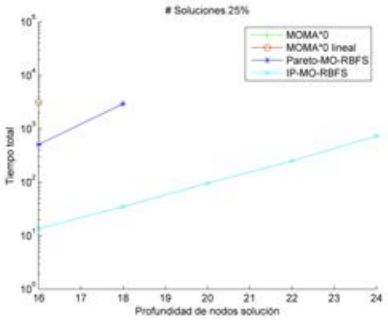
Figura 8.19: Comparativa de algoritmos multiobjetivo *RBFS*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para profundidades 22 y 24 de las soluciones con límite temporal.



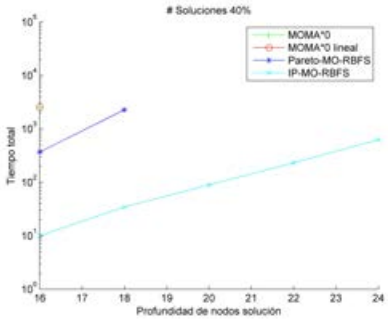
(a) 1 % de soluciones



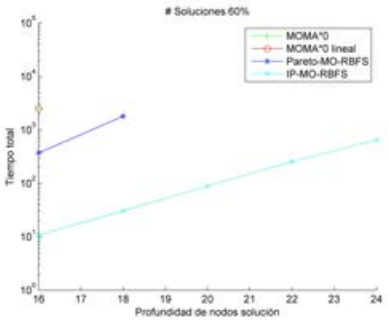
(b) 10 % de soluciones



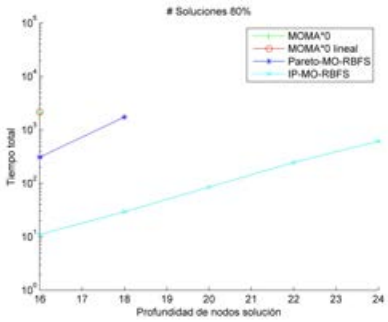
(c) 25 % de soluciones



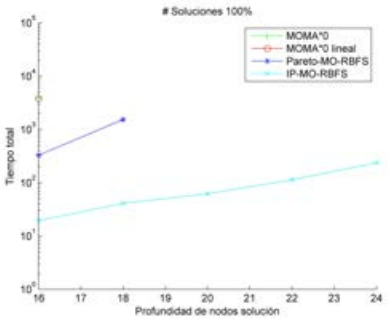
(d) 40 % de soluciones



(e) 60 % de soluciones



(f) 80 % de soluciones



(g) 100 % de soluciones

Figura 8.20: Comparativa de algoritmos multiobjetivo *RBFS*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de la profundidad de las soluciones para distintos porcentajes de soluciones.

MO-RBFS, la cota es multivectorial y almacena los vectores de coste que han sido discontinuados hasta el instante actual de la búsqueda.

Las otras tres opciones ($MOMA^*0$, $MOMA^*0$-*lineal* e *IP-MO-RBFS*) usan un umbral formado por un único vector, el cual se calcula como el mejor lexicográfico, el mejor lineal y el punto ideal respectivamente de la frontera de Pareto. Es precisamente esta cota de expansión, al igual que sucedía con los algoritmos de profundización iterativa (ver sección 8.5) la que determina la eficiencia.

De modo similar a lo que sucedía con $LEXIDMOA^*$, el orden lexicográfico empleado por $MOMA^*0$ hace que de modo general la búsqueda vaya guiada fundamentalmente por uno de los objetivos del problema, lo cual ralentiza la velocidad de expansión de las diferentes ramas. La versión lineal de $MOMA^*0$, $MOMA^*0$-*lineal*, tiene un rendimiento similar, dado que el coste lineal de un nodo (suma de las componentes de su vector de coste) no permite determinar de modo individual la calidad de dicho nodo para cada uno de los objetivos considerados.

Por lo que respecta a *Pareto-MO-RBFS*, si bien su eficiencia es mejor que la de los algoritmos $MOMA^*0$ y $MOMA^*0$-*lineal*, a medida que se incrementa la profundidad del árbol considerado, la frontera de Pareto que debe mantener aumenta considerablemente, lo que acarrea una importante degradación de su rendimiento. Aunque ninguno de los tres algoritmos anteriores ha finalizado problemas de profundidad superior al nivel 18, es de esperar que la pendiente de rendimiento de $MOMA^*0$ y $MOMA^*0$-*lineal* no sea tan acusada como la de *Pareto-MO-RBFS*, al mantener los primeros un umbral de tamaño constante y usar *Pareto-MO-RBFS* un umbral proporcional al tamaño del árbol de búsqueda. Por este motivo, nuestra hipótesis es que, al igual que en los algoritmos multiobjetivo de profundización iterativa, a profundidades mayores *Pareto-MO-RBFS* será el algoritmo más ineficiente.

Finalmente, el rendimiento de *IP-MO-RBFS* resulta ser el mejor de las opciones estudiadas. Al igual que sucedía con $IPID^*$, su contrapartida en profundización iterativa, *IP-MO-RBFS* mantiene una cota de expansión constante y formada por un único vector. Además este vector almacena información asociada a la calidad de todos los objetivos en la frontera de Pareto, lo cual facilita una mayor profundización cada vez que se expande un nodo, recuperando información de más sucesores, lo cual permite ajustar la cota de un modo más eficiente. Además, el hecho de que incluya temporalmente soluciones subóptimas no influye notoriamente en su rendimiento.

Al igual que $IPID^*$, su rendimiento a medida que aumenta la profundidad del problema considerado no conlleva un empeoramiento muy acusado. La pendiente de rendimiento, como se puede observar en las gráficas de la figura 8.20 se mantiene prácticamente constante. Es por todo ello que *IP-MO-RBFS* se presenta como la mejor opción dentro de las variantes *RBFS*.

8.7. Comparativa de variantes multiobjetivo de *DF-BnB*

En esta sección realizaremos una evaluación de las diferentes variantes de *DF-BnB* multiobjetivo en espacios de estados infinitos. Dado que, como ya se comentó en el Capítulo 6, *MO-DF-BnB* no es en general una opción viable en este tipo de problemas, ya que no puede asegurar su completitud, es necesario suministrar de algún modo a este algoritmo una cota inicial, en forma de solución (óptima o no), que le sirva a *MO-DF-BnB* como cota superior de expansión del árbol de búsqueda.

En la sección 8.7.1 se muestra una comparativa de *MO-DF-BnB* y un algoritmo de profundización iterativa, $PIDMOA^*$, sobre árboles finitos. La sección 8.7.2 incluye una comparativa, también sobre árboles finitos, de *MO-DF-BnB* y de variantes del mismo con diferentes cotas. Estas cotas no son necesarias para este tipo de problemas y, como se verá en el análisis de los resultados, su cálculo no aporta mejora de rendimiento sobre el algoritmo básico.

Por último, en la sección 8.7.3 se muestra el impacto que tiene la cota inicial sobre el rendimiento de *MO-DF-BnB* sobre árboles infinitos. Para la obtención de esta cota se han usado diferentes algoritmos de búsqueda escalar, así como algoritmos de búsqueda multiobjetivo adaptados para que devuelvan la primera solución que localicen.

8.7.1. *DF-BnB* multiobjetivo en árboles finitos

En esta sección compararemos *MO-DF-BnB* y $PIDMOA^*$, un algoritmo con un rendimiento sensiblemente inferior al de otras variantes de profundización iterativa, tal y como se vio en la sección 8.5. Se analizará el comportamiento de ambos algoritmos, principalmente en función de la cantidad de soluciones finales (óptimos o no de Pareto).

Experimentos

La batería de problemas generada para esta comparativa entre *MO-DF-BnB* y $PIDMOA^*$ está formada por una serie de árboles **finitos** aleatorios (ver sección 8.3.1) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad del árbol: nivel 23 (nodo raíz en nivel 0).
- Profundidad a la que se encuentran las soluciones: niveles 9, 10, 11, 12, 13, 14 y 15.
- Rango de costes de ambos objetivos: [1,10].
- Correlación cero entre los objetivos.

- Pruebas con búsqueda ciega ($\vec{h}(n) = \vec{0}, \quad \forall n$) y heurística (usando el heurístico descrito en 8.3.1). Se generan un total de 20 árboles aleatorios, 10 para la búsqueda ciega y 10 para la búsqueda heurística.
- Cantidad de nodos solución: 1, 6, 11, 16, 21, 26, 31, 36. Estas cantidades son absolutas, no porcentuales. Además las soluciones no tienen porque ser todas óptimos de Pareto.
- Para cada combinación (profundidad solución, cantidad nodos solución) se genera un total de 10 conjuntos aleatorios de nodos finales (se presentan las medias).

Resultados

En las gráficas de la figura 8.21 se representa la cantidad de nodos expandida por cada uno de los algoritmos, promediada para los 100 problemas de cada tipo generados (10 problemas para búsqueda ciega/heurística combinados cada uno de ellos con 10 conjuntos diferentes de soluciones). Los problemas de búsqueda ciega y heurística se han evaluado de modo separado. Se muestran las gráficas asociadas a profundidades 9, 11, 13 y 15 para los nodos solución. En el eje X se representa la cantidad de nodos finales del problema, mientras que en el eje Y se representa la cantidad de nodos expandida por cada algoritmo. Se ha empleado escala logarítmica para el eje Y.

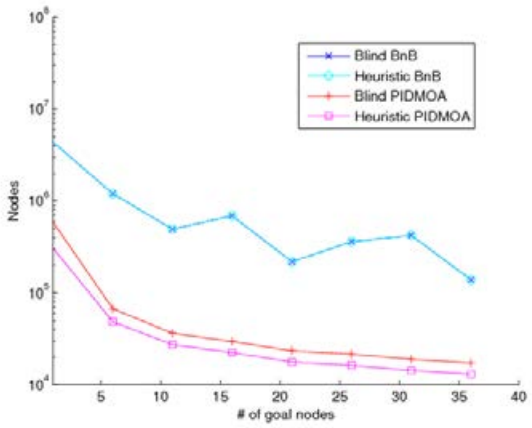
En las gráficas de la figura 8.22 se representa el tiempo de procesamiento (segundos) por cada uno de los algoritmos, promediado para los 100 problemas de cada tipo generados . Se muestran las gráficas asociadas a profundidades 9, 11, 13 y 15 para los nodos solución. En el eje X se representa la cantidad de nodos finales del problema, mientras que en el eje Y se representa el tiempo total de procesamiento para cada algoritmo. Se ha empleado escala logarítmica para el eje Y.

En las gráficas de la figura 8.23 se representa el tiempo medio de procesamiento por nodo expandido para los mismos casos de las figuras 8.21 y 8.22. En el eje X se representa la cantidad de nodos finales del problema, mientras que en el eje Y se representa el tiempo medio de procesamiento de un nodo en segundos (escala logarítmica).

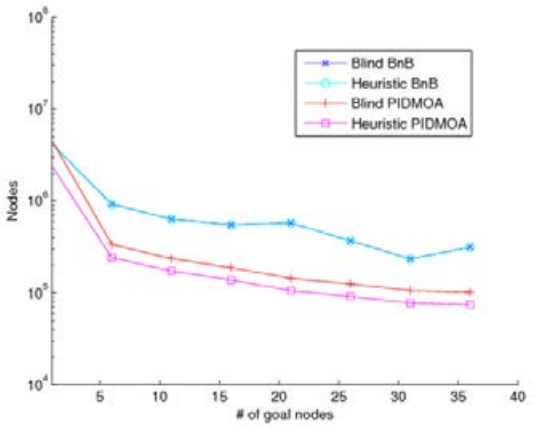
Análisis

Podemos apreciar que el comportamiento de los algoritmos en función de la cantidad de nodos finales es muy similar. En ambos casos, a mayor número de soluciones, mayor probabilidad de encontrar dichas soluciones (incluso óptimos de Pareto) en las primeras iteraciones ($PIDMOA^*$) ó ramas exploradas (*MO-DF-BnB*). Debido a ello el número de podas se incrementa y el número de nodos procesados disminuye considerablemente.

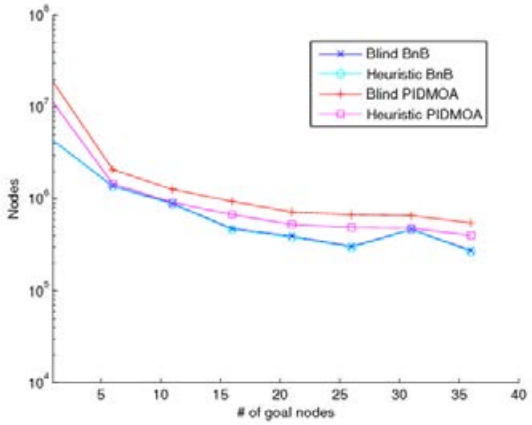
Tal y como podemos apreciar en las gráficas de la figura 8.21, $PIDMOA^*$ expande muchos menos nodos que *MO-DF-BnB* en los problemas en los cuales los nodos solución son más superficiales. Esto se debe a que el coste asociado a la reexpansión durante las diferentes iteraciones no es significativa, al producirse en niveles del árbol que no



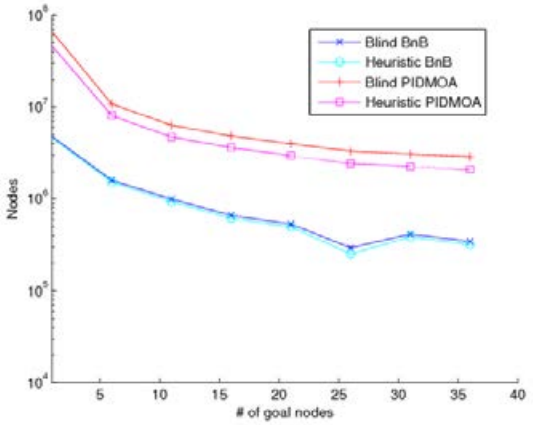
(a) Profundidad soluciones 9



(b) Profundidad soluciones 11

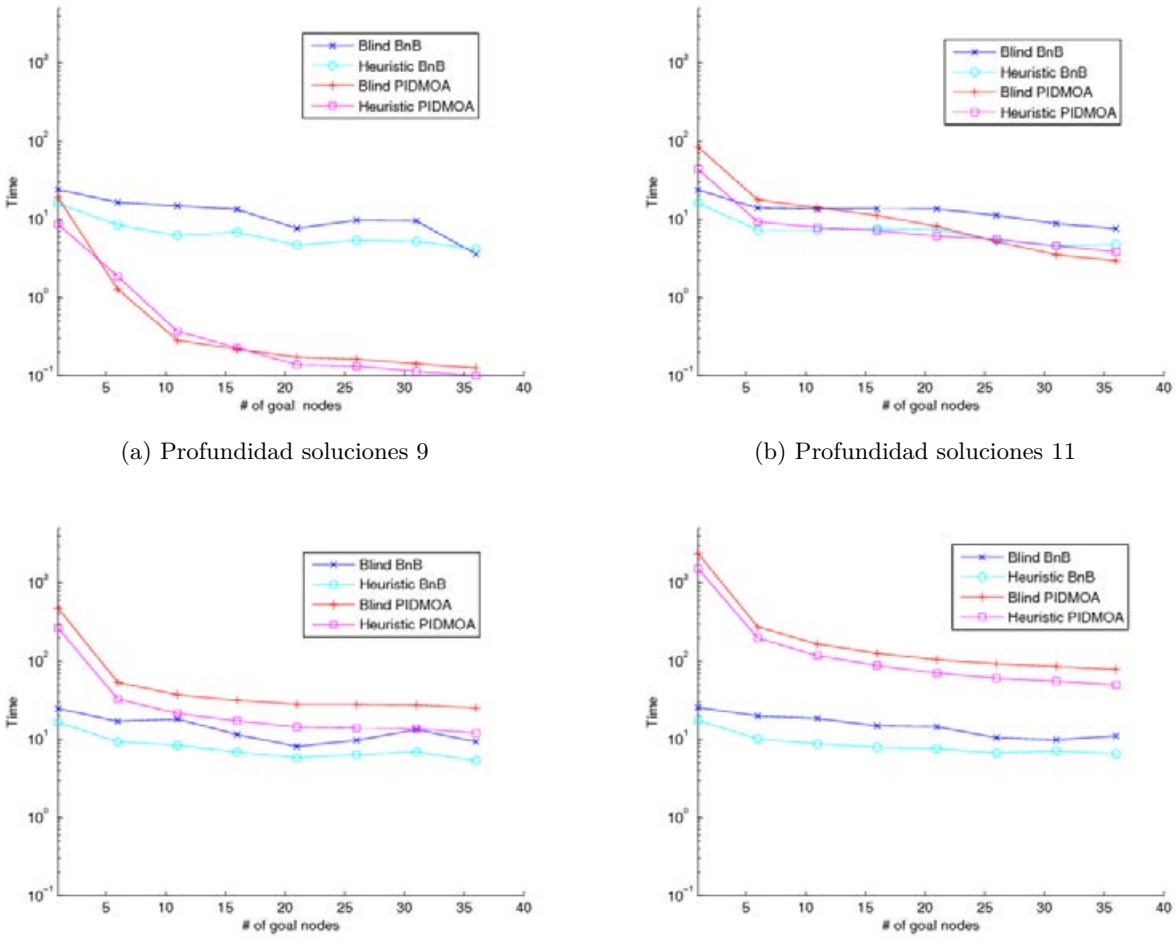


(c) Profundidad soluciones 13



(d) Profundidad soluciones 15

Figura 8.21: Comparativa PIDMOA* vs. DF-MO-BnB. Árboles finitos binarios. Nodos expandidos.



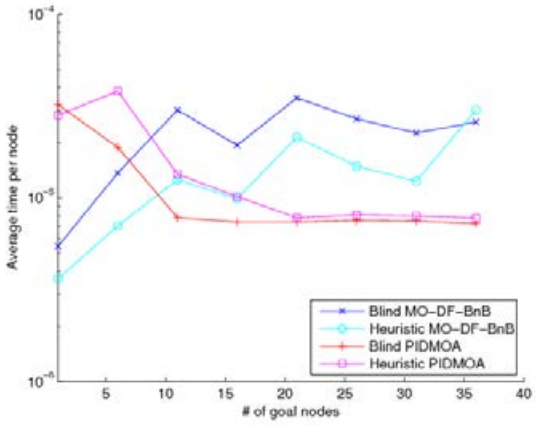
(a) Profundidad soluciones 9

(b) Profundidad soluciones 11

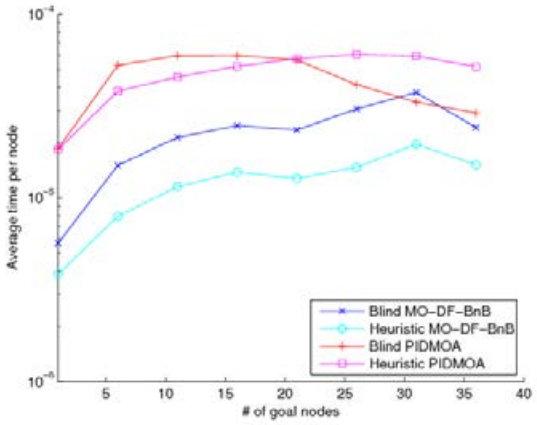
(c) Profundidad soluciones 13

(d) Profundidad soluciones 15

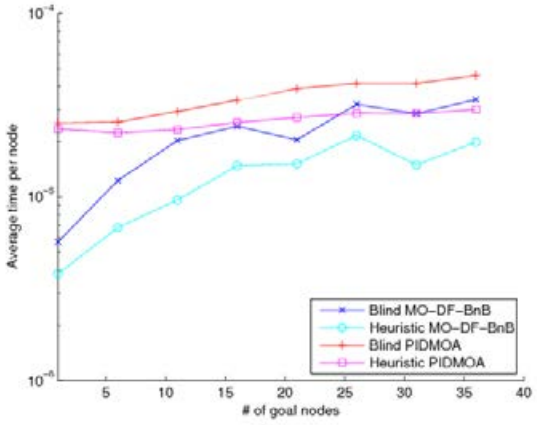
Figura 8.22: Comparativa PIDMOA* vs. DF-MO-BnB. Árboles finitos binarios. Tiempo.



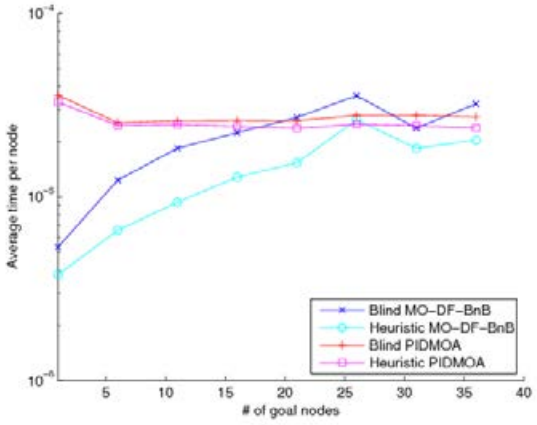
(a) Profundidad soluciones 9



(b) Profundidad soluciones 11



(c) Profundidad soluciones 13



(d) Profundidad soluciones 15

Figura 8.23: Comparativa $PIDMOA^*$ vs. *MO-DF-BnB*. Árboles finitos binarios. Tiempo medio de procesamiento por nodo.

tienen un número de nodos muy elevado. Por el contrario, *MO-DF-BnB* se puede ver obligado a explorar ramas completas del árbol, llegando incluso hasta nodos hoja, sobre todo durante la búsqueda de la cota/solución inicial. Esto es más acusado cuando hay un número de nodos solución bajo, pues es más improbable para *MO-DF-BnB* el encontrar dicha solución inicial pronto. Sin embargo, a medida que aumentamos la profundidad a la cual se encuentran ubicados los nodos solución, *MO-DF-BnB* no incrementa sustancialmente la cantidad de nodos que expande, mientras que sí lo hace $PIDMOA^*$, al tener que profundizar iterativamente hasta niveles cada vez más alejados del nodo raíz, y por tanto incrementando considerablemente el número de nodos explorados.

Además, aunque la función heurística no tiene una calidad destacable, tampoco se percibe una gran diferencia entre los resultados con búsqueda ciega o con búsqueda heurística para *DF-BnB*. $PIDMOA^*$ parece sacar mejor provecho de dicha información heurística.

De modo complementario, en las gráficas de la figura 8.22 incluímos el tiempo de procesamiento para los mismos casos. Los resultados son análogos a los de nodos expandidos. En ambos algoritmos, a mayor número de nodos solución se producen más podas, con lo cual se expanden menos nodos y se realizan muchos menos tests de dominancia, mejorando los tiempos de procesamiento frente a los casos con menos nodos solución. Además en las gráficas podemos percibir que la ganancia obtenida del uso del heurístico es mínima, probablemente debido a la baja calidad de este heurístico genérico.

Aunque los resultados son similares, las diferencias entre algoritmos se amplían en escala. Por ejemplo, con los nodos solución a profundidad 15, $PIDMOA^*$ expande aproximadamente 10 veces más nodos que *MO-DF-BnB*, pero en cambio es aproximadamente 100 veces más lento. El motivo viene dado por el crecimiento de la frontera de Pareto en cada iteración, y con ello el incremento de los tests de dominancia a realizar sobre dicho umbral, tal y como se vio en la sección 8.5.

Los numerosos vectores que hacen las veces de umbral para $PIDMOA^*$ son los que contribuyen a que la diferencia en el número de nodos entre $PIDMOA^*$ y *MO-DF-BnB* se vea incrementada considerablemente cuando tenemos en cuenta el tiempo de procesamiento.

En las gráficas de la figura 8.23 podemos apreciar que, excepto para algunos casos más sencillos (pocas soluciones finales ubicadas en niveles cercanos a la raíz), el tiempo de procesamiento de un nodo es significativamente superior en $PIDMOA^*$, aunque las variaciones en este algoritmo sean menores.

Las tablas 8.1 y 8.2 muestran una serie de medidas estadísticas relativas al tiempo de procesamiento (en segundos) para árboles con soluciones finales ubicadas a profundidades 9 y 15 respectivamente.

Tabla 8.1: Medidas estadísticas para el tiempo de procesamiento (en segundos) para soluciones a nivel 9

		Cantidad de nodos solución							
		1	6	11	16	21	26	31	36
BnB Ciego	Mínimo	0,03	0,08	0,06	0,11	0,03	0,00	0,03	0,00
	Máximo	32,71	24,62	20,75	22,09	19,06	20,19	19,34	18,16
	Media	23,92	16,45	14,86	13,39	7,65	9,71	9,55	3,60
	Desviación estándar	9,03	7,93	7,45	8,78	8,52	8,95	9,10	6,91
BnB Heurístico	Mínimo	0,05	0,12	0,08	1,36	0,03	0,00	0,03	0,00
	Máximo	25,44	15,05	9,39	11,92	7,86	9,39	9,38	7,36
	Media	15,90	8,46	6,20	6,85	4,66	5,37	5,22	4,18
	Desviación estándar	7,65	3,69	3,13	3,13	2,94	3,28	3,42	2,86
PIDMOA ciego	Mínimo	0,14	0,08	0,05	0,05	0,06	0,05	0,03	0,05
	Máximo	80,29	20,34	0,97	0,61	0,41	0,56	0,30	0,28
	Media	18,89	**1,27**	**0,28**	**0,22**	0,17	0,16	0,14	0,12
	Desviación estándar	14,56	3,30	0,17	0,10	0,08	0,08	0,06	0,05
PIDMOA Heurístico	Mínimo	0,13	0,05	0,03	0,05	0,03	0,03	0,02	0,03
	Máximo	32,46	8,39	2,00	1,68	0,34	0,48	0,25	0,25
	Media	**8,64**	1,85	0,37	0,23	**0,14**	**0,13**	**0,11**	**0,10**
	Desviación estándar	5,50	2,55	0,51	0,30	0,07	0,07	0,05	0,04

Tabla 8.2: Medidas estadísticas para el tiempo de procesamiento (en segundos) para soluciones a nivel 15

		Cantidad de nodos solución							
		1	6	11	16	21	26	31	36
BnB Ciego	Mínimo	2,04	17,13	2,06	0,31	1,98	0,16	0,23	0,12
	Máximo	32,90	24,90	22,25	20,03	19,41	18,55	19,64	19,28
	Media	25,51	19,89	18,44	14,84	14,45	10,48	9,85	11,06
	Desviación estándar	5,35	2,40	2,06	6,41	6,07	7,58	8,12	7,93
BnB Heurístico	Mínimo	6,60	6,71	6,57	6,35	6,49	1,37	1,53	0,13
	Máximo	25,57	16,43	13,17	11,01	10,47	8,35	9,41	8,63
	Media	**17,46**	**10,10**	**8,76**	**7,91**	**7,65**	**6,66**	**7,14**	**6,51**
	Desviación estándar	6,10	3,15	1,54	1,12	0,96	1,46	1,51	1,87
PIDMOA ciego	Mínimo	57,80	42,98	31,93	33,42	30,67	23,04	25,27	29,91
	Máximo	11874,73	1084,57	609,62	445,23	248,35	198,95	166,47	211,23
	Media	2378,98	273,66	165,56	126,63	104,09	92,41	85,75	78,85
	Desviación estándar	2572,98	189,71	99,98	71,43	47,53	39,14	30,66	35,29
PIDMOA Heurístico	Mínimo	33,81	21,97	14,71	16,19	13,26	8,78	10,83	13,17
	Máximo	7267,77	774,83	464,90	345,56	184,32	146,49	119,34	153,29
	Media	1522,03	198,07	117,61	88,05	70,46	60,59	55,53	49,93
	Desviación estándar	1585,54	139,96	77,61	55,95	37,46	30,44	23,96	27,49

La desviación estándar permanece constante para *MO-DF-BnB*, mientras que en $PIDMOA^*$ decrece cuando aumenta el número de nodos solución. Es decir, a mayor complejidad del problema en lo que a soluciones finales se refiere, más estabilidad presenta $PIDMOA^*$, siendo un algoritmo más predecible. Sin embargo, a medida que aumentamos la profundidad de la solución, más aumenta la desviación estándar para $PIDMOA^*$. *MO-DF-BnB* no parece depender de estos factores.

Podemos concluir como resultado de esta comparativa que la elección de un algoritmo depende más de la profundidad a la que se encuentran los nodos solución que de la cantidad de los mismos. La opción *MO-DF-BnB* se muestra como la más idónea con problemas de relativa dificultad (árboles más profundos), mostrando también un comportamiento más predecible que $PIDMOA^*$.

8.7.2. Variantes de *DF-BnB* multiobjetivo sobre árboles finitos

En esta sección se evaluarán *MO-DF-BnB* y diferentes variantes de este algoritmo con cota inicial sobre árboles finitos, aunque esta cota no es imprescindible en *Branch & Bound* para resolver este tipo de problemas.

Experimentos

Para la evaluación de los algoritmos se emplearon una serie de árboles finitos aleatorios (ver sección 8.3.1) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 16, 18, 20 y 22 del árbol de búsqueda, teniendo dicho árbol una profundidad máxima de 24 niveles.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos.
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Porcentaje de la cantidad de nodos solución en la profundidad fijada para los mismos: 100 %. Estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 20 problemas aleatorios diferentes (se presentan las medias).

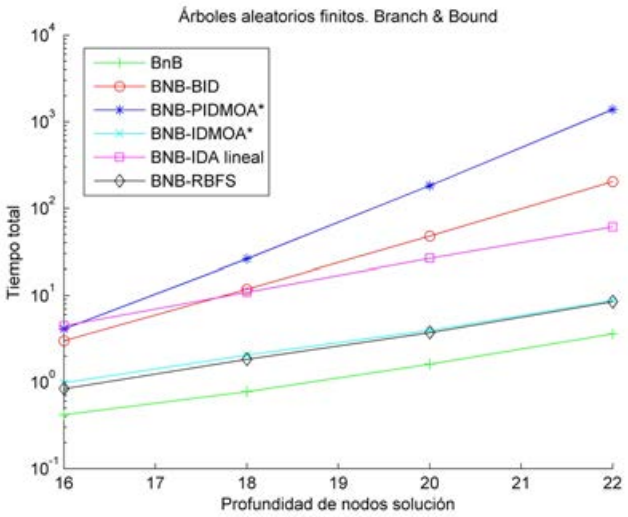


Figura 8.24: Comparativa DF-MO-BnB y variantes con cota. Árboles finitos binarios. Tiempo de procesamiento.

Resultados

En la figura 8.24 se representa el tiempo de procesamiento (segundos) por cada uno de los algoritmos, promediado para los 20 problemas de cada tipo generados . En el eje X se representa la profundidad a la cual se encuentran los nodos finales del problema, mientras que en el eje Y se representa el tiempo total de procesamiento para cada algoritmo. Se ha empleado escala logarítmica para el eje Y.

Análisis

Los algoritmos multiobjetivo empleados en estas pruebas para obtención de cota son variantes de $IDMOA^*$ y $PIDMOA^*$ que devuelven la primera solución encontrada. De los resultados obtenidos se deduce que la obtención de la cota inicial no representa una ventaja en el procesamiento de árboles de búsqueda finitos, pues *MO-DF-BnB* es la opción más eficiente de los algoritmos probados. Los resultados de las variantes con cota son similares a los obtenidos para las pruebas con árboles infinitos de la sección 8.7.3.

De entre las variantes con cota analizadas, las que emplean BID y $PIDMOA^*$ son las menos eficientes. Esto se debe a que al realizar un avance más lento en el árbol de búsqueda, encuentran la cota mucho más tarde que el resto de opciones que trabajan sobre una única magnitud: $RBFS$, IDA^* lineal (que trabaja con una combinación lineal de los objetivos considerados) e $IDMOA^*$ (que al devolver la primera solución que encuentra, únicamente ha procesado el primer objetivo). En el caso de BID, en el procesamiento del árbol por niveles de profundidad se producen tantas reexpansiones como niveles tenga el árbol considerado. Sin embargo, con $PIDMOA^*$, la cantidad de

iteraciones será generalmente mucho mayor, al regirse por todas las componentes de coste de los caminos explorados.

8.7.3. *DF-BnB* multiobjetivo en árboles infinitos

Como ya se ha comentado anteriormente, el algoritmo *MO-DF-BnB* se presenta como una buena alternativa para cierto tipo de árboles infinitos, pero siempre y cuando se disponga de una cota inicial para asegurar su completitud. En nuestro análisis consideraremos pues un algoritmo secuencial:

- En una primera fase utilizaremos un algoritmo completo en grafos infinitos para calcular una solución inicial P.
- En una segunda fase aplicaremos *MO-DF-BnB* usando como cota superior inicial el coste de P.

La opción más lógica para la primera fase es la familia de algoritmos de profundización iterativa. En nuestra evaluación los algoritmos seleccionados para obtener la cota inicial han sido los siguientes:

- BID: Profundización iterativa basada exclusivamente en el nivel de profundidad del árbol. La primera cota es el nivel 0, luego el nivel 1, y así sucesivamente. Para las cotas, por tanto, no se tienen en cuenta los costes de ninguno de los objetivos. El umbral devuelto por este algoritmo a *MO-DF-BnB* será el vector de coste de la primera solución localizada, la cual puede ser óptima o subóptima. Sólo tenemos asegurado que es la menos profunda.
- IDA^* $obj1$: Profundización iterativa para un único objetivo. Se emplea el algoritmo IDA^*, contemplando únicamente el coste asociado a uno de los objetivos del problema La solución obtenida como cota inicial de *MO-DF-BnB* es óptima en cuanto a dicho objetivo, pero podría no ser un óptimo de Pareto. En las pruebas realizadas sobre problemas biobjetivo se ha escogido la primera componente del vector de coste.
- IDA^* lineal: Se emplea nuevamente el algoritmo IDA^* para obtener una cota inicial, pero en lugar de usar el coste de uno de los objetivos, se calcula una combinación lineal de ambos, la cual actúa como coste del camino. En nuestro caso esta combinación lineal viene dada por $\frac{c_1+c_2}{2}$, donde c_1 y c_2 son las componentes del vector de coste. En este caso, la primera solución encontrada será un óptimo de Pareto.
- *RBFS*: Se emplea el algoritmo *RBFS*, contemplando únicamente el coste asociado al objetivo del problema asociado a la primera componente de coste. Esto, al igual que en el caso de IDA^*, no permite asegurar una solución óptimo de Pareto.

- $IPID^*$: Se emplea una variante de $IPID^*$ que devuelve la primera solución encontrada. Dado que $IPID^*$ puede localizar soluciones subóptimas, no tenemos garantizado que la cota inicial localizada por el algoritmo sea una solución óptima.

La selección incluye fundamentalmente algoritmos para un solo objetivo, ya que no precisan realizar operaciones vectoriales, las cuales degradan considerablemente el rendimiento. Se incluye un algoritmo que usa una combinación lineal de costes y, a efectos de completitud de la comparativa, un algoritmo multiobjetivo, el cual ha sido convenientemente modificado para que únicamente devuelva la primera solución que localice. También se ha jugado con la calidad de las soluciones devueltas, ya que se incluyen tanto algoritmos que devuelven una cota que es óptimo de Pareto como algoritmos que podrían devolver soluciones subóptimas. Los algoritmos multiobjetivo han sido modificados para que finalicen una vez han localizado la primera solución (p.ej. en la tabla 6.2 podemos ver la modificación realizada a $IPID^*$).

Experimentos

La batería de problemas generada para este análisis de *MO-DF-BnB* con diferentes cotas iniciales está formada por una serie de árboles infinitos aleatorios (ver sección 8.3.2) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 16, 18, 20 y 22 del árbol de búsqueda.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos.
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Porcentaje de la cantidad de nodos solución en la profundidad fijada para las soluciones. Estos porcentajes son 1 %, 10 %, 25 %, 40 %, 60 %, 80 % y 100 %. Este último caso representa un árbol finito. Tal y como comentamos en la sección 8.3, estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 10 problemas aleatorios diferentes (se presentan las medias).

Los experimentos son similares a los usados para la evaluación de algoritmos multiobjetivo basados en profundización iterativa (secciones 8.5.2 y 8.5.3).

Resultados

En las gráficas de las figuras 8.25 y 8.26 se representan los tiempos de procesamiento en segundos en el eje Y con escala logarítmica, agrupados por profundidad de las soluciones para el algoritmo *MO-DF-BnB* combinado con las diferentes opciones para la obtención de cota inicial. Se representa para cada profundidad tanto el tiempo requerido para obtener la cota inicial como el tiempo total de resolución. El eje X representan los diferentes porcentajes de soluciones finales en el nivel fijado.

En las gráficas de la figura 8.27 se representan los tiempos totales de procesamiento en segundos en el eje Y con escala logarítmica, agrupados por profundidad de las soluciones para el algoritmo *MO-DF-BnB* combinado con las diferentes opciones para la obtención de cota inicial. El eje X representa los diferentes porcentajes de soluciones finales en el nivel fijado.

En las gráficas de la figura 8.28 se representan los tiempos totales de procesamiento en segundos en el eje Y con escala logarítmica, agrupados por porcentaje de soluciones a la profundidad establecida para el algoritmo *MO-DF-BnB* combinado con las diferentes opciones para la obtención de cota inicial. El eje X representa los diferentes niveles de profundidad fijados para los nodos solución.

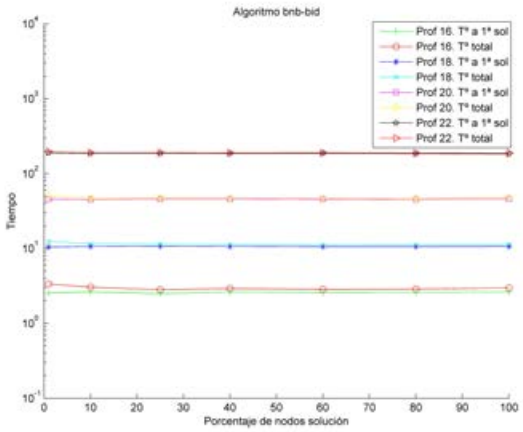
Análisis

Como se puede apreciar claramente en las gráficas de las figuras 8.25 y 8.26, el tiempo de procesamiento viene dado casi exclusivamente por la búsqueda de la primera solución. El tiempo restante empleado por *MO-DF-BnB* para localizar todas las soluciones óptimas, una vez obtenida la cota inicial, es residual, independientemente del algoritmo empleado para obtener dicha cota inicial o de la dificultad del problema en cuanto a profundidad o cantidad de soluciones.

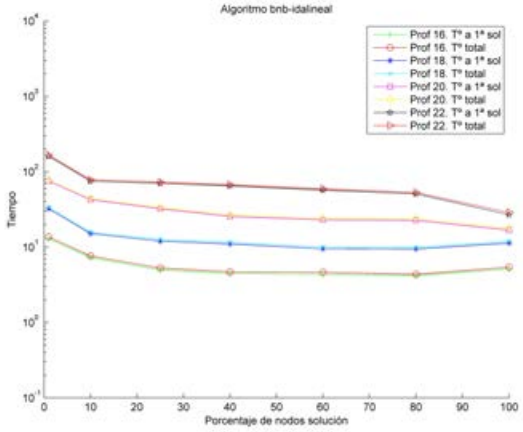
De modo similar a lo ya visto para las variantes de profundización iterativa y *RBFS*, en general el comportamiento de los algoritmos basados en *DF-BnB* mejora a medida que se aumenta la cantidad de soluciones del problema. La única excepción es el algoritmo BID, el cual, al realizar una expansión del árbol basada exclusivamente en la profundidad de los nodos, tiene que explorar el árbol de búsqueda hasta la profundidad dada, independientemente de la cantidad de soluciones que haya disponibles.

Como cabe suponer, los algoritmos más rápidos para conseguir la cota inicial son aquellos con un enfoque de un objetivo, en este caso IDA^* y *RBFS*, tal y como puede apreciarse en las gráficas de las figuras 8.27 y 8.28. El algoritmo $IPID^*$ presenta un rendimiento ligeramente inferior al de los anteriormente mencionados, debido fundamentalmente a la necesidad de realizar los tests de dominancia vectoriales de los cuales están exentos los algoritmos *escalares*.

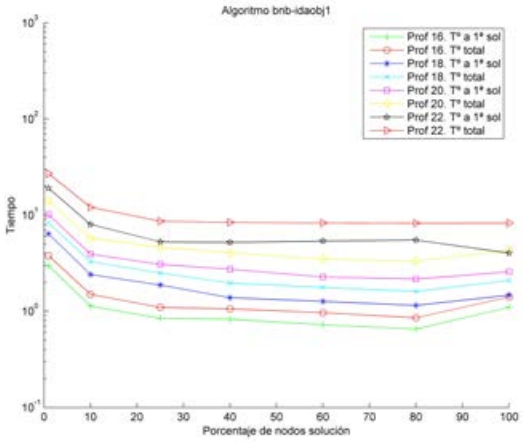
El algoritmo que presenta peor rendimiento para localizar esta primera solución es BID, ya que al realizar la profundización iterativa basándose no en costes, sino en la profundidad del árbol, su avance se realiza de modo más lento, incrementando por lo tanto el número de reexpansiones. IDA^* *lineal*, aunque en la práctica realiza un



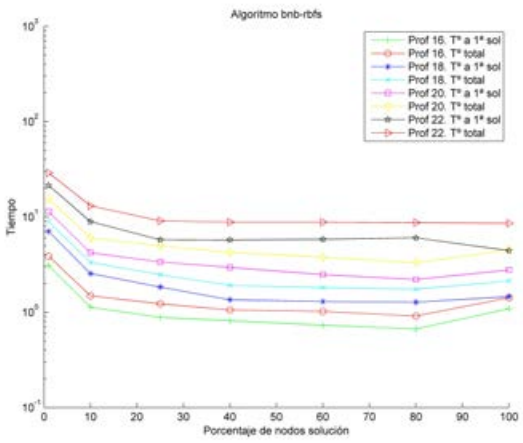
(a) Cota inicial con BID



(b) Cota inicial con IDA^* lineal

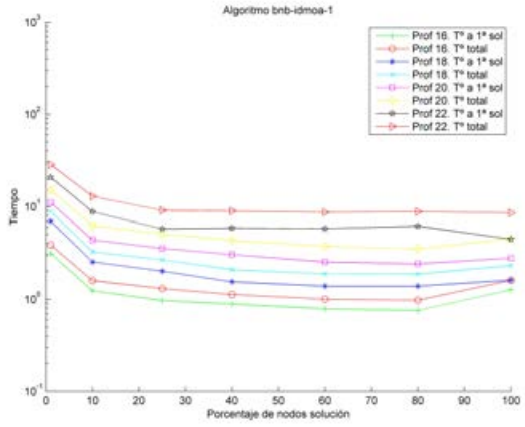


(c) Cota inicial con IDA^* Obj. 1

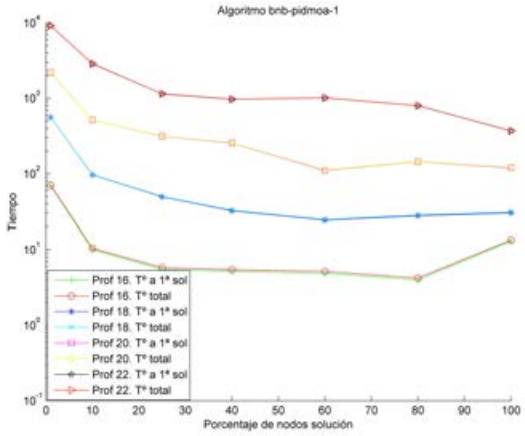


(d) Cota inicial con $RBFS$

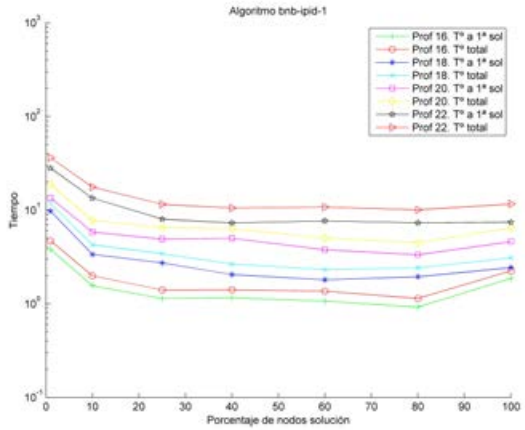
Figura 8.25: Comparativa de algoritmos multiobjetivo *DF-BnB* con cota inicial (BID, IDA^* lineal, IDA^* con objetivo 1, $RBFS$). Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de distintos porcentajes de soluciones para diferentes algoritmos de cálculo de cotas iniciales.



(a) Cota inicial con $IDMOA^*$

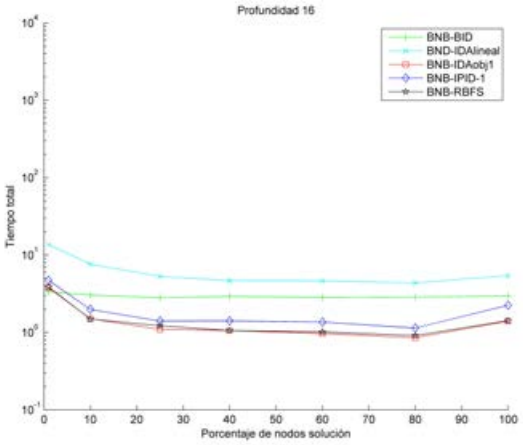


(b) Cota inicial con $PIDMOA^*$

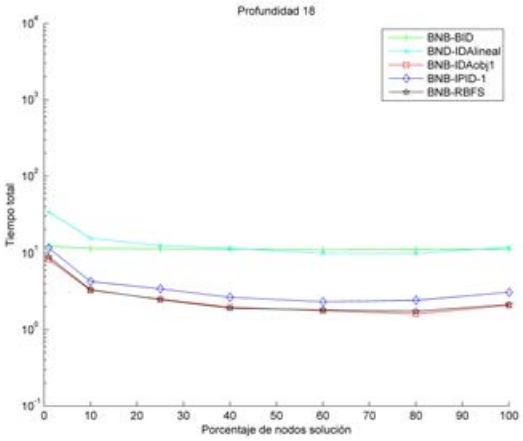


(c) Cota inicial con $IPID^*$

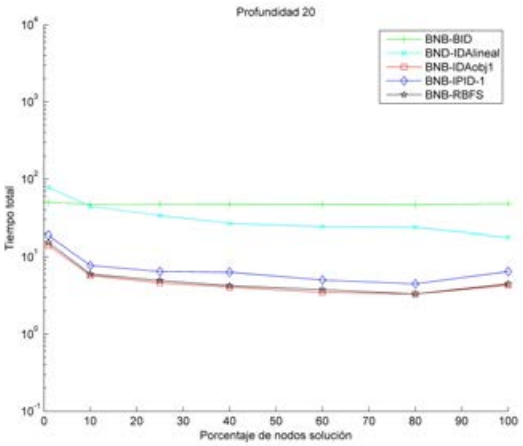
Figura 8.26: Comparativa de algoritmos multiobjetivo *DF-BnB* con cota inicial ($IDMOA^*$, $PIDMOA^*$, $IPID^*$). Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de distintos porcentajes de soluciones para diferentes algoritmos de cálculo de cotas iniciales.



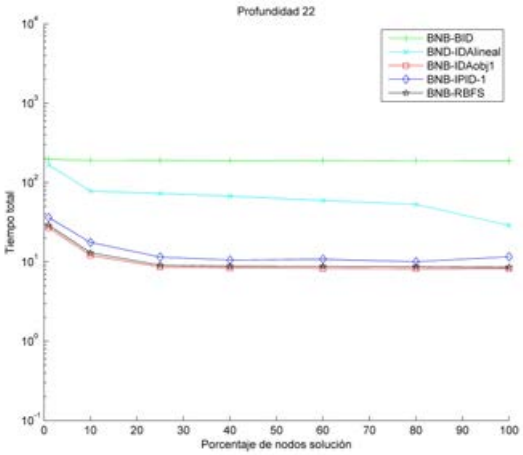
(a) Soluciones a profundidad 16



(b) Soluciones a profundidad 18

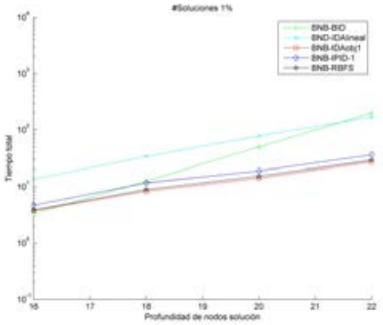


(c) Soluciones a profundidad 20

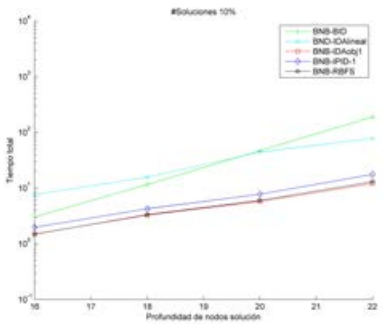


(d) Soluciones a profundidad 22

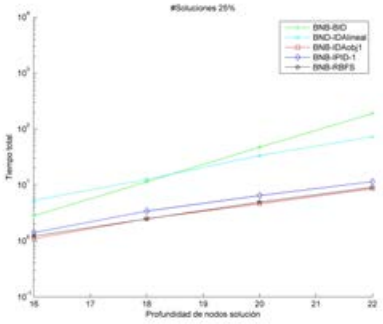
Figura 8.27: Comparativa de algoritmos multiobjetivo *DF-BnB*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de distintos porcentajes de soluciones para distintas profundidades de dichos nodos solución.



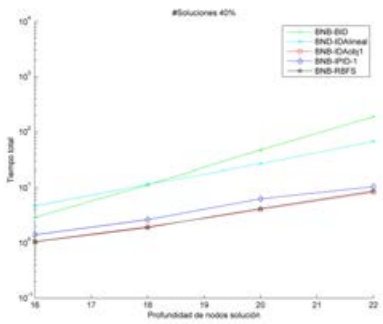
(a) 1 % de soluciones



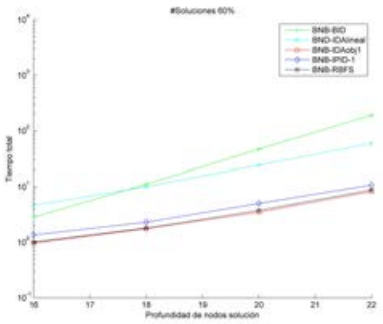
(b) 10 % de soluciones



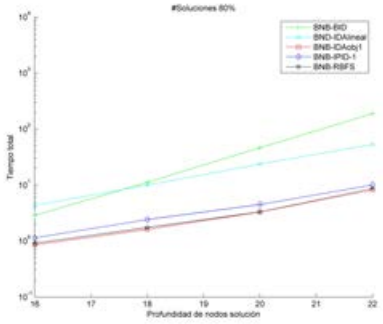
(c) 25 % de soluciones



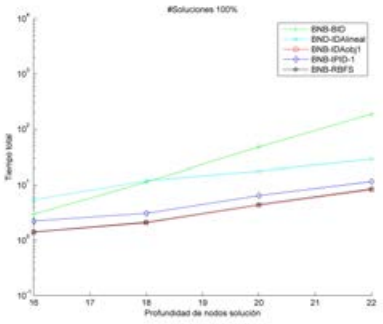
(d) 40 % de soluciones



(e) 60 % de soluciones



(f) 80 % de soluciones



(g) 100 % de soluciones

Figura 8.28: Comparativa de algoritmos multiobjetivo *DF-BnB*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de distintas profundidades de los nodos solución para diferentes porcentajes de soluciones.

procesamiento escalar sobre un objetivo *ficticio*, que es la combinación lineal de las componentes del vector de coste, ve penalizado su rendimiento como consecuencia de estos cálculos adicionales.

Las diferencias entre algoritmos son más acusadas a medida que crece la profundidad a la cual se encuentran las soluciones del problema, y por lo tanto su dificultad.

La evaluación realizada nos permite comprobar que la calidad de la cota inicial no afecta significativamente al tiempo de procesamiento. Como ya se ha comentado, para ninguna de las combinaciones hay una gran diferencia entre el tiempo a la primera solución y el tiempo total de procesamiento. Por ejemplo, el hecho de que IDA^* *lineal* proporcione un óptimo de Pareto como cota inicial no parece influir en una localización más rápida del conjunto de soluciones buscado. Por ello se antoja como algoritmo auxiliar idóneo aquel que localice lo antes posible una cota para *MO-DF-BnB*. Según los resultados obtenidos, esta mejor opción sería el uso de IDA^* ó *RBFS* sobre uno de los objetivos considerados en el problema, ya que estos dos algoritmos presentan un rendimiento similar sobre los problemas resueltos. En la práctica, la elección de uno u otro puede venir dada por la velocidad de expansión de los nodos (Hatem et al., 2015).

8.8. Comparativa general de algoritmos *depth-first*

En las secciones previas hemos realizado diversas evaluaciones para cada una de las tres categorías de algoritmos multiobjetivo exactos *depth-first* identificados en este trabajo: basados en profundización iterativa, basados en *RBFS* y basados en *DF-BnB*. Como resultado de estas comparativas, los mejores candidatos de cada categoría en la resolución de árboles infinitos han resultado ser $IPID^*$, *IP-MO-RBFS* y *MO-DF-BnB* con cota inicial basada en el algoritmo escalar IDA^* (o *RBFS*) para uno cualquiera de los objetivos contemplados.

El uso del *punto ideal* se ha revelado un buen esquema para los algoritmos basados en cotas de expansión, al mantener un umbral de expansión sencillo, formado por un único vector, y al mismo tiempo almacenar información relativa a los caminos discontinuados teniendo en cuenta todos los objetivos del problema. Por lo que respecta a la variante *DF-BnB*, el rendimiento del algoritmo en grafos infinitos viene marcado por el del algoritmo auxiliar empleado para obtener una cota inicial. Según la evaluación realizada en la sección 8.7, cuanto más sencillo sea este algoritmo auxiliar, mayor rendimiento presentará *MO-DF-BnB*, independientemente de la calidad de dicha cota inicial. Es por ello que en esta sección seleccionaremos IDA^* para uno de los objetivos, un algoritmo escalar de profundización iterativa, que asegura la completitud.

En el caso de árboles finitos, la única diferencia es la elección de *MO-DF-BnB* como representante de la categoría *DF-BnB*, pues con este tipo de problemas este algoritmo no precisa cota y ha presentado mejor rendimiento que las variantes que sí calculaban una cota inicial.

La finalidad de las pruebas aquí presentadas es determinar el algoritmo multiobje-

tivo exacto *depth-first* con mejores prestaciones para el caso general.

8.8.1. Comparativa IDA^*+***MO-DF-BnB***, $IPID^*$ e IP-MO-$RBFS$ sobre árboles infinitos

Como ya hemos comentado, en esta sección recopilaremos los resultados de las diferentes pruebas realizadas para los mejores representantes de cada una de las categorías de algoritmos establecidas sobre un conjunto de árboles infinitos.

Experimentos

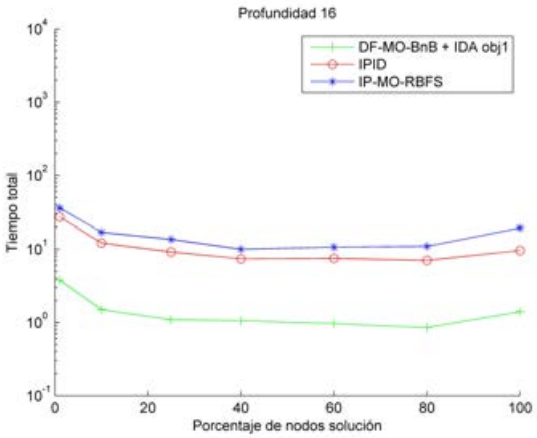
Para la evaluación de los algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS* se emplearon una serie de árboles infinitos aleatorios (ver sección 8.3.2) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 16, 18, 20, 22 y 24 del árbol de búsqueda.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos.
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Porcentaje de la cantidad de nodos en la profundidad fijada para las soluciones. Estos porcentajes son 1 %, 10 %, 25 %, 40 %, 60 %, 80 % y 100 %. Este último caso representa un árbol finito. Tal y como comentamos en la sección 8.3, estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 10 problemas aleatorios diferentes (se presentan las medias).

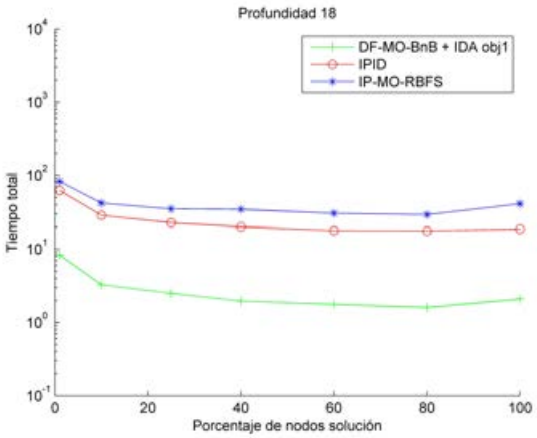
Estos árboles son similares a los empleados para la evaluación de algoritmos multiobjetivo basados en profundización iterativa (sección 8.5) y basados en *RBFS* (sección 8.6).

Resultados

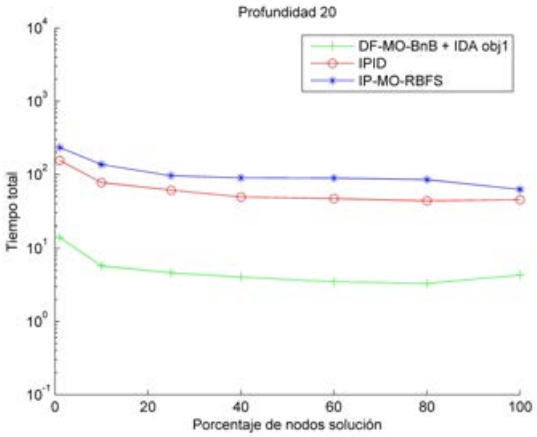
En la figura 8.29 se pueden ver los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada una de las profundidades fijadas para los nodos solución. En el eje X se representa el tanto por ciento de soluciones en el nivel fijado, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.



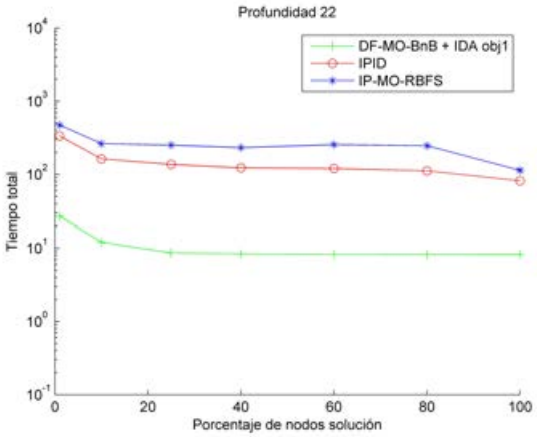
(a) Soluciones en nivel 16 del árbol



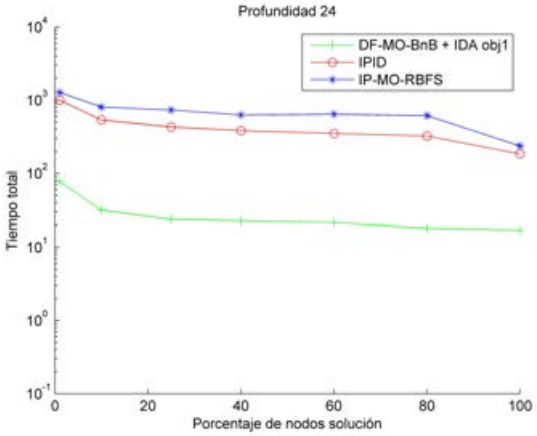
(b) Soluciones en nivel 18 del árbol



(c) Soluciones en nivel 20 del árbol



(d) Soluciones en nivel 22 del árbol



(e) Soluciones en nivel 24 del árbol

Figura 8.29: Comparativa de algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para distintos niveles de profundidad de las soluciones.

En la figura 8.30 se pueden ver los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada uno de los porcentajes de nodos solución fijados. En el eje X se representa la profundidad a la cual se encuentran las soluciones, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.

Análisis

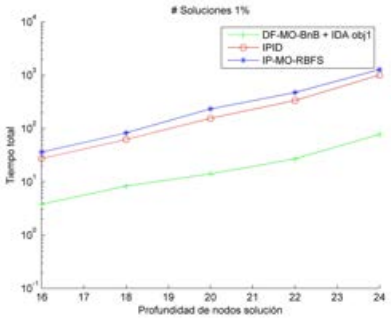
Como se puede apreciar en las gráficas de las figuras 8.29 y 8.30, el algoritmo que presenta claramente un mejor rendimiento de los tres es la variante secuencial IDA^* *+MO-DF-BnB* para la cota inicial, con una diferencia de aproximadamente un orden de magnitud con respecto al segundo más eficiente, $IPID^*$, incrementándose esta diferencia ligeramente a medida que aumentamos la profundidad del árbol.

Como ya vimos en la sección 8.7, el tiempo de computación asociado a IDA^* *+MO-DF-BnB* viene dado mayoritariamente por la búsqueda de la primera solución por parte del algoritmo escalar. Una vez obtenida esta cota, una solución del problema que puede incluso ser subóptima, el algoritmo *MO-DF-BnB* realiza una búsqueda *depth-first* podando con el conjunto de soluciones localizadas en cada instante. Este conjunto, aún cuando es posible que albergue también soluciones subóptimas, permite realizar una poda drástica del espacio de búsqueda, mejorando notablemente la eficiencia del algoritmo. IDA^* *+MO-DF-BnB* únicamente realiza tests de dominancia contra el conjunto de soluciones actuales, tanto cuando usa este conjunto como cota superior para determinar si discontinúa la búsqueda en un nodo como para comprobar, cuando localiza una solución, si ésta domina a alguna de las ya existentes.

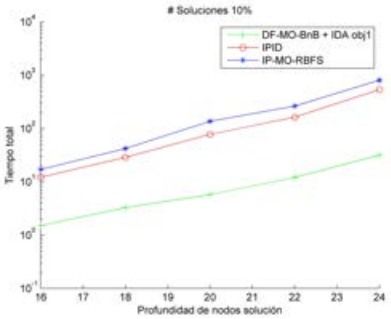
Por el contrario, los algoritmos que emplean una cota calculada como el punto ideal de la frontera de Pareto se ven obligados a realizar más tests. Además de comprobar las nuevas soluciones contra el conjunto de soluciones ya localizadas, cada nodo debe verificar si es estrictamente mejor que la cota fijada. Además también influyen en el rendimiento las reexpansiones de las distintas profundizaciones.

La diferencia que se observa en las gráficas entre $IPID^*$ e $IP\text{-}MO\text{-}RBFS$ radica en la cantidad de cotas que deben calcular cada uno de los algoritmos. $IP\text{-}MO\text{-}RBFS$ expande en general menos nodos que $IPID^*$, al permitir la propagación de información de expansiones previas hacia los nodos sucesores mediante la función *MaxVector*. Sin embargo la cota en este algoritmo debe calcularse por cada nodo procesado. Además esta cota variará para un mismo nodo entre diferentes reexpansiones del mismo, pues el procesamiento realizado entre estas reexpansiones habrá profundizado la frontera de Pareto del árbol de búsqueda.

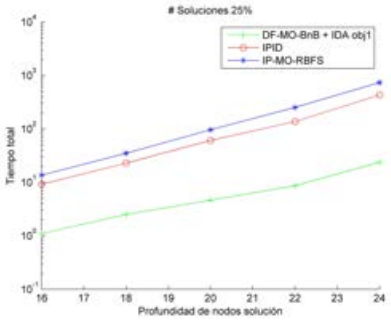
$IPID^*$, por el contrario, calcula una única cota de expansión por iteración, siendo esta cota única para todos los nodos del árbol de búsqueda generado durante dicha iteración. Este ahorro compensa el mayor número de nodos generado por $IPID^*$.



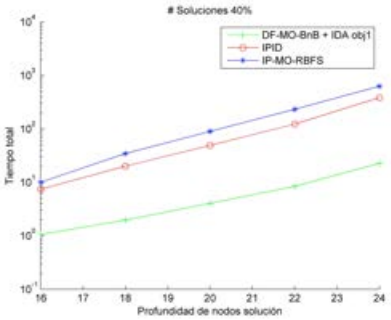
(a) 1 % de soluciones



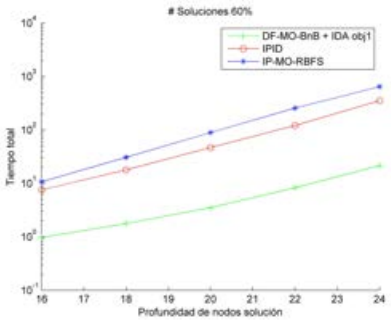
(b) 10 % de soluciones



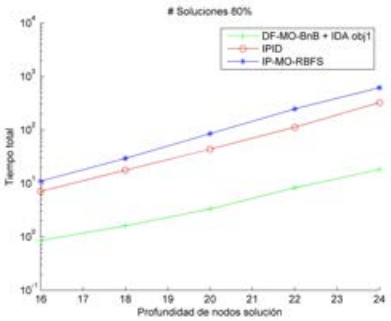
(c) 25 % de soluciones



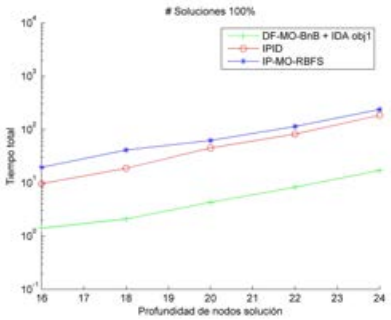
(d) 40 % de soluciones



(e) 60 % de soluciones



(f) 80 % de soluciones



(g) 100 % de soluciones

Figura 8.30: Comparativa de algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS*. Árboles infinitos binarios. Tiempo de procesamiento (segundos) en función de la profundidad de las soluciones para distintos porcentajes de soluciones.

8.8.2. Efecto de la correlación sobre los algoritmos

En la siguiente sección evaluamos el comportamiento de los algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS* en presencia de correlaciones positivas y negativas.

Experimentos

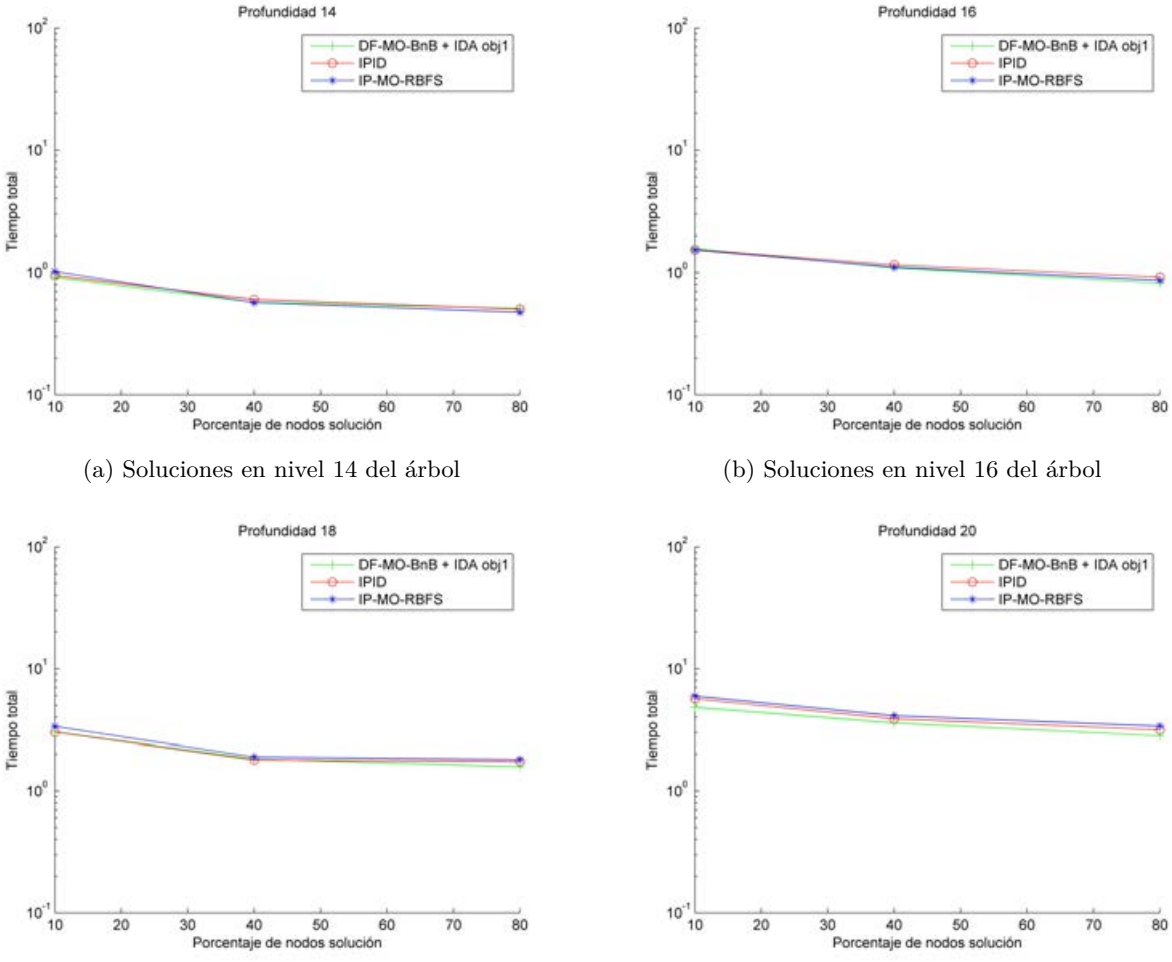
Para la evaluación de los algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS* se emplearon una serie de árboles infinitos aleatorios (ver sección 8.3.2) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.
- Profundidad a la que se encuentran las soluciones: niveles 14, 16, 18, 20 y 20 del árbol de búsqueda.
- Rango de costes de ambos objetivos: [1,50].
- Correlación entre los objetivos: -0.5, 0.5 y 1.
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Es un porcentaje de la cantidad de nodos en la profundidad fijada para las soluciones. Estos porcentajes son 10 %, 40 % y 80 %. Tal y como comentamos en la sección 8.3, estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 10 problemas aleatorios diferentes (se presentan las medias).

Se realizaron pruebas adicionales con correlación -1 con un límite temporal proporcional ($5\times$) al tiempo de procesamiento de la variante *DF-BnB*, pero en ningún caso los algoritmos $IPID^*$ e *IP-MO-RBFS* finalizaron la resolución de los árboles.

Resultados

En las figuras 8.31, 8.32 y 8.33 se pueden ver (para correlaciones 1, 0,5 y $-0{,}5$ respectivamente) los tiempos de procesamiento (medidos en segundos) de los algoritmos para cada una de las profundidades fijadas para los nodos solución. En el eje X se representa el tanto por ciento de soluciones en el nivel fijado, mientras que en el eje Y representamos el tiempo de procesamiento en segundos, empleando escala logarítmica para el eje del tiempo.



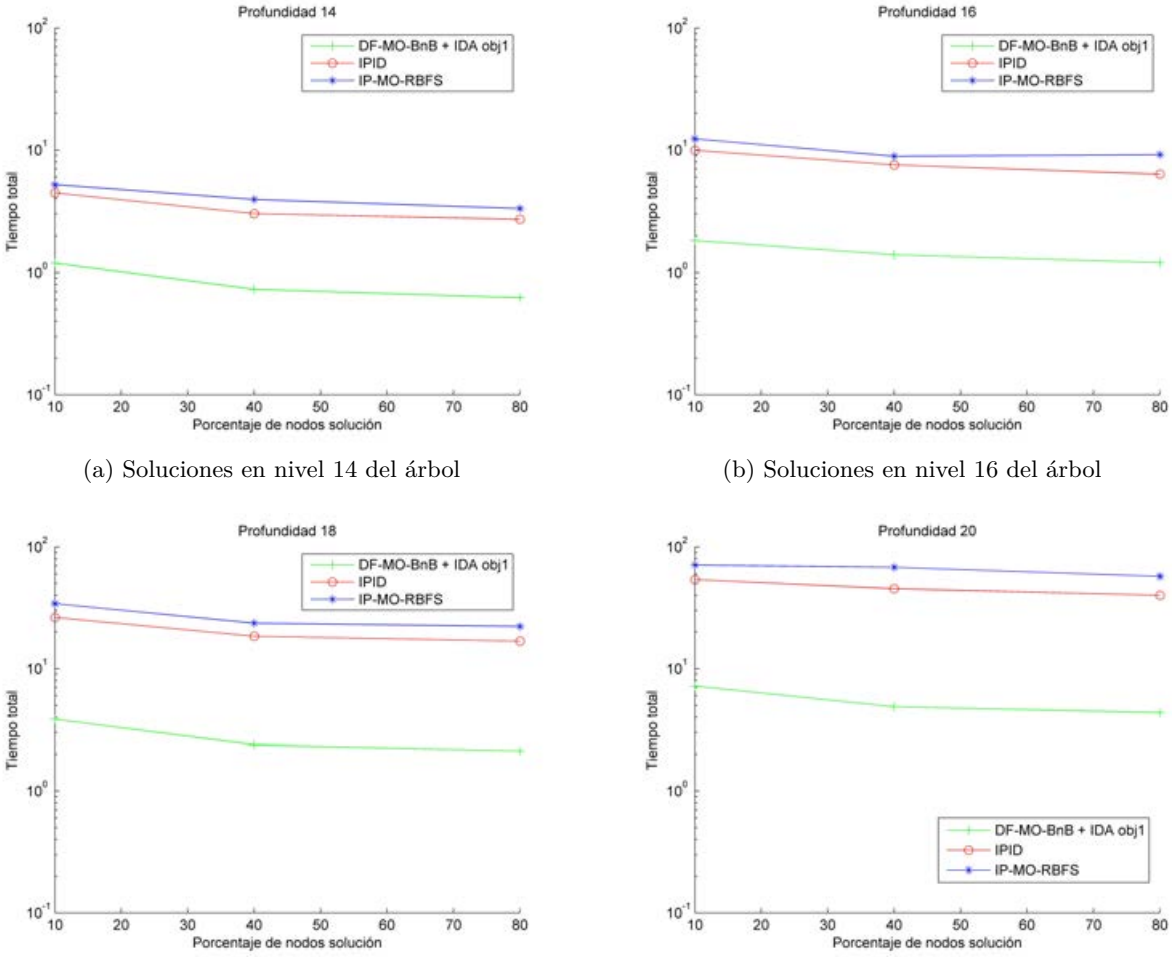
(a) Soluciones en nivel 14 del árbol

(b) Soluciones en nivel 16 del árbol

(c) Soluciones en nivel 18 del árbol

(d) Soluciones en nivel 20 del árbol

Figura 8.31: Comparativa de algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS*. Árboles infinitos binarios. Correlación 1 entre los objetivos. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para distintos niveles de profundidad de las soluciones.



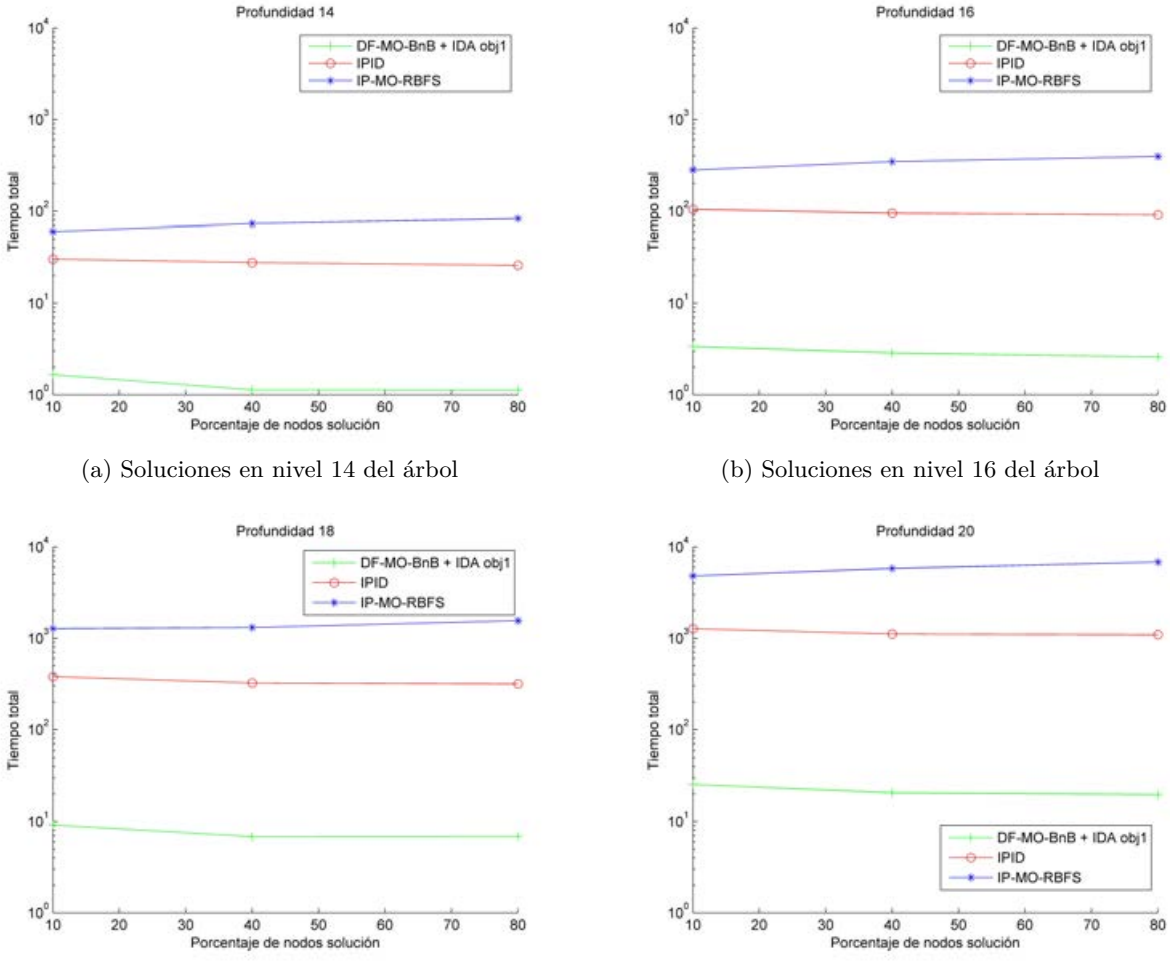
(a) Soluciones en nivel 14 del árbol

(b) Soluciones en nivel 16 del árbol

(c) Soluciones en nivel 18 del árbol

(d) Soluciones en nivel 20 del árbol

Figura 8.32: Comparativa de algoritmos IDA^* $+MO\text{-}DF\text{-}BnB$, $IPID^*$ e $IP\text{-}MO\text{-}RBFS$. Árboles infinitos binarios. Correlación 0.5 entre los objetivos. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para distintos niveles de profundidad de las soluciones.



(a) Soluciones en nivel 14 del árbol

(b) Soluciones en nivel 16 del árbol

(c) Soluciones en nivel 18 del árbol

(d) Soluciones en nivel 20 del árbol

Figura 8.33: Comparativa de algoritmos IDA^* +*MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS*. Árboles infinitos binarios. Correlación -0.5 entre los objetivos. Tiempo de procesamiento (segundos) en función del porcentaje de soluciones para distintos niveles de profundidad de las soluciones.

Análisis

En el caso de correlación 1 entre objetivos, representado en la figura 8.31, la igualdad entre algoritmos es prácticamente absoluta, independientemente de la dificultad del problema. Esto se debe a que esta correlación genera vectores de coste con el mismo valor para todas las componentes. De hecho, todos estos problemas tienen un único óptimo de Pareto. Al reducirse al mínimo la cantidad de vectores de coste no dominados entre sí y haber una única solución óptima, los algoritmos $IPID^*$ e *IP-MO-RBFS* prácticamente igualan el rendimiento conseguido por IDA^* *+MO-DF-BnB* multiobjetivo.

Para una correlación positiva $0'5$, cuyos resultados se presentan en las gráficas de la figura 8.32, los resultados son similares a los obtenidos para correlación 0. Al haber más correlación entre los objetivos, la variabilidad en las componentes de coste es menor, y con ello se reduce la correspondiente frontera de Pareto usada por $IPID^*$ e *IP-MO-RBFS* para calcular su cota de expansión. Esto se traduce en que la diferencia de rendimiento con respecto a IDA^* *+MO-DF-BnB* multiobjetivo sea inferior al orden de magnitud obtenido con correlación 0.

Las gráficas de la figura 8.33 muestran los resultados de ejecutar problemas con correlación negativa, $-0{,}5$, entre los objetivos. Esta correlación negativa tiene el efecto opuesto al visto en el caso anterior. Esta correlación conlleva la existencia de una mayor cantidad de vectores no dominados. Para los algoritmos que realizan más operaciones de tipo vectorial, como es el caso de $IPID^*$ e *IP-MO-RBFS*, esto se traduce en una degradación del rendimiento, incrementándose la diferencia con IDA^* *+MO-DF-BnB* multiobjetivo a casi dos órdenes de magnitud para profundidades altas de las soluciones.

Como ya comentamos en esta sección, en las pruebas con correlación -1 los algoritmos $IPID^*$ e *IP-MO-RBFS* no consiguieron solucionar ningún problema, al contrario que IDA^* *+MO-DF-BnB* multiobjetivo.

8.8.3. Comparativa *MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS* sobre árboles finitos

En esta sección se incluyen los resultados de las diferentes pruebas realizadas para los mejores representantes de cada una de las categorías de algoritmos establecidas sobre un conjunto de árboles finitos.

Experimentos

Para la evaluación de los algoritmos se emplearon una serie de árboles finitos aleatorios (ver sección 8.3.1) con la siguiente configuración:

- Factor de ramificación: 2 (árboles binarios).
- Número de objetivos: 2.

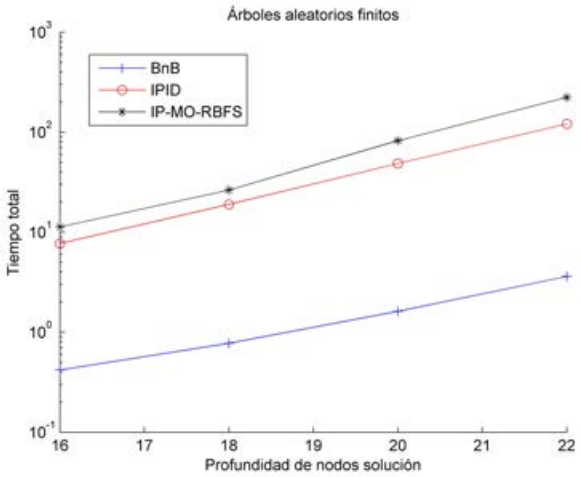


Figura 8.34: Comparativa *MO-DF-BnB*, $IPID^*$ e *IP-MO-RBFS*. Árboles finitos binarios. Tiempo de procesamiento.

- Profundidad a la que se encuentran las soluciones: niveles 16, 18, 20 y 22 del árbol de búsqueda, teniendo éste una profundidad máxima de 24 niveles.
- Rango de costes de ambos objetivos: [1,50].
- Correlación cero entre los objetivos.
- Heurístico: $\vec{h}(n) = 0, \forall n$.
- Porcentaje de la cantidad de nodos solución en la profundidad fijada para los mismos: 100 %. Estas soluciones pueden ser tanto óptimos de Pareto como soluciones subóptimas.
- Por cada combinación (profundidad soluciones, cantidad soluciones) se resolvieron un total de 20 problemas aleatorios diferentes (se presentan las medias).

Resultados

En la figura 8.34 se representa el tiempo de procesamiento (segundos) por cada uno de los algoritmos, promediado para los 20 problemas de cada tipo generados . En el eje X se representa la profundidad a la cual se encuentran los nodos finales del problema, mientras que en el eje Y se representa el tiempo total de procesamiento para cada algoritmo. Se ha empleado escala logarítmica para el eje Y.

Análisis

Los resultados obtenidos son análogos a los de las pruebas realizadas sobre árboles infinitos. La sencillez de *MO-DF-BnB*, que no precisa mantener una cota de expansión

adicional, sino que emplea como tal el propio conjunto de soluciones que va localizando, aún cuando pueda incluir de modo transitorio soluciones subóptimas, hace que el rendimiento sea notablemente superior a las otras dos opciones.

Por otro lado, y de modo similar a los experimentos de la sección 8.8.1, $IPID^*$ es más eficiente que *IP-MO-RBFS*, debido a que este último precisa calcular una nueva cota de expansión por cada nodo generado, mientras que $IPID^*$ lo hace por cada iteración del algoritmo.

8.9. Conclusiones generales

En este Capítulo hemos presentado una evaluación exhaustiva de las diferentes aproximaciones *depth-first* al problema de la búsqueda multiobjetivo. Las conclusiones generales que podemos obtener de las diferentes comparativas realizadas entre los algoritmos vistos son las siguientes:

- La implementación de un algoritmo multiobjetivo basado en profundización iterativa debe buscar un equilibrio entre la cantidad de iteraciones (y, por lo tanto, reexpansiones de nodos) que realiza y el tamaño y naturaleza (escalar o vectorial) del umbral que establece la cota de expansión para cada iteración.

- De los diferentes algoritmos con profundización iterativa incluídos en nuestra evaluación, $IPID^*$ se muestra a rasgos generales como la opción más eficiente. Más concretamente, $IPID^*$ mejora a $IDMOA^*$, la propuesta multiobjetivo de profundización progresiva previamente descrita en la literatura. Mientras $IDMOA^*$ usa un umbral escalar y procesa individualmente los objetivos, realizando una reexpansión excesiva de nodos, $PIDMOA^*$ emplea una cota para cada iteración formada, sobre todo durante la búsqueda de la primera solución, por un numeroso conjunto de vectores. Los tiempos de reexpansión, en el primer caso, y los asociados a los tests de dominancia, en el segundo, hacen que estos algoritmos no obtengan un rendimiento adecuado. $LEXIDMOA^*$, que emplea como cota un único vector, el mejor lexicográfico de los costes de los caminos discontinuados, reduce considerablemente la cantidad de tests de dominancia, pero a costa de un avance más lento de dicha cota y el consiguiente incremento de reexpansiones. $IPID^*$ mantiene también un umbral con un único vector, el punto ideal de los costes de los caminos discontinuados, compensando las reexpansiones que realiza con la sencillez de los tests que efectúa contra dicho umbral y el rápido avance del mismo.

- De los diferentes algoritmos basados en *RBFS* propuestos en este trabajo, *IP-MO-RBFS* es el que presenta mejor rendimiento, mejorando tanto el enfoque con umbral multivectorial, *Pareto-MO-RBFS*, como la propuesta previa basada en orden lexicográfico, $MOMA^*0$. La variante lineal de $MOMA^*0$ analizada no presenta ninguna mejora significativa del algoritmo. Al igual que sucede en

profundización iterativa, el peor rendimiento de estos dos últimos algoritmos se debe a la gran cantidad de tests de dominancia que induce la cota multivectorial en el primer caso, y al lento avance del umbral en el segundo. El esquema intermedio basado en el *punto ideal*, empleado por $IP\text{-}MO\text{-}RBFS$, compensa ambas situaciones.

- El uso de un umbral multivectorial, caso de $PIDMOA^*$ o *Pareto-MO-RBFS*, a pesar de reducir la cantidad de expansiones en el árbol de búsqueda, penaliza excesivamente el rendimiento, al obligar a comparar todos los nodos susceptibles de expansión contra dicho umbral. Esta ineficiencia es directamente proporcional a la profundidad del árbol explorado.

- El uso de órdenes lexicográficos, empleado en algoritmos como $LEXIDMOA^*$ ó $MOMA^*0$, no presenta un buen rendimiento debido a que escalona la búsqueda por objetivos.

- El concepto de *punto ideal* presenta buenos resultados en diferentes algoritmos *depth-first* tales como $IPID^*$ e $IP\text{-}MO\text{-}RBFS$, debido fundamentalmente a que plantea el uso de una cota de expansión sencilla, que reduce el número de tests de expansión, y que además recopila información de calidad acerca de los nodos pendientes de expansión.

- En caso de tratar con problemas infinitos, es preciso aportar una cota inicial a *MO-DF-BnB* para que pueda ser completo. El algoritmo que proporcione esa cota es el que marca la eficiencia de la combinación. La calidad de dicha cota inicial ha probado **no** ser importante en la eficiencia de *MO-DF-BnB* para localizar a posteriori la frontera de Pareto.

- En caso de tratar con problemas finitos, *MO-DF-BnB* es el algoritmo más eficiente, no aportando ninguna ventaja el cálculo previo de una cota incial de expansión, algo de todos modos innecesario para garantizar la completitud en este tipo de problemas.

- A pesar de las mejoras introducidas, las variantes multiobjetivo basadas en profundización iterativa y en *RBFS* no pueden mejorar a IDA^* +*MO-DF-BnB* en los problemas analizados.

En el siguiente Capítulo presentaremos las conclusiones de este trabajo y plantearemos posibles nuevas líneas de investigación.

Parte III

Conclusiones y trabajo futuro

En esta tercera parte se revisan los objetivos de esta tesis, detallando las conclusiones obtenidas de los análisis formales y empíricos realizados a lo largo de este trabajo. Finalmente se sugieren posibles líneas de trabajo futuro.

- En el capítulo 9 se recopilan las conclusiones obtenidas de este trabajo de investigación, sugiriendo posibles líneas futuras de trabajo.

Capítulo 9

Conclusiones y trabajo futuro

Muchos problemas del mundo real precisan de la optimización de varios objetivos de modo simultáneo. Dada la mayor complejidad de este tipo de problemas, el diseño y análisis de técnicas eficientes para búsqueda multiobjetivo está cobrando gran importancia dentro del área de la Inteligencia Artificial. De modo general, los algoritmos de búsqueda multiobjetivo se pueden clasificar en dos grandes grupos: los exactos, que devuelven el conjunto completo de soluciones óptimas, y los aproximados, que devuelven soluciones generalmente de alta calidad, pero posiblemente subóptimas.

Esta tesis está basada en el análisis de algoritmos exactos, concretamente de tipo *depth-first*, que para la exploración de un árbol requieren un consumo de memoria lineal con la profundidad del mismo. Estos algoritmos mantienen en memoria únicamente la rama del árbol de búsqueda explorada, frente a los algoritmos de tipo *best-first*, los cuales mantienen en memoria un árbol con todos los caminos interesantes explorados hasta el momento. Este ahorro de memoria se puede traducir, sin embargo, en el caso general, en un mayor coste temporal, debido a la necesidad de reexpansión de subárboles de búsqueda. Esta reexpansión, en el caso multiobjetivo, degrada rápidamente el rendimiento, dado que el tiempo de procesamiento de un nodo en este tipo de problemas es más alto que en la búsqueda escalar.

9.1. Contribuciones

El objetivo principal de esta tesis es la realización de un mapa detallado de los algoritmos multiobjetivo *depth-first* exactos previos, el diseño de nuevas variantes, generalmente basadas en enfoques escalares previos, y una evaluación exhaustiva del conjunto completo de algoritmos sobre problemas generados aleatoriamente para identificar sus debilidades y fortalezas, y determinar la mejor opción existente. En resumen, las principales contribuciones de esta tesis son las siguientes:

1. **Categorización sistemática de los algoritmos multiobjetivo exactos de tipo *depth-first*.**

Esta categorización está basada en dos criterios básicos: por un lado la estrategia de expansión empleada en la exploración de un árbol, y por otro lado la naturaleza de la cota empleada como límite de profundización para las ramas exploradas.

- *Estrategia de búsqueda.* Se distinguen tres grandes tipos de estrategias de búsqueda *depth-first*: la profundización iterativa (que realiza sucesivas reexpansiones del árbol explorado basadas en cotas actualizables para cada profundización), la búsqueda recursiva *best-first* (que realiza reexpansiones de ramas en base a cotas fijadas por cada nodo procesado) y la búsqueda secuencial ó por fases (bien con una fase de búsqueda por cada objetivo considerado, como el en caso de $IDMOA^*$, o bien con una primera etapa de localización de una cota inicial y una segunda etapa de profundización *MO-DF-BnB* con cotas de expansión fijadas por el conjunto de soluciones encontradas).
- *Naturaleza de la cota.* Las estrategias de profundización iterativa y *RBFS* utilizan cotas para controlar la profundidad de la búsqueda. La idoneidad de esta cota está ligada a su dimensionalidad y al criterio de selección. En cuanto a la dimensión, la cota puede ser un valor escalar, un vector o un conjunto de vectores. Por lo que respecta a la selección, los enfoques escalares determinan el objetivo a procesar en cada instante, siguiendo un criterio de minimización del mismo. En el caso de cotas multivectoriales, se emplea toda la frontera de Pareto, mientras que en las cotas formadas por un único vector, éste se selecciona con un criterio de minimización, bien de todas las componentes de coste (*punto ideal*), bien en en base a un orden total (lexicográfico o lineal).

El empleo de diferentes enfoques para cada uno de estos criterios genera las correspondientes variantes de búsqueda, muchas de las cuales han sido analizadas y, en algunos casos, diseñadas en este trabajo.

2. **Diseño y formalización de nuevos algoritmos multiobjetivo exactos de tipo *depth-first*.** El empleo de diferentes enfoques para cada uno de los criterios anteriormente mencionados genera una gran diversidad de algoritmos de búsqueda. Sin embargo, sólo una pequeña parte ha sido descrita previamente en la literatura, sin que existiera un análisis sistemático ni comparativo de sus ventajas e inconvenientes. Es por ello que una parte importante del trabajo aportado por esta tesis está relacionado con el diseño de nuevos algoritmos multiobjetivo, centrados fundamentalmente en el empleo de cotas basadas en el concepto de *orden lexicográfico* ($LEXIDMOA^*$ y variante de $MOMA^*0$) y en el uso de la frontera de Pareto ($PIDMOA^*$ y *Pareto-MO-RBFS*), tanto para una estrategia de profundización iterativa como recursiva *best-first*. Asimismo, se aporta como novedad destacable la aplicación del concepto de *punto ideal* al cálculo de la cota, también para ambas estrategias, dando lugar a algoritmos que han

demostrado ser los más eficientes en sus respectivas categorías. También se ha considerado la aplicación de *Branch & Bound* complementado con el cálculo de una cota inicial para el caso de árboles infinitos. En el caso de los nuevos algoritmos, se incluye un análisis formal de sus principales propiedades, completitud y admisibilidad, garantizando así que compiten en igualdad de condiciones.

3. **Evaluación empírica de los algoritmos multiobjetivo exactos de tipo *depth-first*.** La evaluación realizada ha permitido determinar la mejor opción para las diferentes categorías de estrategias de búsqueda analizadas (profundización iterativa, recursividad *best-first* y *Branch & Bound*). Asimismo, una posterior evaluación entre estos candidatos ha permitido determinar de modo general el algoritmo multiobjetivo exacto *depth-first* más conveniente: *Branch & Bound* multiobjetivo con cálculo de cota inicial basada en un algoritmo completo escalar (por ejemplo, IDA^* o *RBFS*). La comparativa ha estado basada fundamentalmente en la resolución de problemas en forma de árboles infinitos, en los cuales se ha jugado con parámetros tales como el rango de costes y la correlación entre los valores de los objetivos o la cantidad y profundidad de las soluciones disponibles.

El resto de este Capítulo presenta una descripción más detallada de las conclusiones y establece posibles líneas de trabajo futuro.

9.2. Conclusiones

Las principales conclusiones obtenidas de la investigación realizada se pueden resumir como sigue:

1. **Análisis de parámetros de rendimiento en la evaluación de algoritmos multiobjetivo.**

   Previamente a este trabajo se ha empleado el número de nodos expandidos como una medida habitual de la eficiencia de un algoritmo. En el caso escalar, esta medida puede no ser del todo equitativa, pues algunos algoritmos mantienen conjuntos de nodos ordenados, con la consiguiente sobrecarga en tiempo. En el caso multiobjetivo, el tiempo de procesamiento presenta una mayor variabilidad, dado que el coste vectorial de cada nodo tiene que ser generalmente comparado contra la cota de expansión del algoritmo *depth-first* y, adicionalmente, contra el conjunto de soluciones localizadas. Por lo tanto el esfuerzo asociado al procesamiento de un nodo viene determinado por la dimensión de la cota y del conjunto de soluciones. Es por ello que la cantidad de nodos expandidos no se perfila a priori como un buen parámetro de rendimiento en nuestro caso. El tiempo de procesamiento y, auxiliarmente, la cantidad de tests vectoriales (de dominancia o asociados a los diferentes órdenes empleados en las cotas) son una medida más realista de la eficiencia de los algoritmos multiobjetivo de cualquier tipo. Adicionalmente, otras medidas como la evolución del tamaño de los conjuntos

umbral o de soluciones a lo largo del proceso de búsqueda permiten entender el comportamiento de muchos de los algoritmos estudiados.

2. **Análisis de variantes basadas en profundización iterativa.**

   Antes de este trabajo no existía un algoritmo que generalizase de forma directa la profundización iterativa al caso multiobjetivo. La única aproximación previa, $IDMOA^*$, realiza un enfoque secuencial, procesando de modo independiente los diferentes objetivos con IDA^*. En esta tesis se presenta $PIDMOA^*$, una adaptación completa del algoritmo escalar IDA^* al caso multiobjetivo, con el uso de una cota de expansión formada por la frontera de Pareto del árbol de búsqueda expandido en cada iteración. También se presenta una variante que basa la profundización en un orden lexicográfico ($LEXIDMOA^*$). El uso de este tipo de órdenes es habitual en algoritmos multiobjetivo de tipo *best-first*, pero nunca había sido explorado en una estrategia *depth-first*. Por último se introduce una estrategia innovadora que controla la profundización usando el *punto ideal* de la frontera de Pareto ($IPID^*$). Todos los algoritmos propuestos son demostrablemente completos y admisibles bajo suposiciones razonables. El análisis empírico realizado fundamentalmente sobre árboles infinitos con diversos porcentajes de soluciones a profundidades crecientes muestra que:

   a) La cota basada en la frontera de Pareto es la que presenta peor rendimiento, ya que aunque avanza rápidamente, reduciendo el número de reexpansiones de nodos, su naturaleza multivectorial provoca un considerable aumento del número de tests de dominancia.

   b) $LEXIDMOA^*$ usa un único vector como cota, pero requiere un número muy elevado de profundizaciones, y por lo tanto de reexpansiones de nodos, fundamentalmente debido al empleo de orden lexicográfico, que motiva el que la cota trate en general de mejorar el coste para uno de los objetivos cada vez.

   c) $IPID^*$ es el mejor de los algoritmos presentados, ya que reduce el número de tests de dominancia, al emplear un único vector como cota, pero, a diferencia de $LEXIDMOA^*$, este vector contempla información relacionada con todos los objetivos considerados, lo que propicia un avance más rápido de la cota.

3. **Análisis de variantes basadas en búsqueda recursiva *best-first*.**

   En este caso, ya existía un algoritmo multiobjetivo basado en *RBFS* que emplea un orden lexicográfico ($MOMA^*0$). No obstante, no se había realizado nunca un análisis completo del algoritmo, ni tampoco sobre la bondad de su aproximación al enfoque recursivo *best-first*. En este trabajo se proponen sendas variantes basadas en un enfoque puramente multiobjetivo (*Pareto-MO-RBFS*, con el uso de una cota de expansión multivectorial) y en el cálculo del *Punto Ideal* (*IP-MO-RBFS*, con el uso de una cota formada por un único vector), de modo análogo

al diseño realizado en los algoritmos de profundización iterativa. Al igual que en el caso anterior, la evaluación realizada prueba la mejor eficiencia de la variante basada en *Punto Ideal*, debido nuevamente al empleo de una cota univectorial, la cual reduce la cantidad de operaciones a realizar contra la misma. Los algoritmos son también completos y admisibles bajo supuestos razonables y el análisis empírico ofrece unos resultados muy similares a los obtenidos para las variantes multiobjetivo de profundización iterativa:

a) Las cotas basadas en frontera de Pareto (*Pareto-MO-RBFS*) y en orden lexicográfico ($MOMA^*0$) muestran un bajo rendimiento, debido principalmente al gran número de tests de dominancia y gran cantidad de reexpansiones de nodos que deben realizar respectivamente.

b) La variante basada en *punto ideal* es la que muestra un mejor rendimiento, al conseguir un equilibrio entre los tests de dominancia (usando una cota de un único vector) y la cantidad de reexpansiones (calculando la cota con información de todos los objetivos).

4. **Análisis de *Branch & Bound* multiobjetivo.**

   Existen numerosas aplicaciones de *Branch & Bound* a la búsqueda multiobjetivo de tipo aproximado. La única variante de tipo exacto, *MO-DF-BnB*, únicamente permite resolver, en aplicación directa, problemas con un espacio de estados finito. Su adaptación a espacios de estados infinitos requiere localizar una cota superior inicial mediante un algoritmo completo bajo este tipo de problemas. Afortunadamente, para el caso multiobjetivo es relativamente fácil establecer esta cota inicial, incluso en el caso de árboles infinitos. Para ello se pueden utilizar, por ejemplo, algoritmos completos escalares. En esta tesis se han empleado y adaptado diferentes algoritmos *depth-first*, tanto escalares como multiobjetivo, para la búsqueda de esta cota. Las diferencias entre estos algoritmos radican fundamentalmente en la calidad de la cota y el tiempo de búsqueda de la misma. Los análisis realizados han mostrado que la calidad de la cota no es significativa. El esfuerzo realizado por algunos algoritmos como $PIDMOA^*$, que localizan una cota óptimo de Pareto, no se ve reflejado en una mayor eficiencia. De hecho, la evaluación muestra como el peso del rendimiento recae casi en su totalidad sobre el algoritmo inicial, siendo residual el tiempo empleado por *MO-DF-BnB* para calcular los restantes óptimos de Pareto. El compañero de viaje idóneo para *MO-DF-BnB* resulta ser un algoritmo de tipo escalar (por ejemplo, IDA^* ó *RBFS*) aplicado sobre uno de los objetivos considerados. En el caso de árboles finitos, *Branch & Bound* sin cota inicial se ha mostrado como la opción con mejor rendimiento.

5. **Análisis de las cotas de expansión**

   El uso de cotas en algoritmos de tipo *depth-first* es ineludible, con el fin de asegurar la completitud de los mismos. La naturaleza de estas cotas tienen un gran

impacto en el rendimiento. La apuesta por una cota escalar ($IDMOA^*$) simplifica enormemente las operaciones de control a realizar sobre los nodos, pero en la práctica su funcionamiento equivale a la aplicación secuencial del algoritmo IDA^* una vez por cada objetivo considerado. Esto ocasiona demasiadas reexpansiones de nodos, aún cuando el procesamiento de éstos sea relativamente sencillo. Un enfoque lexicográfico procesa de modo parecido el espacio de estados, ya que se centra en la mejora de una componente, procesando otras únicamente en caso de igualdad. Este comportamiento *cuasi* escalar empeora debido al mantenimiento de una cota vectorial. El uso de una cota multivectorial ($PIDMOA^*$ y *Pareto-MO-RBFS*), formada por los costes de los caminos que se van discontinuando, favorece una expansión más rápida del árbol, al contemplar diferentes caminos y todos los objetivos simultáneamente, reduciendo con ello la cantidad de nodos generados. Sin embargo, tal y como se ha visto en las pruebas realizadas, un incremento en la dificultad del problema (profundidad del espacio de estados ó correlación negativa entre objetivos), dispara la cantidad de vectores no dominados, y por consiguiente el tamaño del umbral y las operaciones vectoriales a realizar con el mismo. Un enfoque innovador, basado en *Punto Ideal*, mantiene un umbral constante (un único vector) que contempla la calidad de todos los objetivos. El exceso de reexpansiones (frente a algoritmos como $PIDMOA^*$) se ve compensado por el reducido número de operaciones vectoriales que realiza contra el umbral, resultando ser la mejor opción tanto para algoritmos de profundización iterativa ($IPID^*$) como de búsqueda recursiva *best-first* ($IP\text{-}MO\text{-}RBFS$).

6. **Análisis de la estrategia de búsqueda: mejor algoritmo *depth-first***

   La búsqueda basada en profundización iterativa mantiene sendas cotas de expansión: una que establece el límite de profundización para la iteración actual y otra definida por el conjunto de soluciones ya encontradas, que determina la discontinuidad de una rama de modo definitivo. Los algoritmos basados en *RBFS* mantienen estas mismas cotas, si bien en su caso la primera de ellas se calcula por cada nodo del árbol de búsqueda, mientras que en profundización iterativa se fija por iteración o profundización. *RBFS* no realiza profundizaciones sucesivas, sino *contracciones* y *expansiones* de caminos del árbol explorado, para mantener un consumo lineal de memoria. Aunque en el caso escalar, el algoritmo *RBFS* presenta mejor rendimiento que IDA^* bajo determinadas condiciones, las pruebas realizadas en esta tesis muestran que para el caso multiobjetivo, el comportamiento es diferente. El hecho de procesar cotas vectoriales por cada expansión de un nodo penaliza las variantes *RBFS*.

   Las variantes de profundización iterativa y *RBFS* basadas en el *punto ideal* ($IPID^*$, $IP\text{-}MO\text{-}RBFS$) han mejorado todos los algoritmos con profundización previos ($IDMOA^*$ y $MOMA^*0$), así como el resto de propuestas incluídas en este trabajo basadas en frontera de Pareto ($PIDMOA^*$ y *Pareto-MO-RBFS*) y orden lexicográfico ($LEXIDMOA^*$). Tal y como se ha comentado anteriormente,

esta mejora de rendimiento tiene su origen en el punto de equilibrio que consiguen estos algoritmos entre la complejidad de la cota (naturaleza y/o cantidad de vectores que la componen) y la información manejada para su actualización.

Sin embargo, la comparativa realizada en este trabajo muestra que la eficiencia de las estrategias anteriores queda eclipsada por el alto rendimiento de las variantes multiobjetivo de *Branch & Bound*. *MO-DF-BnB* emplea únicamente una cota superior, formada también por las soluciones que ha localizado el algoritmo en cada momento, expandiéndose un nodo mientras no esté dominado por alguna de dichas soluciones. Esto, aunque puede conllevar la expansión de una gran cantidad de nodos subóptimos, permite a su vez una poda muy eficiente del árbol de búsqueda. El empleo de una cota inicial, además de permitir el uso en espacios de estados finitos, mejora el rendimiento del algoritmo.

9.3. Trabajo futuro

Esta investigación ha realizado un estudio detallado de los algoritmos multiobjetivo exacto de tipo *depth-first*, completando con nuevas aportaciones las diferentes estrategias de búsqueda. Al mismo tiempo, han surgido nuevas cuestiones que plantean posibles futuras líneas de trabajo, entre las cuales caben destacar las siguientes:

- El empleo en esta investigación de un entorno de pruebas basado en árboles aleatorios infinitos ha facilitado un estudio exhaustivo del comportamiento de los algoritmos. Sin embargo, este campo de pruebas no permite el análisis de un heurístico de calidad. El heurístico empleado no aporta prácticamente información en el caso de problemas con un rango de costes elevado para los objetivos. En este aspecto sería interesante aplicar los algoritmos vistos a problemas reales que dispongan de un heurístico más informado, para confirmar si el comportamiento y la eficiencia de estos algoritmos se mantiene o presenta alguna variación.

- Una vez establecida la calidad de los algoritmos exactos analizados en esta tesis, sería interesante realizar una comparativa con otros algoritmos de carácter aproximado, de tal modo que se pueda evaluar la relación entre el rendimiento y la calidad de las soluciones obtenidas en base a la dificultad de los problemas seleccionados.

- La evaluación realizada en esta tesis ha probado que la naturaleza vectorial de los problemas multiobjetivo penaliza el rendimiento de cualquier tipo de algoritmos. Una reducción en la cantidad de vectores, y por consiguiente en la cantidad de operaciones a realizar con los mismos, tal y como se vio en los algoritmos basados en *Punto Ideal*, mejora notablemente los resultados. El desarrollo de tipos de datos y procedimientos mejorados para la realización de este tipo de operaciones vectoriales (test de dominancia, órdenes lineales y lexicográficos, etc.) puede favorecer el rendimiento de los diferentes algoritmos.

Bibliografía

Bausch, R. A multicriteria scheduling tool using a branch-and-bound algorithm. *European Journal of Operational Research*, vol. 61(1-2), páginas 215 – 218, 1992. ISSN 0377-2217. IFORS-SPC Conference on Decision Support Systems.

Bell, D., Keeney, R., Raïffa, H. y for Applied Systems Analysis, I. I. *Conflicting Objectives in Decisions*. International Series on Applied Systems Analysis. Wiley, 1977. ISBN 9780471995067.

Bell, D. E. Multiattribute utility functions: Decompositions using interpolation. *Management Science*, vol. 25(8), páginas 744–753, 1979.

Bellman, R. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, vol. 60(6), páginas 503–515, 1954.

Björnsson, Y., Enzenberger, M., Holte, R. C. y Schaeffer, J. Fringe search: Beating a* at pathfinding on game maps. En *CIG*. 2005.

Bogetoft, P. General communication schemes for multiobjective decision making. *European Journal of Operational Research*, vol. 26(1), páginas 108 – 122, 1986. ISSN 0377-2217. Second {EURO} Summer Institute.

Boxnick, S., Klopfer, S., Romaus, C. y Klopper, B. Multiobjective search for the management of a hybrid energy storage system. En *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, páginas 745–750. 2010.

Buer, T. y Pankratz, G. Solving a bi-objective winner determination problem in a transportation procurement auction. *Logistics Research*, vol. 2(2), páginas 65–78, 2010. ISSN 1865-035X.

Chakrabarti, P., Ghose, S., Acharya, A. y de Sarkar, S. Heuristic search in restricted memory. *Artificial Intelligence*, vol. 41(2), páginas 197–222, 1989.

CHANKONG, V. y HAIMES, Y. *Multiobjective Decision Making: Theory and Methodology*. North-Holland, 1983.

CHARNETSKI, J., ÉCOLE POLYTECHNIQUE (MONTRÉAL, Q. D. D. G. I. y SOLAND, R. *Multiple Attribute Decision Making with Partial Information : the Comparative Hypervolume Criterion*. Rapport technique - Ecole polytechnique de Montréal. Département de génie industriel, Ecole polytechnique de Montréal, 1977.

CLÍMACO, J. C. N., CRAVEIRINHA, J. M. F. y PASCOAL, M. M. B. A bicriterion approach for routing problems in multimedia networks. *Networks*, vol. 41(4), páginas 206–220, 2003.

COEGO, J. Backtracking search algorithms for multicriteria problems. En *Doctoral Consortium, XIII Conference of the Spanish Association for Artificial Intelligence (CAEPIA'09)*. 2009.

COEGO, J., MANDOW, L. y PÉREZ DE LA CRUZ, J. L. A new approach to iterative deepening multiobjective A*. En *AI*IA 2009, LNCS 5883*, páginas 264–273. 2009.

COEGO, J., MANDOW, L. y PÉREZ DE LA CRUZ, J. L. A comparison of multiobjective depth-first algorithms. *Journal of Intelligent Manufacturing*, 2010.

COEGO, J., MANDOW, L. y PÉREZ DE LA CRUZ, J. L. Ideal point guided iterative deepening. En *ECAI 2012, LNCS 5883*, páginas 246–251. 2012.

COHON, J. *Multiobjective Programming and Planning*. Dover Books on Computer Science Series. Dover Publications, 2004. ISBN 9780486432632.

DASGUPTA, P., CHAKRABARTI, P. y DESARKAR, S. *Multiobjective Heuristic Search*. Vieweg, Braunschweig/Wiesbaden, 1999.

DECHTER, R. y PEARL, J. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, vol. 32(3), páginas 505–536, 1985.

DELORT, C. y SPANJAARD, O. A hybrid dynamic programming approach to the biobjective binary knapsack problem. *ACM Journal of Experimental Algorithmics*, vol. 18, 2013.

DEUTSCH, D. N. A “dogleg” channel router. En *Papers on Twenty-five Years of Electronic Design Automation*, 25 years of DAC, páginas 111–119. ACM, New York, NY, USA, 1988. ISBN 0-89791-267-5.

DYER, J. S. y SARIN, R. K. Measurable multiattribute value functions. *Operations Research*, vol. 27(4), páginas 810–822, 1979.

ESCHENAUER, H., KOSKI, J. y OSYCZKA, A. Multicriteria optimization: Fundamentals and motivation. En *Multicriteria Design Optimization* (editado por H. Eschenauer, J. Koski y A. Osyczka), páginas 1–32. Springer Berlin Heidelberg, 1990. ISBN 978-3-642-48699-9.

GALAND, L., ISMAILI, A., PERNY, P. y SPANJAARD, O. Bidirectional preference-based search for state space graph problems. En *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013.*. 2013.

GALAND, L. y PERNY, P. Search for compromise solutions in multiobjective state space graphs. En *Proc. of the XVII European Conference on Artificial Intelligence (ECAI'2006)*, páginas 93–97. 2006.

GALAND, L., PERNY, P. y SPANJAARD, O. A branch and bound algorithm for choquet optimization in multicriteria problems. En *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems - Proceedings of the 19th International Conference on Multiple Criteria Decision Making, Auckland, New Zealand, January 7-12, 2008.*, páginas 355–365. 2008.

GALAND, L., PERNY, P. y SPANJAARD, O. Choquet-based optimisation in multiobjective shortest path and spanning tree problems. *European Journal of Operational Research*, vol. 204(2), páginas 303–315, 2010.

GALAND, L. y SPANJAARD, O. Exact algorithms for owa-optimization in multiobjective spanning tree problems. *Computers & OR*, vol. 39(7), páginas 1540–1554, 2012.

GOICOECHEA, A., HANSEN, D. y DUCKSTEIN, L. *Multiobjective decision analysis with engineering and business applications*. Wiley, 1982. ISBN 9780471064015.

GONZALEZ-RODRIGUEZ, I., VELA, C. y PUENTE, J. A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty. *Journal of Intelligent Manufacturing*, vol. 21(1), páginas 65–73, 2010. ISSN 0956-5515.

DE GROOT, A. *Thought and Choice in Chess*. Mouton & Co Publishers, The Hague, The Netherlands, 1965. ISBN 90-279-7914-6.

HAIMES, Y. y CHANKONG, V. *Decision making with multiple objectives: proceedings of the Sixth International Conference on Multiple-Criteria Decision Making, held at the Case Western University, Cleveland, Ohio, USA, June 4-8, 1984*. Lecture notes in economics and mathematical systems. Springer, 1985. ISBN 9783540152231.

HANSEN, P. Bicriterion path problems. En *Lecture Notes in Economics and Mathematical Systems 177*, páginas 109–127. Springer, 1979.

HARIKUMAR, S. y KUMAR, S. Iterative deepening multiobjective A*. *Information Processing Letters*, vol. 58, páginas 11–15, 1996.

HARIKUMAR, S. y KUMAR, S. Multiobjective search based algorithms for circuit partitioning problem for acceleration of logic simulation. En *Tenth Int. Conf. on VLSI Design*, páginas 239–242. 1997.

HART, P., NILSSON, N. y RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics SSC-4*, vol. 2, páginas 100–107, 1968.

HATEM, M., KIESEL, S. y RUML, W. Recursive best-first search with bounded overhead. En *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, páginas 1151–1157. 2015.

HWANG, C. y MASUD, A. *Multiple objective decision making, methods and applications: a state-of-the-art survey*. Lecture notes in economics and mathematical systems. Springer-Verlag, 1979. ISBN 9780387091112.

KEENEY, R. y RAIFFA, H. *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Wiley series in probability and mathematical statistics. Applied probability and statistics. Cambridge University Press, 1993. ISBN 9780521438834.

KOK, M. The interface with decision makers and some experimental results in interactive multiple objective programming methods. vol. 26(1), páginas 96–107, 1986.

KORF, R., ZHANG, W., THAYER, I. y HOHWALD, H. Frontier search. *JACM*, vol. 52(5), páginas 715–748, 2005.

KORF, R. E. Depth-first iterative-deepening A*: an optimal admissible tree search. En *Proc. of the IX Int. Joint Conf. on Artificial Intelligence (IJCAI'85)*, páginas 1034–1036. 1985a.

KORF, R. E. Depth first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, vol. 27, páginas 97–109, 1985b.

KORF, R. E. Linear-space best-first search: Summary of results. En *AAAI*, páginas 533–538. 1992.

KORF, R. E. Linear-space best-first search. *Artificial Intelligence*, vol. 62, páginas 41–78, 1993.

KORF, R. E. Algorithms and theory of computation handbook. capítulo Artificial Intelligence Search Algorithms, páginas 22–22. Chapman & Hall/CRC, 2010. ISBN 978-1-58488-820-8.

KORF, R. E. y CHICKERING, D. M. Best-first minimax search. *Artif. Intell.*, vol. 84(1-2), páginas 299–337, 1996. ISSN 0004-3702.

KORHONEN, P. A hierarchical interactive method for ranking alternatives with multiple qualitative criteria. vol. 24(2), páginas 265–276, 1986.

KORHONEN, P., MOSKOWITZ, H. y WALLENIUS, J. A progressive algorithm for modeling and solving multiple-criteria decision problems. *Oper. Res.*, vol. 34(5), páginas 726–731, 1986. ISSN 0030-364X.

LÁSZLÓ MÉRÕ. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, vol. 23(1), páginas 13–27, 1984.

LAWLER, E., SHMOYS, D., LENSTRA, J. y KAN, A. A. *The Traveling Salesman Problem*. Wiley Interscience Series in Discrete Mathematics. John Wiley & Sons, 1985. ISBN 9780471904137.

LAWLER, E. L. y WOOD, D. E. Branch-and-bound methods: A survey. *Operations Research*, vol. 14(4), páginas 699–719, 1966.

LI, X., YALAOUI, F. y AMODEO, L. Metaheuristics and exact methods to solve a multiobjective parallel machines scheduling problem. *Journal of Intelligent Manufacturing*, vol. 21(1), páginas 89–99, 2010.

LIU, Z., WONG, Y. y LEE, K. A manufacturing-oriented approach for multi-platforming product family design with modified genetic algorithm. *Journal of Intelligent Manufacturing*, vol. 22(6), páginas 891–907, 2011. ISSN 0956-5515.

MACHUCA, E., MANDOW, L. y GALAND, L. An evaluation of best compromise search in graphs. En *Advances in Artificial Intelligence - 15th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2013, Madrid, Spain, September 17-20, 2013. Proceedings*, páginas 1–11. 2013.

MANDOW, L. y DE LA CRUZ, J. A memory-efficient search strategy for multiobjective shortest path problems. En *KI 2009: Advances in Artificial Intelligence* (editado por B. Mertsching, M. Hund y Z. Aziz), vol. 5803 de *Lecture Notes in Computer Science*, páginas 25–32. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04616-2.

MANDOW, L. y PÉREZ DE LA CRUZ, J. L. Multicriteria heuristic search. *European Journal of Operational Research*, vol. 150, páginas 253–280, 2003.

MANDOW, L. y PÉREZ DE LA CRUZ, J. L. A new approach to multiobjective A* search. En *Proc. of the XIX Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, páginas 218–223. 2005.

MANDOW, L. y PÉREZ DE LA CRUZ, J. L. Multiobjective A* search with consistent heuristics. *Journal of the ACM*, vol. 57(5), páginas 27:1–25, 2010.

MARTINS, E. On a multicriteria shortest path problem. *European Journal of Operational Research*, vol. 16, páginas 236–245, 1984.

MCFARLAND, M. C., PARKER, A. C. y CAMPOSANO, R. Tutorial on high-level synthesis. En *Proceedings of the 25th ACM/IEEE Design Automation Conference*, DAC '88, páginas 330–336. IEEE Computer Society Press, Los Alamitos, CA, USA, 1988. ISBN 0-8186-8864-5.

MOND, B. y ROSINGER, E. E. Interactive weight assessment in multiple attribute decision making. vol. 22(1), páginas 19–25, 1985.

MÜLLER-HANNEMANN, M. y WEIHE, K. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals OR*, vol. 147(1), páginas 269–286, 2006.

NAVINCHANDRA, D. *Exploration and innovation in design: towards a computational model*. Symbolic computation: Artificial intelligence. Springer-Verlag, 1991. ISBN 9780387974811.

OPPENHEIMER, K. *Proxy Approach to Multi-attribute Decision Making*. Department of Engineering-Economic Systems, Stanford University., 1977.

PEARL, J. *Heuristics*. Addison-Wesley, Reading, Massachusetts, 1984.

PÉREZ-DE-LA-CRUZ, J., MANDOW, L. y MACHUCA, E. A case of pathology in multiobjective heuristic search. *J. Artif. Intell. Res. (JAIR)*, vol. 48, páginas 717–732, 2013.

PERNY, P. y SPANJAARD, O. Near admissible algorithms for multiobjective search. En *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, páginas 490–494. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008. ISBN 978-1-58603-891-5.

PULIDO, F. J., MANDOW, L. y DE-LA CRUZ, J.-L. P. A two-phase bidirectional heuristic search algorithm. En *STAIRS* (editado por K. Kersting y M. Toussaint), vol. 241 de *Frontiers in Artificial Intelligence and Applications*, páginas 240–251. IOS Press, 2012. ISBN 978-1-61499-095-6.

PULIDO, F. J., MANDOW, L. y PÉREZ-DE-LA-CRUZ, J. Multiobjective shortest path problems with lexicographic goal-based preferences. *European Journal of Operational Research*, vol. 239(1), páginas 89–101, 2014.

REFANIDIS, I. y VLAHAVAS, I. Multiobjective heuristic state-space planning. *Artificial Intelligence*, vol. 145, páginas 1–32, 2003.

REINEFELD, A. y MARSLAND, T. Enhanced iterative-deepening search. *IEEE Trans Pattern Anal Mach Intell*, vol. 16, páginas 701–710, 1994.

ROLLÓN, E. y LARROSA, J. Bucket elimination for multiobjective optimization problems. *Journal of Heuristics*, vol. 12(4-5), páginas 307–328, 2006. ISSN 1381-1231.

ROLLÓN, E. y LARROSA, J. Constraint optimization techniques for multiobjective branch and bound search. En *Lecture Notes in Economics and Mathematical Systems, Vol. 618*, páginas 89–98. 2009.

ROMERO, C. *Teoría de la decisión multicriterio: Conceptos, técnicas y aplicaciones*. Alianza Universidad Textos. Alianza Editorial, 1993. ISBN 84-206-8144-X.

SARKAR, U. K., CHAKRABARTI, P. P., GHOSE, S. y SARKAR, S. C. D. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artif. Intell.*, vol. 50(2), páginas 207–221, 1991.

SCHOEMAKER, P. J. H. y WAID, C. C. An experimental comparison of different approaches to determining weights in additive utility models. *Management Science*, vol. 28(2), páginas 182–196, 1982.

SLATE, D. J. y ATKIN, L. R. Chess 4.5 - the northwestern university chess program. En *Chess Skill in Man and Machine* (editado por P. W. Frey), páginas 82–118. Springer-Verlag New York, Inc., 1979. ISBN 0387079572.

SOEIRO FERREIRA, J., ANTONIO NEVES, M. y FONSECA E CASTRO, P. A two-phase roll cutting problem. vol. 44(2), páginas 185–196, 1990.

SOURD, F. y SPANJAARD, O. A multiobjective branch-and-bound framework: Application to the bi-objective spanning tree problem. *INFORMS Journal on Computing*, vol. 20(3), páginas 472–484, 2008.

SPRONK, J. y ZIONTS, S. Introduction to special issue on multiple criteria decision making. *Management Science*, vol. 30/11, páginas 1265–1267, 1984.

STEUER, R. E. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, New York, 546 pp, 1986.

STEWART, B. S. y WHITE, C. C. Multiobjective A*. *Journal of the ACM*, vol. 38(4), páginas 775–814, 1991.

SYKES, E. A. y WHITE, I., C.C. Multiobjective intelligent computer-aided design. *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21(6), páginas 1498–1511, 1991. ISSN 0018-9472.

TUNG, C. T. y CHEW, K. L. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, vol. 62, páginas 203–209, 1992.

VANSNICK, J.-C. On the problem of weights in multiple criteria decision making (the noncompensatory approach). *European Journal of Operational Research*, vol. 24(2), páginas 288 – 294, 1986. ISSN 0377-2217. Mathematical Programming Multiple Criteria Decision Making.

VEMPATY, N. R., KUMAR, V. y KORF, R. E. Depth-first versus best-first search. En *AAAI*, páginas 434–440. 1991.

WEBER, M. A method of multiattribute decision making with incomplete information. *Management Science*, vol. 31(11), páginas 1365–1371, 1985.

WHITE, D. A branch and bound method for multi-objective boolean problems. *European Journal of Operational Research*, vol. 15(1), páginas 126 – 130, 1984. ISSN 0377-2217.

WHITE, D. J. The set of efficient solutions for multiple objective shortest path problems. *Computers & OR*, vol. 9(2), páginas 101–107, 1982.

YU, P., LEE, Y. y STAM, A. *Multiple-criteria decision making: concepts, techniques, and extensions*. Mathematical concepts and methods in science and engineering. Plenum Press, 1985. ISBN 9780306419652.

ZELENY, M. *Multiple Criteria Decision Making*. McGraw-Hill Series in Quantitative Methods for Management. McGraw-Hill, 1982. ISBN 9780070727953.

ZHANG, W. *State-Space Search: Algorithms, Complexity, Extensions, and Applications*. Springer, 1999.

ZHANG, W. y KORF, R. Performance of linear-space search algorithms. *Artificial Intelligence*, vol. 79, páginas 241–292, 1995.

ZHANG, W. y KORF, R. E. Depth-first vs. best-first search: New results. En *AAAI*, páginas 769–775. 1993.