Universidad Carlos III de Madrid

TESIS DOCTORAL

Búsqueda Heurística en Planificación Basada en Costes

Autor: Dª. Raquel Fuentetaja Pizán
Directores: Dr. D. Daniel Borrajo Millán y Dr. D. Carlos Linares López

Departamento de Informática

Leganés, Junio 2010

TESIS DOCTORAL

Búsqueda Heurística en Planificación Basada en Costes

Autor: Dª. Raquel Fuentetaja Pizán

Directores: Dr. D. Daniel Borrajo Millán y Dr. D. Carlos Linares López

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Vocal:

Vocal:

Secretario:

Calificación:

Leganés, ... de 2010

A mi madre
A mis chicos: Carlos y Héctor

Agradecimientos

Me gustaría dejar constancia de mi agradecimiento a todas las personas que me han apoyado y ayudado en el desarrollo de esta tesis. En primer lugar a mis directores, Daniel Borrajo y Carlos Linares, por todo lo que me han enseñado. Por el tiempo dedicado a revisiones y reuniones. Por darme la oportunidad en empezar este trabajo y por ayudarme a terminarlo. Por los esfuerzos en infundirme confianza en lo que hago, algo de lo que suelo carecer.

A todos los que son y han sido miembros del grupo PLG. En especial, a Susana por su amistad y apoyo. A Sergio por ser como es. Por nuestras charlas de *planning* camino a la piscina. A Tomás. Yo también quiero ser como tú. Al final, en ocasiones, hemos conseguido no hablar de nodos. A Fernando, por sus consejos y su envidiable actitud. A Rocio, por escucharme siempre. A Angel, por su enfoque positivo, y por usar mi planificador.

A la gente del grupo de planificación en Barcelona, por su acogida y sus comentarios. Especialmente a Emil Keyder, por prestarme sus dominios.

A los demás compañeros de la universidad que me faltan por mencionar. En especial a Javier Carbó y a Agapito. Llevamos ya muchos cafés, que hacen que cada día sea mejor. Gracias por vuestros ánimos y por las conversaciones de papás.

Tengo la suerte de contar con muy buenos amigos. Gracias a todos vosotros por vuestro apoyo y por los ratos de evasión que me regaláis. A Mónica y a Gema. A las chicas de Segovia y a los de Madrid. A Edu y Esther; y a mis compañeros pontificios.

Por supuesto, gracias a mi familia. A mi madre, por su fortaleza. Por enseñarme a mirar hacia delante. A mi padre, por confiar en mi. A mi segunda madre, mi abuela Petra. El mejor ejemplo que se puede tener. Por su entrega. Ójala estuviérais aquí. A mi hermano y a Sonia. Gracias por vuestra cercanía y por todo lo que me dais.

Finálmente y siempre, a Héctor y a mi hijo Carlos. No se que habría hecho sin vosotros.

Resumen

En los últimos años se han producido avances importantes en el mundo de la Planificación Automática. La Planificación Automática es una rama de la Inteligencia Artificial que se centra principalmente en el estudio de técnicas computacionales genéricas para resolver tareas de planificación. Las tareas de planificación típicamente consisten en obtener un conjunto ordenado de acciones, cuya ejecución permite alcanzar unos objetivos determinados. La dificultad más importante que presentan este tipo de tareas es que su resolución tiene un coste computacional muy elevado. Por este motivo, muchos planificadores utilizan heurísticas para guiar su proceso de planificación. Actualmente, una de las técnicas con más éxito es la *Búsqueda Heurística*.

Gran parte de la investigación en Planificación Automática se ha centrado en las técnicas desde un punto de vista teórico, utilizando dominios simplificados. Estas técnicas se han ido dotando de mecanismos que las hacen cada vez más capaces de trabajar con características más cercanas al mundo real. Entre otras, una de estas características es que las acciones suelen tener un coste asociado. La incorporación de los costes de las acciones en la tarea de planificación da lugar a la *Planificación Basada en Costes*. El interés por la planificación basada en costes es relativamente reciente. Así, el estudio de distintas heurísticas y las relaciones entre ellas, y de su comportamiento, bien de forma aislada o combinada, con distintos algoritmos de búsqueda es bastante limitado.

Esta tesis estudia técnicas de búsqueda heurística aplicadas a planificación basada en costes. Cuenta con una parte más teórica, en la que se ha generado una heurística numérica, se han adaptado a planificación basada en costes otras heurísticas, y se ha establecido un marco teórico general que permite comparar y definir heurísticas numéricas que pertenecen a una misma clase. Asimismo, se han propuesto una serie de algoritmos de búsqueda heurística y algunas variantes de los mismos. En la parte más experimental se ha valorado el comportamiento de los algoritmos combinados con distintas heurísticas. Como resultado, se ha obtenido una combinación heurísticas-algoritmo con un comportamiento muy adecuado en planificación basada en costes.

Abstract

Automated Planning has achieved significant progress in the last years. Automated Planning is the area of Artificial Intelligence that studies the computational techniques for solving planning tasks in a generic way. Planning tasks are tasks whose objective is to obtain an ordered set of actions such that, if applied, some objectives are reached. The most important difficulty for solving planning tasks is that their computational cost is very high. For this reason, most planners are endowed with heuristics to guide their search process. Currently, one of the most successful techniques for solving planning tasks is *Heuristic Search*.

A great part of the research in Automated Planning have focussed on the techniques from a theoretical point of view, using simplified domains. Then, the techniques have been provided with mechanisms to be able of managing more real-world features. One such features is that, usually, actions in real-world have costs. When the action costs are incorporated in the planning task, the task is called *Cost-Based Planning*. The increment of the interest in Cost-Based Planning is relatively recent. Thus, the study of different heuristics, their relationships, and the behavior of such heuristics in combination with search algorithms is rather limited.

This thesis studies different heuristic search techniques applied to Cost-Based Planning. From the theoretical point of view, it includes the development of a new heuristic, the adaptation of other heuristics from planning without costs, and a general theoretical framework for defining and comparing heuristics of the same class. In addition, several search algorithms and variants have been proposed for planning with action costs. From the empirical point of view, it has been performed several experiments in order to determine which combination of heuristics and algorithms are more adequate. As a result, we have obtained a heuristics-algorithm combination with very good performance.

Índice general

Índice de figuras

Índice de tablas

Parte I

Introducción y estado de la cuestión

Capítulo 1

Introducción

En este capítulo se introduce la tesis doctoral mediante la descripción del contexto en que se sitúa el trabajo, los causas que han motivado su desarrollo y los objetivos planteados inicialmente. Tanto la motivación como los objetivos se derivan del estudio del estado de la cuestión del capítulo siguiente.

1.1. Contexto de la tesis doctoral

Esta tesis doctoral se enmarca en el campo de la Inteligencia Artificial y dentro de éste en la Planificación Automática. La Planificación Automática es el área de la Inteligencia Artificial que estudia los procesos de planificación desde un punto de vista computacional. Normalmente se entiende que un proceso de planificación es un proceso abstracto y deliberativo orientado a determinar y organizar acciones, suponiendo que se conocen *a priori* sus efectos. El resultado del proceso es un *plan*, es decir, un conjunto de acciones ordenado total o parcialmente, cuya ejecución debe alcanzar ciertos objetivos a partir de un estado inicial determinado.

La investigación en Planificación Automática comenzó a finales de la década de los 50 como aplicación por separado de las técnicas de demostración de teoremas y de búsqueda. Respecto a la demostración de teoremas, McCarthy propuso la utilización del cálculo de predicados y de situaciones como un mecanismo para proporcionar un método deductivo de razonamiento inteligente (Mccarthy and Hayes, 1969). Un problema de planificación se expresaba mediante axiomas del cálculo de situaciones, que posteriormente se utilizaban para inferir secuencias de acciones. Por otro lado, las técnicas de búsqueda estaban orientadas a resolver problemas de planificación mediante la aplicación de una secuencia de transformaciones. En cada paso, se selecciona la alternativa a aplicar, que transforma una situación en otra. La construcción de planes de acciones se veía entonces como un proceso formal. Se construyeron sistemas como GPS (*General Problem Solver*) (Newell and Simon, 1995) y el sistema de planificación QA3 de Green (Green, 1969) como un proceso de demostración de teoremas. En los años 70 se desarrolló STRIPS (*STanford Institute Problem Solver*) (Fikes and Nilsson, 1971), una aplicación práctica en el campo de la robótica que integraba las

técnicas de demostración de teoremas y búsqueda. La acogida de STRIPS fue tan grande que sus ideas sobre la representación del espacio de situaciones se siguen utilizando en la actualidad, constituyendo la base del modelo de planificación clásica: dado un estado inicial, un conjunto de metas y un conjunto de operadores que permiten hacer transiciones entre estados, para los cuales se expresan sus pre y post-condiciones, el algoritmo de planificación debe encontrar una secuencia de operadores instanciados (un plan), que ejecutados a partir del estado inicial alcancen un estado en el que sean ciertas todas las metas. A partir de STRIPS se han desarrollado una gran cantidad de planificadores que siguen muy diversas estrategias de planificación.

Una cuestión que ha venido siendo de crucial importancia en el mundo de la planificación es la **eficiencia** de los planificadores. Esto se debe a que encontrar un plan puede llegar a ser una tarea computacionalmente muy cara, dependiendo de la complejidad del dominio y del problema a resolver. Desde el punto de vista teórico, el problema de determinar si existe alguna solución para un problema de planificación que se representa según el lenguaje proposicional de STRIPS, es en general *PSPACE-complete* (Bylander, 1994).[1] Debido a esto, muchos investigadores han centrado sus objetivos en el diseño de técnicas y estrategias para conseguir planificadores eficientes. La motivación es simple: de nada sirve contar con un planificador que no es capaz de encontrar ningún plan. Aunque conceptualmente el funcionamiento del planificador sea correcto, si su uso supone un consumo de recursos tal que agota las capacidades de las máquinas actuales en términos de memoria o tarda un tiempo inaceptable, el planificador no sirve de gran cosa. Así, durante algún tiempo, el objetivo principal de la planificación ha sido que los planificadores fueran capaces de encontrar algún plan, independientemente de la calidad de éste. Tener en cuenta algún tipo de medida de calidad hace la tarea de planificar aún más compleja, ya que en este caso no es suficiente con encontrar cualquier plan, sino que se requiere que el plan tenga una calidad mínima para que sea útil haberlo encontrado. Por este motivo, el primer criterio que se planteó utilizar para valorar los planificadores en la primera Competición Internacional de Planificación (IPC), en el año 1998, fue el tiempo de ejecución, de manera que se consideraban mejores los planificadores más rápidos. Dado que un único criterio no resultaba suficiente, se combinó con un segundo criterio: la longitud de los planes generados. Estos dos criterios se han mantenido a través de las siguientes competiciones, que se han venido organizando cada dos años, hasta la celebrada en el año 2008. Aunque la longitud de los planes generados sí que es una forma de medir la **calidad** de los planes, no siempre es la más adecuada. En muchos de los problemas del mundo real como planificación en procesos de manufactura (Vancza and Markus, 2001), planificación de recursos en empresas (Adam and Sammon, 2004), planificación aplicada al mundo militar (Upal, 2003), planificación de las actividades de los *rovers* en Marte (Joslin et al., 2005) etc., no es suficiente con conseguir el plan más corto. Lo normal es que en los dominios reales sea necesario optimizar una o varias medidas de calidad como el coste del plan, el tiempo de ejecución del plan, la cantidad de recursos que se consumen, etc. En los últimos años, el hecho de que los planificadores hayan ido siendo más eficientes ha provocado que se haya prestado un poco más de atención a la calidad de los planes. Así, en la competición del año 2002 participaron algunos planificadores que manejaban funciones multicriterio para

[1] Se suele asumir que los problemas *PSPACE-complete* son más difíciles que los problemas *NP-complete*.

medir la calidad de los planes generados. En esta competición hubo un apartado especial en el que uno de los criterios para valorar los planificadores era esta calidad. En realidad, lo interesante es que los planificadores sean capaces de conseguir planes de *buena* calidad, sin sacrificar demasiado la eficiencia. El extremo es la planificación óptima, aunque hoy en día este tipo de planificación supone un sacrificio excesivo de la eficiencia.

Uno de los modelos más sencillos de planificación con métricas de calidad es el que asume que cada acción tiene un coste fijo y el coste total de un plan se obtiene de forma aditiva, sumando los costes de las acciones que contiene. En este caso, la calidad de los planes es inversamente proporcional a su coste. Esta tesis doctoral se centra en la planificación automática asumiendo este modelo de planificación con métricas de calidad, que denominaremos *Planificación Basada en Costes*. El problema de la planificación basada en costes se puede solucionar utilizando distintas estrategias de planificación. Sin embargo, esta tesis estudia exclusivamente técnicas basadas en búsqueda heurística en el espacio de estados que genera directamente el propio problema de planificación. Las técnicas de búsqueda heurística han tenido mucho éxito en los últimos años, en parte debido a la aparición de heurísticas potentes que se calculan de forma independiente del dominio.

1.2. Motivación de la tesis doctoral

La resolución de problemas de planificación basada en costes de una forma relativamente eficiente es un problema abierto que hasta los últimos años no ha suscitado gran interés, en gran parte debido a que la planificación tipo STRIPS sin métrica de calidad ya es suficientemente compleja. Ésta es la principal razón que motiva el trabajo de la tesis que se presenta. Partiendo de esta motivación y teniendo en cuenta las técnicas que se han aplicado para resolver problemas de planificación basada en costes, existen razones más específicas que motivan este trabajo, como las que se describen a continuación en relación con la aplicación de técnicas de búsqueda heurística:

1. La aplicación de distintos algoritmos de búsqueda se ha evaluado con mucho más detalle en planificación sin costes (es decir, asumiendo costes unitarios). Sin embargo, el comportamiento del mismo algoritmo puede diferir bastante cuando se aplica a planificación basada en costes. El problema a resolver es distinto: en lugar de encontrar el plan de menor longitud se trata de encontrar el plan de menor coste, lo que supone que las heurísticas que se utilizan para guiar al algoritmo de búsqueda también sean distintas, ya que deben ser capaces de estimar costes.

2. La generación de heurísticas que se calculan de forma independiente del dominio ha estado a su vez mucho más orientada a la planificación clásica. Algunas de las heurísticas que se han propuesto para planificación basada en costes (como la de METRIC-FF (Hoffmann, 2003)) son aparentemente mejorables. Otras, aunque su definición es lo suficientemente general para ser utilizadas en planificación basada en costes no se han evaluado o se han evaluado mínimamente desde el punto de vista experimental para este tipo de planificación.

3. Respecto a las heurísticas numéricas, que estiman el coste de alcanzar un estado meta desde un estado fuente mediante un valor numérico, éstas se han venido definiendo

desde distintos puntos de vista. Algunas tienen definiciones procedurales, mientras que otras tienen definiciones declarativas. Ambos tipos de definiciones son correctas y de hecho se complementan. Es tan necesario definir *cómo* se calcula la heurística, como *qué* es lo que se calcula. Sin embargo, la falta de unificación hace que sea relativamente complicado entender las similitudes y diferencias entre las heurísticas.

4. Las IPCs permiten determinar los planificadores más competitivos. Sin embargo, muchos de los planificadores que participan, y en particular los que aplican búsqueda heurística, por lo general deben su buen funcionamiento a una combinación de heurísticas acertada. Dado que algunas heurísticas son más adecuadas para unos dominios que para otros (o incluso en el mismo dominio, para distintos problemas), la diversificación y/o la combinación de heurísticas permite que el planificador tenga buen funcionamiento en un amplio rango de dominios. La parte negativa es que es bastante complicado determinar por qué un planificador funciona bien en un determinado dominio/problema, ya que se desconoce cuál de los criterios heurísticos es el que más aporta al algoritmo de búsqueda para encontrar la solución. Para esto se requiere hacer una evaluación experimental en la que se aisle cada una (o al menos alguna combinación) de las heurísticas que el planificador utiliza.

Estas razones dan lugar a los objetivos que se describen a continuación.

1.3. Objetivos de la tesis doctoral

El objetivo principal de la tesis es profundizar tanto desde el punto de vista teórico como desde el punto de vista experimental, en los mecanismos que subyacen a las técnicas de búsqueda heurística en el marco de la planificación basada en costes, investigando:

1. Diferentes aproximaciones para hacer estimaciones heurísticas de forma independiente del dominio.
2. Diferentes algoritmos de búsqueda y variantes de los mismos con el objetivo de identificar cuál es el que ofrece una mejor relación entre la calidad de la solución y el tiempo que se tarda en encontrarla.

Para esto, el trabajo se centra concretamente en la planificación mediante búsqueda heurística progresiva. El comportamiento de un proceso de búsqueda progresiva se puede orientar hacia planes de calidad de distintas formas:

- Mediante el uso de un algoritmo de búsqueda determinado.
- Mediante el uso de heurísticas numéricas que sean lo suficientemente precisas en la valoración que realizan de la bondad de los nodos en términos de coste.
- Mediante el uso de técnicas heurísticas para podar el espacio de estados.
- Mediante el uso de técnicas heurísticas que permitan seleccionar primero los sucesores más prometedores de cada nodo.

- Mediante el uso de mecanismos para obtener de forma progresiva soluciones de mayor calidad.

Los objetivos específicos de esta tesis, que conllevan estudiar distintas variaciones de estos métodos para orientar al planificador hacia la obtención de un buen balance calidad/tiempo, son los siguientes:

1. Establecer un marco teórico que unifique la definición de las heurísticas numéricas en planificación basada en costes, de manera que se puedan entender claramente sus similitudes, diferencias y deficiencias.
2. Desarrollar nuevas heurísticas y compararlas con las ya existentes.
3. Explorar la aplicación de distintos algoritmos de búsqueda (incluyendo algoritmos *anytime*, que generan soluciones de calidad incremental), combinados con distintas heurísticas numéricas y distintas variaciones relativas a los métodos que se utilizan para realizar podas y para seleccionar los sucesores más prometedores.
4. Establecer un marco experimental común para comparar el comportamiento de las distintas heurísticas y de los distintos algoritmos de búsqueda en planificación basada en costes.

1.4. Organización de la memoria

Este documento se estructura en cuatro partes. La **Parte I** contiene en el capítulo 1 la presente introducción y en el capítulo 2 una revisión del estado de la cuestión. El estado de la cuestión se divide a su vez en dos partes bien diferenciadas: la planificación clásica (sección 2.3) y la planificación basada en costes (sección 2.4). Para cada uno de estos dos tipos de planificación se introduce el modelo conceptual de planificación al que se refieren y el lenguaje de representación que utilizan. A continuación se hace una revisión de los planificadores existentes y de las técnicas de planificación que utilizan. Finalmente se describen las distintas heurísticas y algoritmos de búsqueda que se han aplicado en planificación mediante búsqueda progresiva en el espacio de estados.

Las **Partes II** y **III** describen el trabajo desarrollado en la tesis doctoral. La **Parte II** engloba el trabajo relacionado con el estudio de heurísticas, que se ha organizado en cinco capítulos. Los tres primeros se refieren a heurísticas numéricas. El capítulo 3 describe una heurística desarrollada durante el trabajo de esta tesis, la heurística h_{level}, y su relación desde el punto de vista procedural con otras heurísticas. El capítulo 4 presenta un estudio teórico orientado a determinar la relación entre las distintas heurísticas basadas en grafos de planificación relajados, concluyendo con una descripción declarativa de cada una de ellas. El capítulo 5 presenta una descripción declarativa unificada que permite definir la mayoría de las heurísticas numéricas que se han desarrollado en planificación mediante búsqueda en el espacio de estados. El último capítulo de esta parte está dedicado al estudio de otras heurísticas adicionales no numéricas.

La **Parte III** engloba el trabajo relacionado con el estudio de distintos algoritmos de búsqueda en planificación basada en costes. Primero se describe el marco de evaluación que se utilizará en los experimentos posteriores. Después, hay un capítulo dedicado a cada algoritmo, que incluye tanto la descripción del algoritmo, como las variaciones del mismo que se sugieren y los resultados experimentales que se obtienen utilizando distintas heurísticas y en relación con los algoritmos de los capítulos anteriores.

Finalmente, la **Parte IV** contiene las conclusiones generales después del desarrollo de la tesis, las aportaciones de la misma y los posibles trabajos futuros.

Capítulo 2

Estado de la cuestión

En este capítulo se revisan las principales técnicas que se han aplicado en Planificación Automática, tanto en planificación clásica como en planificación basada en costes, haciendo especial énfasis en las técnicas que subyacen a la resolución del problema mediante búsqueda heurística. En ambos casos la Planificación Automática se entiende como un proceso general de resolución de problemas.

2.1. Introducción

Para planificar es necesario tener la capacidad de pensar qué se quiere conseguir y de determinar cómo se conseguirá. Planificar requiere contar al menos con las siguientes habilidades:

1. La habilidad de determinar unos **objetivos** y de identificar la **situación actual**.
2. La habilidad de conocer las **acciones** que se pueden realizar y de *predecir* sus efectos.
3. La habilidad de razonar acerca de la forma de organizar o encadenar acciones de manera que se consigan los objetivos. Es decir, de **buscar** un **plan** que conduzca a los objetivos marcados.

Estas habilidades normalmente se consideran *inteligentes*, por lo que se puede afirmar que planificar es una actividad que exhibe *inteligencia*. Por lo tanto, no es de extrañar que la Planificación Automática haya venido siendo y sea uno de los campos de investigación más relevantes de la Inteligencia Artificial.

La mayor parte del trabajo realizado en Planificación Automática se centra en automatizar el punto 3. En cualquier caso, la solución a cualquier problema que se pretende solucionar de manera automática siempre pasa por dotar la máquina de mecanismos para **buscar** esa solución. En el caso de la Planificación Automática, para que esto sea posible, se requiere (Geffner, 2002):

- Determinar el **modelo de planificación** a nivel conceptual. Esto supone definir las características de los problemas a resolver que se van a tener en cuenta, y definir *qué* información hay que representar para modelar el problema dadas estas características. Dependiendo de las propiedades que se vayan a representan y de la información que se utiliza para poder representarlas tendremos distintos modelos de planificación. Sin embargo, puesto que en definitiva se trata de planificar, todos los modelos deben contener (1) información que permita describir o determinar la dinámica del mundo que se representa, es decir, las acciones que se pueden llevar a cabo; (2) información para poder describir total o parcialmente la situación actual; e (3) información para poder describir total o parcialmente los objetivos que se persiguen.

- Elegir un **lenguaje de representación** formal que permita expresar problemas de planificación del modelo definido. Siempre se puede construir software *ad-hoc* para solucionar automáticamente un problema de planificación determinado y en este caso no habría necesidad de diseñar un lenguaje de representación como tal. Sin embargo, desde los inicios de la Inteligencia Artificial se ha venido haciendo especial esfuerzo en el estudio de técnicas generales de resolución de problemas. Las técnicas generales determinan *cómo* se resuelven problemas, y usualmente requieren de lenguajes de representación formales para declarar *cúal* es el problema concreto a resolver. De esta forma la técnica se *independiza* del dominio en el que se resuelven problemas. En Planificación Automática, el lenguaje de representación debe al menos permitir expresar de forma declarativa la información básica para declarar un problema de planificación: la situación actual, las acciones y los objetivos. Esta información normalmente se separa a su vez en dos partes: el *dominio* de planificación y el *problema* de planificación. Las acciones constituyen el dominio de planificación, mientras que la situación actual y los objetivos determinan el problema concreto a resolver en ese dominio. Esta separación permite que se puedan declarar distintos problemas sin necesidad de volver a definir el dominio. Por otro lado, el lenguaje de representación en planificación también puede ser más expresivo, permitiendo declarar información, que sin formar parte de la definición básica del problema, sea útil al proceso general de búsqueda de la solución. Cuando la planificación se aplica a problemas del mundo real, es común que haya mucha información del dominio disponible. Incorporar esta información al modelo puede simplificar la resolución de problemas en ese dominio. Para que el proceso de resolución de problemas siga siendo independiente del dominio hay que definir mecanismos generales para declarar y utilizar esta información adicional.

- Construir el **motor de planificación o planificador**, es decir el algoritmo que resolverá los problemas de planificación expresados en el lenguaje de representación elegido. Desde un punto de vista computacional, encontrar planes con un planificador general es una tarea muy compleja (Backstrom, 1992). Aún en el caso de definir un modelo de planificación básico, como hace la planificación clásica, el problema es intratable (desde el punto de vista teórico es *PSPACE-complete* (Bylander, 1994)). Por este motivo, en general, el objetivo de los planificadores es encontrar una forma de transformar la situación inicial en una situación en la que se cubran los objetivos, pero explorando el mínimo número de alternativas. La generación se planes se puede asimilar al problema de encontrar el camino más corto en un grafo demasiado grande

para ser construido explícitamente. Para resolver este problema se han estudiado mecanismos orientados a construir sólo una parte del grafo que contenga alguna solución y representaciones del problema que faciliten la búsqueda de la solución mediante el uso de alguna técnica existente. En la última década, la habilidad de sintetizar planes ha mejorado notablemente, y hoy en día existen existen planificadores que pueden generar planes con cientos de acciones.

2.2. Modelos de planificación

El mundo real es complejo, por lo que modelar fielmente su comportamiento es prácticamente inviable. Por lo general, en las tareas que requieren de planificación, se puede asumir que existe un *agente* que es capaz de *percibir* lo que ocurre en un determinado *entorno* y de *ejecutar* acciones en el mismo. Las características más importantes de esta interacción (Ghallab et al., 2004), que a la hora de determinar el modelo de planificación se incluirán en éste o no, se muestran en la Figura 2.1 y son:

- Respecto a la percepción, hay que determinar si el modelo asume que las percepciones son perfectas o por el contrario hay incertidumbre en lo que se percibe.
- Respecto a las acciones, hay que determinar si en el modelo:
  - El resultado que se produce al ejecutar las acciones es siempre el mismo y se conoce, es decir es determinista, o por el contrario es no determinista, es decir, hay acciones cuyo resultado es incierto.
  - La ejecución de las acciones es instantánea o por el contrario hay que tener en cuenta que la ejecución de acciones tiene una duración y pueden ocurrir otros eventos mientras una acción se ejecuta.
  - La ejecución de acciones consume/genera recursos o no.
  - La ejecución de acciones tiene coste o no.
- Respecto al entorno, hay que determinar si se asume que:
  - El entorno es totalmente observable o por el contrario es observable parcialmente o no observable.
  - El entorno es estático, salvo la ejecución de las acciones que se modelan, o por el contrario es dinámico, es decir, se pueden producir cambios en él que no se deben a la ejecución de acciones.
- Respecto a los objetivos, hay que determinar si se asume que:
  - El objetivo de la tarea de planificación es conseguir todas las metas que se propongan o por el contrario es suficiente con que se consigan algunas metas (satisfacción parcial).
  - Existen o no restricciones adicionales sobre las acciones del plan o sobre los estados intermedios que se alcanzan.

- El objetivo de la tarea de planificación es conseguir planes de calidad, donde la calidad de un plan se mide en función de una métrica de calidad, o por el contrario la calidad de los planes es indiferente.

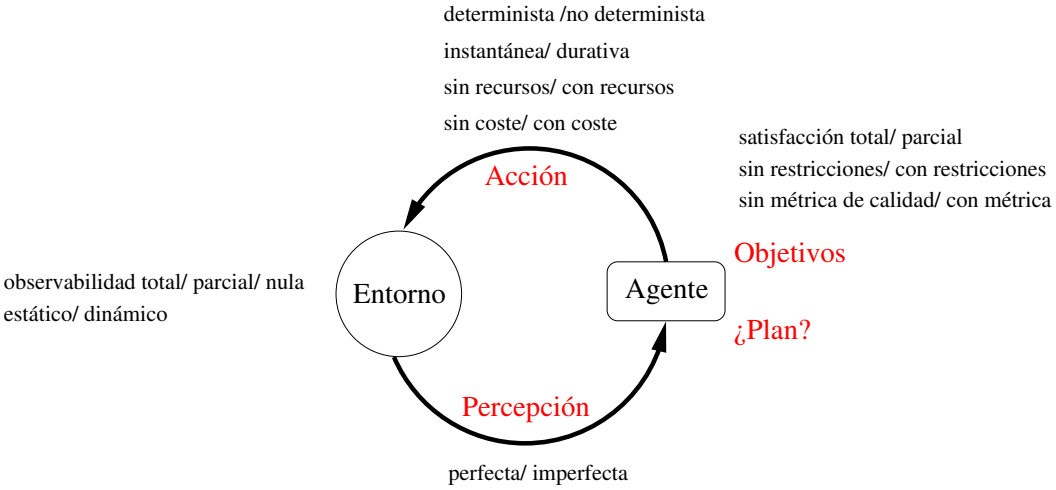

Figura 2.1: Características de los problemas de planificación.

Un modelo de planificación es más sencillo cuantas más simplificaciones se hagan respecto a estas características. El modelo de planificación más simple es el clásico, que consiste en hacer las siguientes suposiciones:

- Respecto a la percepción se asume que las percepciones son perfectas.
- Respecto a las acciones:
  - Las acciones son deterministas, es decir, si una acción se puede aplicar en un estado, su aplicación lleva necesariamente a un único estado. Esto implica que no sea necesario percibir el estado siguiente a la ejecución de las acciones ya que viene determinado de forma única por las acciones que se ejecutan.
  - Las acciones no tienen duración, sino que son transiciones instantáneas entre estados. Por lo tanto no es necesario representar el tiempo explícitamente.
  - Las acciones pueden consumir/generar recursos discretos (en planificación clásica todas las variables de un problema se representan como variables booleanas).
  - Las acciones tienen coste unitario.
- Respecto al entorno:
  - El entorno es completamente observable.
  - El entorno es estático.
- Respecto a los objetivos:

- Se considera que se ha encontrado un plan cuando se han satisfecho todas las metas del problema de planificación.
- No hay una métrica de calidad como tal, aunque normalmente se asume que un plan es mejor que otro si contiene un número de acciones menor.

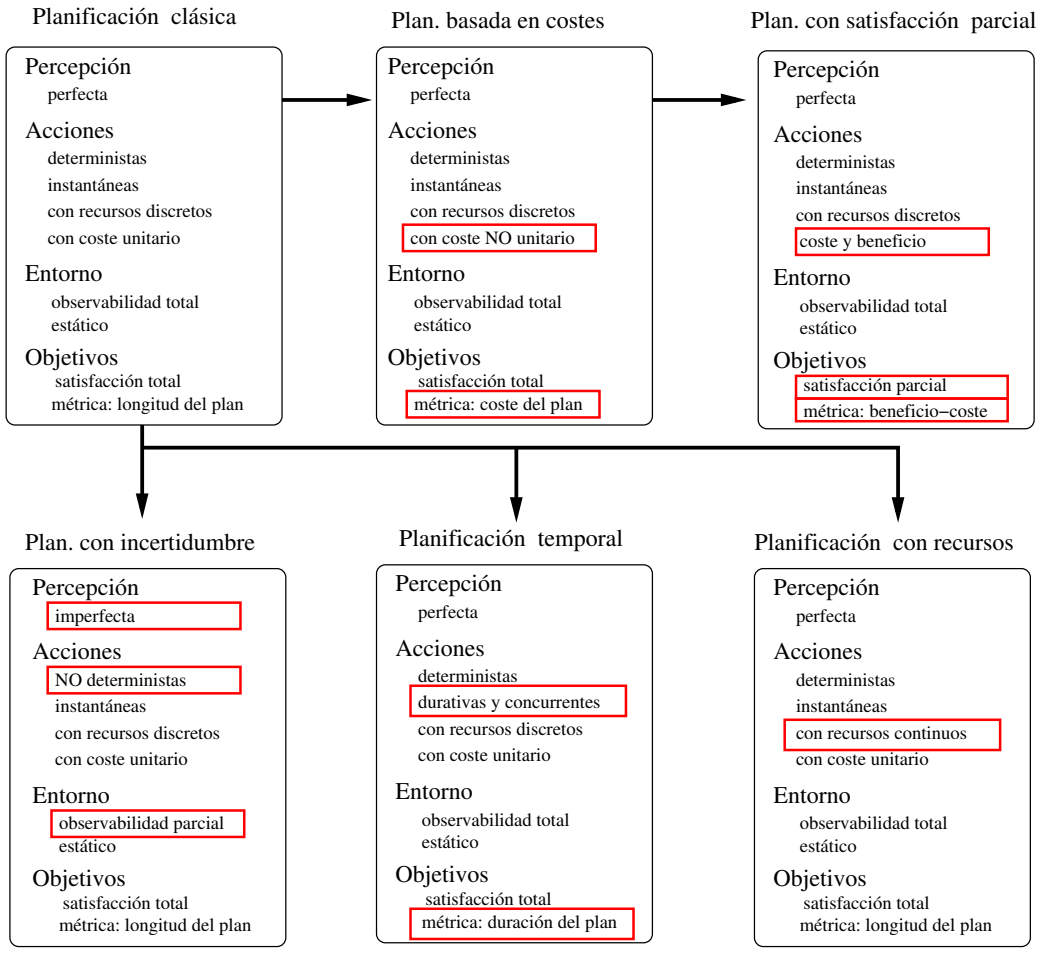

Figura 2.2: Taxonomía de modelos de planificación.

Existen varias formas de extender este modelo clásico, relajando una o varias de las restricciones, para obtener modelos más expresivos y por lo tanto más cercanos al mundo real. La Figura 2.2 muestra una posible taxonomía de los distintos modelos de planificación. En *Planificación Basada en Costes* los costes de las acciones no son uniformes y se consideran mejores los planes cuyo coste total es menor. Adicionalmente, en *Planificación con Satisfacción Parcial*, la satisfacción de metas puede ser parcial, es decir, no es necesario conseguir todas las metas para dar la tarea de planificación por finalizada. En este caso normalmente las acciones suponen un coste y reportan un beneficio. El objetivo de la tarea

de planificación es encontrar un plan que maximice la resta entre el beneficio obtenido y el coste que supone obtenerlo. Desde el punto de vista del lenguaje de representación, en planificación clásica es suficiente con utilizar variables booleanas, mientras que en planificación basada en costes y en planificación con satisfacción parcial se necesitan variables reales para representar costes y beneficios. En *Planificación con Recursos*, se modela el hecho de que las acciones pueden generar y consumir recursos continuos. En este caso, los recursos se representan mediante números enteros o variables que toman valores reales. En *Planificación Temporal* se tienen en cuenta las duraciones de las acciones y la posibilidad de que éstas se ejecuten de forma concurrente. Finalmente, en *Planificación con Incertidumbre* las acciones pueden tener efectos inciertos y además se puede tener un conocimiento incompleto del estado actual, bien porque el sistema es parcialmente observable, o bien porque la percepción es imperfecta. Cuando las acciones tienen efectos estocásticos se habla de *Planificación Probabilística*. Cuando el sistema es parcialmente observable el modelo se denomina de *Planificación Contingente*, mientras que si la observabilidad es nula tendremos un modelo de *Planificación Conformante*. Estos modelos de planificación dan lugar a modelos aún más complejos si se combinan entre sí.

2.3. Planificación clásica

En este apartado se revisan primero los principales lenguajes de representación utilizados en el modelo de planificación clásica. A continuación se introducen las técnicas de búsqueda de la solución que se han aplicado en este tipo de planificación. Finalmente, se presentan los diferentes tipos de heurísticas que se han aplicado para guiar a los planificadores clásicos, haciendo especial énfasis en aquéllas utilizadas por planificadores que hacen búsqueda heurística en el espacio de estados que se deriva directamente del problema de planificación.

2.3.1. Lenguaje de representación

La base del lenguaje de representación en planificación clásica es el lenguaje proposicional del planificador STRIPS (*STandford Research Institute Problem Solver*) (Fikes and Nilsson, 1971). En STRIPS, cada acción a se representa como tres conjuntos de proposiciones, $a = (pre(a), \mathit{eff}(a)^+, \mathit{eff}(a)^-)$: las precondiciones de la acción, la lista de efectos que añade la ejecución de la acción (efectos positivos, también denominados $add(a)$) y la lista de efectos que borra la ejecución de la acción (efectos negativos, también denominados $del(a)$). Dado un estado del mundo, s, una determinada acción, a, se puede ejecutar si la acción es *aplicable* en s; es decir, si sus precondiciones se cumplen en s: $prec(a) \subseteq s$. En este caso, el resultado de aplicar a es:

$$aplicar(s, a) = s \cup \mathit{eff}(a)^+ \setminus \mathit{eff}(a)^-$$

El resultado de ejecutar una secuencia de acciones, $a_1, \ldots, a_m$, en un estado s se define recursivamente mediante:

$$aplicar(s, \{a_1, \ldots, a_m\}) = aplicar(aplicar(s, \{a_1, \ldots, a_{m-1}\}), a_m)$$

En planificación clásica, un problema se define como una tupla $(\mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, donde $\mathcal{P}$ es el conjunto completo de las proposiciones que pueden ser ciertas en un estado, $\mathcal{A}$ es el conjunto de acciones, $\mathcal{I}$ es el estado inicial y $\mathcal{G}$ es un conjunto de proposiciones meta (que constituye una definición parcial del estado final). Dado un problema,$(\mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, una solución al mismo viene dada por un *plan*, que es una secuencia de acciones $\{a_1, \ldots, a_m\} \in \mathcal{A}^*$, cuya ejecución lleva a un estado en el que todas las metas son ciertas: $\mathcal{G} \subseteq aplicar(\mathcal{I}, \{a_1, \ldots, a_m\})$. Si dado un problema existe al menos un plan que lo resuelve, el problema es *resoluble*. Para un planificador que encuentra planes secuenciales, se dice que un plan es *óptimo* si no existe otro plan con menos acciones que resuelva el problema.

Uno de los lenguajes más importantes que extiende las capacidades de representación de STRIPS es ADL (Pednault, 1994) (*Action Description Language*). Las principales extensiones que hace ADL son las siguientes:

- Permite literales positivos y negativos en las precondiciones.
- Permite conjunciones y disyunciones de objetivos. Además, en ADL pueden existir variables cuantificadas en los objetivos.
- Permite efectos condicionales.
- Se puede utilizar el predicado de igualdad.
- Las variables pueden tener tipos.

Hoy en día, el lenguaje considerado como estándar en el mundo de la planificación es PDDL (*Planning Domain Description Language*) (Fox and Long, 2003). PDDL se propuso como lenguaje común para la IPC del año 1998 (AIPS'98 Planning Competition). Actualmente, este lenguaje incluye las características de STRIPS y ADL junto con algunas extensiones que se han ido añadiendo a medida que se han desarrollado nuevas IPCs para cubrir las necesidades de nuevos modelos de planificación. Las últimas versiones son:

- PDDL2.1 (2002): principalmente añade características relativas al manejo del tiempo y funciones objetivo (métricas).
- PDDL2.2 (2004): Añade *derived predicates* y *timed initial literals*. Los primeros son axiomas encadenados hacia atrás que permiten al planificador conseguir hechos haciendo cierto el antecedente de alguno de ellos. Los segundos son literales que se hacen ciertos en un tiempo determinado, independientemente de lo que haga el planificador.
- PDDL3.0 (2006): Añade restricciones y preferencias expresadas en una lógica temporal restringida. Se pueden restringir los estados del plan final y también expresar preferencias sobre el conjunto de metas.
- PDDL3.1 (2008): Añade la posibilidad de expresar los problemas de planificación utilizando un lenguaje funcional (*functional* STRIPS) en lugar de proposicional. De esta forma, para expresar los estados, en lugar de representar que un determinado hecho que afecta a ciertos objetos es cierto, lo que se representa es el valor que toman ciertas funciones que representan propiedades de los objetos.

2.3.2. Técnicas de planificación y planificadores

El planificador STRIPS, que se desarrolló en los años 70, enfocaba la resolución del problema de planificación desde un punto de vista lineal, es decir, asumiendo independencia de metas: no se pasaba a resolver una nueva meta hasta haber encontrado la solución para la meta actual, para una ordenación establecida arbitrariamente. La planificación lineal tiene problemas bien conocidos:

- Es incompleta: en algunos casos, existiendo la solución es incapaz de encontrarla debido a que conseguir una determinada meta imposibilita la consecución de alguna otra. El ejemplo típico es el problema del cohete chino (Veloso and Carbonell, 1993), en el que las metas son que un cohete y un objeto se encuentren en un determinado lugar. El cohete sólo se puede lanzar una vez, así que si se trabaja primero en conseguir que el cohete esté en el lugar, ya no se podrá llevar el objeto.
- Es no óptima: en algunos casos es incapaz de encontrar la solución óptima por mucho tiempo que se utilice. Al tratar las metas de forma independiente, hay ocasiones en que no se puede hacer una gestión óptima de los recursos, por ejemplo, utilizar el mismo recurso para conseguir varias metas. Por otro lado, puede ser necesario deshacer acciones que permitían conseguir una meta para conseguir otra. El ejemplo típico es la anomalía de Sussman[1] (Sussman, 1975).

Los planificadores de hoy en día trabajan simultáneamente en todas las metas del problema (es decir, hacen planificación no lineal) por lo que no tienen estos problemas. Un ejemplo de planificador no lineal es PRODIGY (Veloso et al., 1995). PRODIGY, además de utilizar un lenguaje de representación más expresivo, se ha tomado como base en múltiples investigaciones orientadas a mejorar el comportamiento del planificador mediante el uso de conocimiento generado con aprendizaje automático. Algunos ejemplos son PRODIGY/EBL (Minton, 1988), ANALOGY (Veloso and Carbonell, 1993) y HAMLET (Borrajo and Veloso, 1997).

Actualmente, existe una gran variedad de técnicas para resolver problemas de planificación. Las técnicas se diferencian entre sí en la forma de plantear el problema que hay que resolver. Algunas de las técnicas que se han desarrollado resuelven directamente problemas de planificación. Otras pasan por transformar el problema de planificación en otro tipo de problema y utilizar las técnicas existentes para resolver problemas de ese tipo, como por ejemplo las técnicas SAT o CSP. Las técnicas directas, por lo general, aplican métodos de búsqueda en los que componentes del problema de búsqueda se extráen directamente de la definición del problema de planificación:

- Definen un espacio de estados y un conjunto de operadores que permiten transitar entre estos estados. La semántica de los operadores y de lo que contienen los estados está directamente relacionada con el problema de planificación.
- Plantean un estado inicial y un estado meta (mediante las propiedades que un estado debe cumplir para considerar que es un estado meta).

[1] La anomalía de Sussman es un problema del mundo de los bloques con tres bloques A, B y C. Inicialmente A y B están en la mesa y C está sobre A. Hay dos objetivos: (1) que B esté sobre C y (2) que A esté sobre B.

- Aplican un algoritmo que *busca* los operadores que transforman el estado inicial en un estado meta. Este algoritmo se puede plantear hacer la búsqueda de forma progresiva o hacia delante (del estado inicial a un estado meta); de forma regresiva o hacia atrás (del estado meta al estado inicial); o incluso de forma bidireccional.

Por otro lado, los problemas de planificación se pueden transformar al menos en:

- Problemas de satisfacibilidad, SAT, que se resuelven *buscando* un modelo que haga verdadera cierta fórmula lógica.
- Problemas de restricciones, CSP, que se resuelven *buscando* una asignación de valores a variables de forma que se cumplan todas las restricciones.

Estos problemas también se pueden resolver mediante búsqueda. Sin embargo una vez que el problema de planificación se ha transformado, la semántica de los elementos del proceso de búsqueda pierde su relación con el problema de planificación inicial. Por ejemplo, un posible operador en un problema de restricciones es *asignar valor* a una variable, que no tiene nada que ver con las acciones del problema de planificación inicial.

A continuación se describen las técnicas más comunes. Las técnicas que transforman el problema de planificación en otro tipo de problema se han reunido en un mismo apartado. Las demás son técnicas directas. Cuando los elementos del proceso de búsqueda tienen una semántica directamente relacionada con el problema de planificación diremos que están inducidos por el problema de planificación.

2.3.2.1. Planificación de Orden Parcial (POP)

Los planificadores de orden parcial resuelven directamente el problema de planificación mediante búsqueda. Concretamente, realizan en una búsqueda regresiva en el espacio de planes. Cada nodo de la búsqueda es un plan parcial que se define como una tupla $<\mathcal{A}, \mathcal{O}, \mathcal{L}>$ (Weld, 1994), donde:

- $\mathcal{A}$ es el conjunto de acciones del plan, $\{a_0, \ldots, a_n\}$. Cada acción se corresponde con un operador del dominio, excepto la acción inicial y la acción final. La acción inicial, a_0, es una acción virtual sin precondiciones cuyos efectos consisten en añadir todas las proposiciones del estado inicial del problema. La acción final, a_n, es otra acción virtual que tiene por precondiciones los objetivos del problema y no tiene efectos.
- $\mathcal{O}$ es una ordenación parcial de las acciones del plan. Por lo tanto es un conjunto de restricciones de la forma $a_i \prec a_j$. La acción inicial siempre está ordenada al principio de todas las acciones, mientras que la acción final siempre lo está al final.
- $\mathcal{L}$ es un conjunto de enlaces causales. Un enlace causal, $a_i \xrightarrow{l} a_j$, representa el hecho de que la acción a_i obtiene la precondición l de la acción a_j, y suele tener la semántica asociada de que ninguna acción que aparezca en el plan pudiera ejecutarse entre a_i y a_j elimina el literal l.

El proceso de planificación comienza con un plan que únicamente contiene las acciones inicial y final. En cada iteración se elige una precondición no resuelta (de una acción a_j). Para resolver este subobjetivo habrá que añadir una acción a_i y aparecerá un nuevo enlace causal $a_i \xrightarrow{l} a_j$. Si en un momento determinado del proceso de planificación aparece una acción a_k, de manera que $a_i \prec a_k \prec a_j$ y a_k elimina l, se produce lo que se denomina una *amenaza*, que habrá que resolver añadiendo una restricción de orden, que puede ser:

- De **promoción** ($a_j \prec a_k$): se ordena la acción conflictiva detrás de la segunda acción del enlace causal.
- De **democión** ($a_k \prec a_i$): se ordena la acción conflictiva antes de la primera acción del enlace causal.

Si la eliminación de la amenaza supone la violación de algún orden ya existente, el plan se descarta.

En definitiva, en la planificación de orden parcial se necesitan operadores para:

- Añadir una acción al plan parcial, lo cual modifica las acciones del plan parcial, el orden parcial y los enlaces causales.
- Eliminar amenazas, lo que modifica el orden parcial.

El estado inicial de la búsqueda es un plan vacío (que contiene únicamente las dos acciones virtuales), mientras que el estado final es un plan parcialmente ordenado y sin amenazas. Así, en contraste con la planificación de orden total que genera planes totalmente ordenados o secuenciales, la planificación de orden parcial genera planes parcialmente ordenados. Los planificadores de orden parcial se pueden utilizar para generar planes paralelos (Knoblock, 1994).

La investigación en planificadores de orden parcial tuvo su auge a principios de los años 90, con los planificadores SNLP (McAllester and Rosenblitt, 1991) y UCPOP (Penberthy and Weld, 1992), aunque después se abandonó debido a que estos planificadores sólo eran capaces de resolver problemas que requerían muy pocas acciones (Smith et al., 2000). Sin embargo, a partir del año 2000 se han aplicado nuevas técnicas, como el uso de heurísticas y el análisis de alcanzabilidad, que mejoran notablemente la eficiencia de los algoritmos POP. Ejemplos de planificadores que incorporan estas mejoras son REPOP (Nguyen and Kambhampati, 2001) y VHPOP (Younes and Simmons, 2003).

2.3.2.2. Grafos de planificación

La planificación basada en grafos de planificación surgió en 1995 con GRAPHPLAN (Blum and Furst, 1995). GRAPHPLAN construye una representación compacta del espacio de estados que induce el problema de planificación, un grafo de planificación, y después hace búsqueda sobre ella. El grafo de planificación constituye una representación compacta del espacio de estados porque considera como un único *nodo* todos los estados que se generarían en el mismo nivel haciendo una búsqueda hacia delante.

El grafo de planificación tiene dos tipos de nodos que forman niveles consecutivos: los nodos de proposición y los nodos de acción. Los nodos de proposición contienen todos los efectos de las acciones del nivel de acciones previo. Los nodos de acción contienen todas las acciones cuyas precondiciones están presentes en el nivel de proposiciones previo. El primer nivel de proposiciones contiene todas aquellas proposiciones que son ciertas en el estado inicial. GRAPHPLAN trabaja con todas las acciones instanciadas, por lo que requiere que las acciones se instancien en una etapa de preproceso.

El grafo de planificación contiene distintos tipos de información importante:

- La colección de proposiciones que es cierta tras la aplicación de un número de acciones incremental.
- Los pares de proposiciones y acciones que son mutuamente exclusivas (*mutex*). La propagación de *mutex* comienza en el primer nivel, marcando las acciones que interfieren estáticamente entre si (es decir, sus precondiciones y efectos son inconsistentes). Estos *mutex* se propagan hacia delante usando las dos reglas siguientes: (1) dos proposiciones del nivel k son *mutex* si todas las acciones del nivel $k-1$ que generan una de las proposiciones son *mutex* entre sí con todas las acciones que generan la otra proposición. (2) Dos acciones del nivel k son *mutex* sí interfieren estáticamente entre si, o si alguna de las precondiciones de una acción es *mutex* con alguna de las precondiciones de la otra acción.

En el momento en que todas las metas del problema de planificación aparecen en el grafo y son compatibles, finaliza la fase de construcción del grafo de planificación y comienza la fase de búsqueda de la solución. GRAPHPLAN realiza un proceso de búsqueda hacia atrás para encontrar un plan solución. Aunque se utilizan algunas técnicas para mejorar la eficiencia (como las *memoizations*), esta búsqueda es básicamente una búsqueda hacia atrás en profundidad, que se lleva a cabo de forma iterativa. Para cada meta en el instante t elegida en orden arbitrario, se selecciona alguna acción en el instante $t-1$ que consiga la meta y no sea *mutex* con ninguna acción ya seleccionada. Después, se continúa recursivamente con las siguiente meta del instante t. Si una meta pendiente se consigue (*accidentalmente*) con acciones seleccionadas previamente, no se selecciona una nueva acción para conseguirla. Una vez se han solucionado todas las metas en el instante t, las precondiciones de las acciones seleccionadas conforman el conjunto de metas para $t-1$, y así sucesivamente hasta que todas las metas pendientes son hechos del estado inicial. Si con este proceso no se encuentra ningún estado solución, el grafo se extiende y comienza una nueva búsqueda.

2.3.2.3. Planificación con búsqueda heurística

El éxito de GRAPHPLAN provocó un crecimiento del interés en las técnicas de Planificación Automática, que motivó la exploración de nuevas direcciones. Quizás, la más relevante sea la planificación mediante *búsqueda heurística*. Aunque cualquier proceso de búsqueda que aplique heurísticas se puede denominar búsqueda heurística, cuando se habla de búsqueda heurística en planificación, normalmente uno se está refiriendo a procesos de búsqueda en el espacio de estados y con los operadores que induce el propio problema de planificación. Es decir, los operadores del proceso de búsqueda se corresponden con las acciones

instanciadas del modelo de planificación y cada estado del espacio de estados contiene el estado del mundo que se modela en ese momento.

En planificación clásica con lenguaje STRIPS este estado se representa mediante el conjunto de proposiciones que son ciertas, y el estado meta es cualquier estado en que las proposiciones meta sean ciertas. Partiendo de esta representación, se puede aplicar cualquiera de los algoritmos de búsqueda existentes. Sin embargo, los algoritmos de fuerza bruta son incapaces de resolver problemas de complejidad media/alta y es necesario aplicar heurísticas para guiar al algoritmo.

La planificación con búsqueda heurística se inició con los trabajos de McDermott en el planificador UNPOP (McDermott, 1996; Mcdermott, 1991) y Bonet y Geffner en HSP (Bonet and Geffner, 2001). El uso de heurísticas no es nuevo en el mundo de la planificación, ya que algunos planificadores anteriores como UCPOP (Penberthy and Weld, 1992) permitían el uso de funciones heurísticas para guiar la búsqueda, aunque en general estas heurísticas se codificaban a mano y era relativamente complicado conseguir una función heurística que guiara al planificador hacia las mejores elecciones. En los trabajos mencionados, se demostró que se podían conseguir heurísticas bastante informativas de forma *automática*. A partir de este momento la búsqueda heurística ha sido una de las técnicas de planificación con más éxito.

Puesto que este tipo de planificación es el más relevante para esta tesis, más adelante, en las secciones 2.3.3 y 2.3.4 se describirán con detalle tanto las heurísticas como los algoritmos que se han venido aplicando.

2.3.2.4. Transformación del problema de planificación

Algunos investigaciones han explorado direcciones que consisten en reformular el problema de planificación como un problema de otro tipo, de manera que se puedan aplicar técnicas ya existentes para resolver el problema reformulado. Existen al menos tres aproximaciones: planificación SAT, planificación como CSP (*Constraint Satisfaction Problem*), y planificación como *model-checking*.

- **Planificación** SAT
  En el caso de la planificación SAT los problemas de planificación se transforman en problemas de *satisfacibilidad* que se tratan de resolver utilizando un resolvedor de problemas SAT (Kautz and Selman, 1992). Un problema de satisfacibilidad consiste principalmente en determinar si existe una asignación de valores de verdad a las variables de una fórmula proposicional (en forma normal conjuntiva, CNF) de manera que la fórmula se haga cierta. Los problemas de satisfacibilidad son *NP-completos* desde el punto de vista teórico. Para aplicar esta idea en planificación hay que determinar una fórmula lógica, de manera que el problema de planificación sea resoluble si y sólo si la fórmula es satisfacible. Además, cada modelo de la fórmula se debe corresponder con un posible plan solución del problema. La principal limitación es que con una instancia SAT sólo se puede comprobar la existencia de un plan de una determinada longitud, de manera que si es imposible satisfacer la fórmula, se debe construir una

nueva fórmula para intentar encontrar un plan más largo. Así, un problema de planificación se puede traducir en una secuencia de instancias SAT. El primer planificador que implementó una aproximación SAT fue SATPLAN (Kautz and Selman, 1992).

- **Planificación** CSP
  La aproximación como CSP comenzó con el trabajo de Van Beek (Beek and Chen, 1999), en el que se hacía a mano la traducción de los problemas de planificación a problemas CSP. Recientemente, Do y Kambhampati (Do and Kambhampati, 2000) han mostrado que la traducción se puede hacer de forma automática utilizando un grafo de planificación como estructura intermedia. Un CSP consiste en una colección de variables asociadas a sus posibles dominios de valores, y una colección de restricciones que especifican cómo interactúan los valores de las variables. Un resolvedor de CSPs debe buscar asignaciones de valores a las variables que sean consistentes con las restricciones.

  La codificación de un problema de planificación en un problema de CSP requiere que se defina previamente, como en el caso de SAT, la longitud n de la solución que se busca. Además, se requiere que el problema de planificación esté representado mediante variables de estado, en lenguaje SAS$^+$ (Sandewall and Rönnquist, 1986; Bäckström and Nebel, 1995)[2]. En lenguaje SAS$^+$, una tarea de planificación se define como una tupla $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ en la que:

  - $\mathcal{V}$ es un conjunto finito de variables de estado con dominio $\mathcal{D}_v$. El dominio define implícitamente el dominio extendido $\mathcal{D}_v^+ = \mathcal{D}_v \cup \{u\}$, donde u denota un valor indefinido (de ahí el símbolo $^+$ de SAS$^+$). Un hecho es un par (v, d) y representa que la variable $v \in \mathcal{V}$ toma el valor $d \in D_v$. Cuando un hecho pertenece al estado s, se representa como $(v, d) \in s$ ó $s(v) = d$. Un estado es una asignación de valores a variables de $\mathcal{V}$.
  - $\mathcal{O}$ es un conjunto de operadores, donde cada operador es un par $(\mathit{pre}, \mathit{eff})$ de conjuntos de hechos (asignaciones de valores a variables).
  - s_0 es el estado inicial, y
  - $s_\star$ es una asignación parcial de valores a variables (las metas).

  Un operador $o = (\mathit{pre}, \mathit{eff})$ es aplicable en un estado s si $\mathit{pre} \subseteq s$. En este caso, su aplicación produce el estado s' con $s'(v) = \mathit{eff}(v)$ si v está en los efectos del operador y $s'(v) = s(v)$ en otro caso. Una secuencia de operadores π es un plan si $s_\star \subseteq s_0[\pi]$, donde $s_0[\pi]$ es el estado que se consigue aplicando la secuencia de operadores al estado inicial.

  Las codificaciones SAT y CSP son muy similares. Las principales diferencias están relacionadas con la representación. En CSP, al utilizar variables de estado, la codificación es más sencilla y no se requieren los axiomas de exclusión. También es más sencillo representar variables de estado con valores reales ya que no es necesario discretizarlos, como habría que hacer en SAT.

[2] SAS es un acrónimo de *Simplified Action Structures*, para denotar que es una simplificación del formalismo de *action structures* (Sandewall and Rönnquist, 1986).

- **Planificación con** *model checking*
  La aproximación con *model-checking* (Edelkamp and Helmert, 2000; Edelkamp, 2003) es en realidad una variación de la aproximación SAT. La idea es reformular el problema de planificación como una fórmula lógica que pueda ser expresada como un Diagrama de Decisión Binario (*BDD*), un tipo de grafo para representar funciones boolenas, y entonces utilizar técnicas de *model-checking* para encontrar un modelo para esa fórmula.

2.3.2.5. Planificación híbrida

La planificación híbrida surge debido a que los resultados experimentales demuestran que no existe ningún planificador cuya estrategia sea adecuada para todos los problemas de planificación. En este tipo de planificación se combinan varias técnicas. Ejemplos de planificadores híbridos son:

- El planificador MIPS (Edelkamp and Helmert, 2000) combina GRAPHPLAN con *model checking*. Utiliza primero una estrategia GRAPHPLAN y si en un tiempo determinado no se encuentra solución, el problema se reformula como un problema de *model checking*.
- El planificador BLACKBOX (Kautz and Selman, 1999) combina GRAPHPLAN con planificación SAT.
- El planificador STAN (Fox and Long, 2001) combina GRAPHPLAN con planificación heurística con búsqueda hacia delante, utilizando técnicas de análisis de dominio, como TIM (Fox and Long, 1998), para seleccionar automáticamente cuál de estas dos estrategias utilizar.

2.3.2.6. Planificación jerárquica

La planificación jerárquica más conocida es la que se denomina HTN (*Hierachical Task Network*). Este tipo de planificación está a medio camino entre la planificación independiente del dominio y la planificación dependiente del dominio (D. Nau lo denomina *planificación de dominio configurable* (Nau, 2007)). El planificador en sí es normalmente independiente del dominio, pero sus entradas contienen información adicional para restringir la búsqueda.

La planificación jerárquica supone una visión de la planificación orientada a las tareas. El dominio de un problema de planificación jerárquica expresa cómo se descomponen las tareas que hay que realizar. Puesto que el dominio ya expresa cómo se descomponen las tareas, el planificador se ahorra parte del proceso de buscar cómo hacerlo. Esto hace que con planificación jerárquica sea posible desde un punto de vista computacional resolver problemas de planificación más complejos y cercanos al mundo real. El coste adicional es que requiere mucho más esfuerzo humano a la hora de construir el dominio de planificación que la planificación clásica con lenguaje STRIPS. Aunque no es usual clasificar la planificación HTN como planificación clásica, se puede plantear una planificación jerárquica para el modelo de planificación clásica tal y como se ha definido en esta tesis.

Existen dos diferencias principales entre la planificación HTN y la planificación clásica con lenguaje STRIPS:

- En el dominio, no sólo se representan acciones *primitivas* (como las que se venían representando hasta ahora), sino que además se representan acciones *compuestas*, que son acciones abstractas que requieren de la ejecución de un conjunto de acciones (primitivas o compuestas) para llevarse a cabo.
- El tipo de metas que se pretende conseguir con la planificación varía. El objetivo es llevar a cabo una serie de tareas en lugar de conseguir un determinado conjunto de proposiciones en el estado meta.

En general, los algoritmos de planificación HTN toman como entradas un estado inicial, un conjunto de tareas inicial (metas) y un conjunto de métodos de descomposición (dominio). La búsqueda consiste en elegir los métodos apropiados a expandir para llegar a una red de tareas que únicamente esté compuesta por acciones primitivas (un plan). Algunos planificadores jerárquicos relevantes son O-PLAN (Currie and Tate, 1991), SHOP2 (Nau et al., 2003) y SIADEX (Castillo et al., 2006).

2.3.3. Heurísticas en planificación clásica

En planificación mediante búsqueda heurística se han aplicado heurísticas de muy diversos tipos y naturalezas. A grandes rasgos las heurísticas se pueden dividir en dos categorías: heurísticas dependientes del dominio y heurísticas independientes del dominio. Con heurísticas independientes del dominio nos referimos a aquéllas cuyo proceso de cálculo es el mismo para todos los dominios. Las heurísticas dependientes del dominio, pueden ser muy útiles para guiar la búsqueda, sin embargo, no se pueden aplicar a problemas con estructura distinta a aquellos para los que se generaron. Estas heurísticas requieren de un trabajo adicional en cada dominio para generar la información (o la función) que permite calcular la heurística. Cuando este trabajo se puede realizar de forma automática, se acercan más a las heurísticas independientes del dominio, que son aquéllas que no requieren de ninguna información adicional al propio dominio/problema. Estas últimas están acordes con la idea de la planificación como un proceso *general* de resolución de problemas y han tenido un gran auge en los últimos años.

Las heurísticas independientes del dominio más comunes en planificación mediante búsqueda heurística se pueden clasificar en función de su objetivo como:

1. Heurísticas numéricas: aquéllas que estiman numéricamente la *distancia* entre dos estados: un estado fuente y un *estado destino*.
2. Heurísticas de poda/ordenación: aquéllas que sirven para podar el espacio de estados, o para ordenar ciertos nodos.
3. Heurísticas *lookahead*: aquéllas que permiten generar rápidamente estados *profundos* mediante la aplicación (total o parcial) de una política.

A continuación se explican con detalle las heurísticas de cada uno de estos tipos.

2.3.3.1. Heurísticas numéricas

Normalmente, en búsqueda heurística, las heurísticas numéricas se utilizan para estimar la *distancia* entre un estado fuente y un *estado destino*. El estado fuente es el estado actual en la búsqueda y estado destino es el estado meta. En planificación con búsqueda heurística hacia delante, el estado destino se describe parcialmente, como un estado que contiene las metas del problema. Cuando se realiza búsqueda hacia atrás, el estado destino es el estado inicial del proceso de planificación, y es el estado inicial del proceso de búsqueda lo que se describe parcialmente mediante las metas del problema. La distantica se mide en términos del número de acciones que habría que ejecutar secuencialmente para conseguir un estado que contenga las metas del problema a partir del estado fuente. De esta forma, utilizando la heurística con un algoritmo de búsqueda adecuado se podrá guiar la búsqueda hacia *buenas* soluciones, donde se considera que un plan es mejor que otro si está formado por menos acciones (es decir, la longitud del plan es menor).

En planificación, el tipo de relajación más común para calcular heurísticas consiste en eliminar parcial o totalmente las listas de borrados de las acciones. Teniendo en cuenta esto y dependiendo de la técnica que se utiliza para calcular la heurística, éstas se pueden clasificar en:

- Las que estiman el coste de un conjunto de proposiciones como el coste de algunas proposiciones del conjunto o como una combinación de los costes de las proposiciones del conjunto. Dentro de este tipo de heurísticas estarían las heurísticas de HSP (Bonet and Geffner, 2001) y la familia de heurísticas h^m (Haslum and Geffner, 2000).
- Las que utilizan explícitamente un grafo de planificación para extraer la heurística. Dentro de éstas hay dos grupos:
  - Las que extraen la heurística directamente del grafo de planificación (Nguyen and Kambhampati, 2000), utilizando como coste de cada proposición la primera capa en la que ésta aparece . Un ejemplo sería la heurística implícita en GRAPHPLAN. En (Bryce and Kambhampati, 2006) este tipo de heurísticas, que se denominan *heurísticas basadas en niveles*.
  - Las que a partir del grafo de planificación extraen un plan relajado y a partir de éste obtienen la heurística, como hace el planificador FF (Hoffmann and Nebel, 2001). En (Bryce and Kambhampati, 2006) este tipo de heurísticas se denominan *heurísticas del plan relajado*.
- Las que realizan algún tipo de análisis de dominio para extraer la heurística, como la heurística del grafo causal (Helmert, 2004).
- Heurísticas que ignoran determinados hechos como son las que se obtienen a partir de bases de datos de patrones (*Pattern Data Bases, PDBs*) (Edelkamp, 2001).

Las heurísticas numéricas se pueden calcular hacia delante (aplicando directamente los operadores del dominio) o hacia atrás (aplicando los operadores del dominio invertidos). Existen casos de los dos tipos. Por ejemplo los planificadores FF (Hoffmann and Nebel,

2001), HSP (Bonet and Geffner, 2001), HSPr (Bonet and Geffner, 2001) y ALTALT (Nigenda et al., 2000) calculan la heurística hacia delante (aunque en los dos últimos casos el proceso de búsqueda se realiza hacia atrás). Sin embargo, en GRT (Refanidis and Vlahavas, 2001) tanto el cálculo de la heurística como el proceso de búsqueda se realizan hacia atrás. Hacer búsqueda hacia atrás tiene el problema de que, como las metas expresan un estado incompleto, se pueden generar estados no válidos o hechos que no se pueden conseguir. Sin embargo, hacerlo hacia delante supone que hay que calcular la heurística cada vez que se quiere evaluar un estado, con el incremento de tiempo que esto supone. El uso de búsqueda hacia atrás permite calcular los valores de la heurística en una etapa de preproceso.

Dos propiedades relacionadas con las heurísticas numéricas son la *consistencia* y la *admisibilidad*. Se dice que una heurística es consistente si para cada nodo n del árbol de búsqueda se cumple que el coste estimado de alcanzar el objetivo desde ese nodo no es mayor que el coste estimado de alcanzar el objetivo desde un sucesor n' más el coste de conseguir el sucesor. Es decir si $\forall n, n'$ tal que $n' \in s(n)$ se cumple que $h(n) \leq h(n') + c(n, n')$.

Por otro lado, una heurística es admisible si sus estimaciones nunca sobrestiman el coste real óptimo para llegar a un estado final partiendo del estado que se está estimando, es decir $\forall n, h(n) \leq h^*(n)$. Todas las heurísticas consistentes son a la vez admisibles. Sin embargo, no todas las heurísticas admisibles son consistentes.

La admisibilidad es importante porque existen algoritmos de búsqueda (como A^* e IDA^*) que son admisibles, es decir, que garantizan encontrar la solución óptima siempre y cuando utilicen una heurística admisible. En planificación independiente del dominio se suele sacrificar la admisibilidad principalmente por dos motivos:

1. Las heurísticas admisibles que se han conseguido hasta el momento, excepto las basadas en un uso extensivo de memoria, no parecen ser lo suficientemente informadas, y
2. Los algoritmos de búsqueda que garantizan la optimalidad suelen ser caros computacionalmente cuando se utilizan con heurísticas poco informadas.

Heurísticas de HSP

HSP (Bonet and Geffner, 2001) es un planificador que hace búsqueda heurística en el espacio de estados que induce el problema de planificación. La búsqueda comienza en el estado inicial $\mathcal{I}$, con una función heurística que se deriva de la representación STRIPS del problema y esta inspirada en las ideas de (McDermott, 1996) y (Bonet et al., 1997).

Para resolver un problema, P, la heurística se obtiene considerando un *problema relajado*, P', en el que se ignoran todas las listas de borrados. El coste de la solución óptima, $h'(s)$, del problema relajado, P', en cualquier estado es un límite inferior del coste óptimo, $h^*(s)$, del problema original, P. Por este motivo, la función heurística, $h'(s)$, es una heurística *admisible* para el problema original. Sin embargo, resolver de forma óptima el problema relajado es computacionalmente muy caro. Desde el punto de vista teórico es *NP-hard* (Bylander, 1994). Lo que se propone en (Bonet and Geffner, 2001) es utilizar una

estimación del coste óptimo del problema relajado. Para calcular esta estimación se obtiene primero el coste aproximado para cada una de las metas de manera independiente, y después se estima $h'(s)$ como una combinación de estos costes.

El coste de conseguir un átomo, p, partiendo de un estado, s, es:

$$h(p,s) = \begin{cases} 0 & \text{si } p \in s \\ \min\limits_{a \in A(p)} \{1 + h(pre(a), s)\} & \text{en otro caso} \end{cases}$$

donde $A(p)$ es el conjunto de acciones que añaden p, y $h(pre(a), s)$ es el coste estimado de conseguir las precondiciones de la acción a desde s.

En HSP (Bonet and Geffner, 2001), para calcular la función h se utiliza un procedimiento simple de encadenamiento hacia delante en el que las medidas $h(p, s)$ se inicializan a 0 si $p \in s$ y a ∞ en otro caso. Cada vez que un operador es aplicable en s, se añaden a s todas las proposiciones $p \in add(a)$ y se actualiza $h(p, s)$ de la forma:

$$h(p,s) = \text{mín}\{h(p,s), 1 + h(pre(a), s)\}$$

Las actualizaciones continúan hasta que las medidas $h(p, s)$ no cambian.

La expresión $h(pre(a), s)$ representa el coste estimado del conjunto de proposiciones dado por las precondiciones de la acción a. En general, el coste del conjunto de proposiciones C, $h(C, s)$, se puede definir en términos de las proposiciones del conjunto de varias maneras: como la *suma* de los costes de las proposiciones del conjunto, de esta forma se obtiene la **heurística aditiva (h_{add})**; o como el *máximo* de los costes de las proposiciones del conjunto, con lo que se obtiene la **heurística del máximo (h_{max})**.

Así en el caso de h_{add}:

$$h_{add}(C,s) = \sum_{p \in C} h_{add}(p,s)$$

y en el caso de h_{max}:

$$h_{max}(C,s) = \max_{p \in C} \{h_{max}(p,s)\}$$

Dado un estado a evaluar s, y un conjunto de metas $\mathcal{G}$, la heurística, en cada caso, se calcula como $h(\mathcal{G}, s)$.

La heurística h_{add} asume que todas las (sub)metas son independientes; es decir, que todas las acciones para conseguirlas se ejecutan secuencialmente. En general esto no es cierto porque se puede dar el caso de que conseguir algunas (sub)metas facilite o dificulte la consecución de otra. Es decir de que se produzcan interacciones positivas o negativas entre (sub)metas. Por este motivo, la *heurística aditiva* no es *admisible* cuando hay interacciones positivas entre metas (es decir, conseguir alguna submeta ayuda a conseguir otras) y es poco informada (subestima considerablemente) cuando hay interacciones negativas entre metas (es decir, conseguir una submeta borra otras (sub)metas).

La heurística h_{max} considera que las acciones para conseguir las (sub)metas se ejecutan en paralelo. Así, sólo tiene en cuenta la submeta más costosa. h_{max} es *admisible* puesto que el coste de conseguir un conjunto de proposiciones (las metas) no puede ser menor que el coste de conseguir exclusivamente una meta del conjunto considerando el problema relajado. Por otro lado, h_{max}, es normalmente poco informada ya que se centra sólo en la meta más difícil de conseguir ignorando las demás.

Desde el punto de vista intuitivo, h_{add} es más informada que h_{max} ya que tiene en cuenta todas las (sub)metas. Desde el punto de vista experimental, según los resultados de HSPr (Bonet and Geffner, 2001), h_{add} funciona considerablemente mejor, resolviendo más problemas, en dominios que implican muchas (sub)metas que se consiguen de forma independiente (como *Logistic* o *Gripper*). Sin embargo h_{max} produce mejores resultados en algunos dominios que implican interacciones complejas entre (sub)metas (como *Hanoi* o *Tireworld*). En dominios en los que en cierto grado el problema se puede descomponer (tipo *Blocksworld*) ambas heurísticas resuelven aproximadamente el mismo número de problemas, pero h_{max} utiliza más tiempo.

Las heurísticas descritas en este apartado se pueden calcular utilizando distintos algoritmos. Estos algoritmos se han estudiado en el trabajo de Liu, Koening y Furcy (Liu et al., 2002). El estudio se realizó para el caso de planificación STRIPS pero también es válido para el caso de planificación basada en costes. Los algoritmos estudiados calculan el coste de cada proposición agregando los costes de las precondiciones con la fórmula correspondiente de $h_{max}(p, s)$ o $h_{add}(p, s)$. Una vez se ha calculado el coste de cada proposición, el valor de la heurística se extrae utilizando la fórmula correspondiente para $h_{max}(\mathcal{G}, s)$ o $h_{add}(\mathcal{G}, s)$ respectivamente. Estos algoritmos son: un algoritmo que implementa una forma de *Value Iteration*, el algoritmo *Bellman-Ford generalizado*, y el algoritmo *Dijkstra generalizado*.

La idea de *Value Iteration* es utilizar variables para representar el coste de las proposiciones y variables para representar el coste de las acciones. Las variables que representan proposiciones en el estado inicial se inicializan a cero, mientras que todas las demás variables se inicializan a infinito. Una vez inicializadas, se realiza un bucle. En cada iteración se actualizan todas las variables utilizando la fórmula correspondiente. Es decir, las variables que representan acciones se actualizan sumando el coste de la acción y el coste resultado de agregar los costes de sus precondiciones en la iteración actual. Las variables que representan proposiciones se actualizan con el valor mínimo en la iteración actual de las variables de acciones que las generan. El algoritmo continúa hasta que no hay cambios en los valores.

El algoritmo *Bellman-Ford generalizado* también itera sobre todas las variables que representan acciones, pero se diferencia de *Value Iteration* en que sólo itera sobre aquellas variables de proposiciones cuyo valor ha cambiado.

Finalmente, el algoritmo *Dijkstra generalizado* mantiene una cola ordenada para iterar sobre las variables. Itera primero sobre aquellas variables con menor valor. La cola contiene sólo las variables cuyo valor ha cambiado. Al contrario que los dos algoritmos previos, *Dijkstra generalizado* termina inmediatamente cuando los valores de las variables que representan las proposiciones de las metas son correctos.

El algoritmo de *Value Iteration* explicado y una variante del algoritmo *Bellmand-Ford*

generalizado se han utilizado en los planificadores STRIPS HSP 2.0 y HSP 1.0, respectivamente, para calcular la heurística. En el trabajo citado también se propone la idea de calcular las heurísticas de forma incremental, lo que permite aprovechar en el estado sucesor parte del cálculo realizado para calcular la heurística en el estado previo.

Familia de heurísticas h^m

La idea de la familia de heurísticas h^m (Haslum and Geffner, 2000) es aproximar el coste de un conjunto de proposiciones C mediante el coste del subconjunto más costoso de tamaño m (es decir, con m elementos). Éste es el tipo de relajación que se aplica para calcular todas las heurísticas de esta familia. Para un conjunto de proposiciones C, la familia de heurísticas h^m se caracteriza mediante la siguiente fórmula:

$$h^m(C,s) = \begin{cases} 0 & \text{si } C \subseteq s \\ \min\limits_{<B,a>\in R(C)} \{coste(a) + h^m(B,s)\} & \text{si } |C| \leq m \\ \max\limits_{D\subset C, |D|=m} h^m(D,s) & \text{en otro caso} \end{cases}$$

Es decir, si el conjunto de proposiciones pertenece al estado que se está evaluando su evaluación heurística es cero. Si el conjunto de proposiciones tiene más de m proposiciones, su evaluación heurística es el máximo de las evaluaciones heurísticas de todos los subconjuntos D con exactamente m proposiciones. Para un conjunto de proposiciones con tamaño menor o igual que m, la evaluación heurística es el mínimo de todas las evaluaciones de los conjuntos de proposiciones B que se pueden conseguir haciendo regresión sobre C con todas las acciones que permiten hacer regresión sobre C, más el coste de la acción que se haya utilizado en cada caso (en planificación clásica $coste(a) = 1$ para todas las acciones). Una acción a permite hacer regresión sobre un conjunto de proposiciones C si $C \cap add(a) \neq \emptyset$ y $C \cap del(a) = \emptyset$; es decir, si es una acción que añade algún átomo al conjunto y no borra ningún átomo del conjunto. Por ejemplo, en el caso de $C = \{p, q\}$ (y $m = 2$), las acciones que permiten hacer regresión sobre C son todas aquéllas que (1) añaden p y q, (2) añaden p y no borran q, y (3) añaden q y no borran p. El conjunto de proposiciones que se obtiene haciendo la regresión es $B = C - add(a) + pre(a)$. En el caso (1) $B = pre(a)$; en el caso (2) $B = pre(a) \cup \{q\}$; y en el caso (3) $B = pre(a) \cup \{p\}$. En la fórmula, $R(C)$ es el conjunto de pares $< B, a >$ que se obtienen haciendo regresión sobre el conjunto de proposiciones C.

Todas las heurísticas de esta familia son admisibles, si bien, cuanto mayor es el parámetro m más precisa es la heurística obtenida y también más caro su cálculo computacionalmente hablando. Para un valor fijo de m, el cálculo de la heurística h^m es polinomial en N^m, donde N es el número de proposiciones. Desde el punto de vista de su implementación, se pueden calcular utilizando algoritmos para encontrar el camino más corto en un grafo, ya que como se explica en (Haslum and Geffner, 2000) el cálculo de la heurística es isomorfo a un problema de este tipo relajado.

En el caso particular de $m = 1$, se consigue una heurística denominada heurística *max-atom* que coincide exactamente con h_{max}. En caso de $m = 2$, la heurística se denomina

max-pair, y está muy relacionada con la consideración de pares de mutex que se hace en el cálculo de la heurística implícita en GRAPHPLAN, h_G (Haslum and Geffner, 2000). Sin embargo, mientras que la heurística *max-pair* está definida para planificación secuencial con costes de acciones arbitrarios, la de GRAPHPLAN se define para planificación paralela con costes de acciones unitarios (también existe una definición de la heurística *max-pair* para el caso de planificación paralela, que es equivalente a h_G).

Heurística implícita en GRAPHPLAN

GRAPHPLAN se puede entender también como un planificador heurístico (Haslum and Geffner, 2000), que utiliza una determinada función heurística, h_G, y que realiza un determinado tipo de búsqueda. La estimación heurística del número de acciones que se necesitan para conseguir un conjunto de proposiciones C a partir de un estado, $h_G(C)$, viene dada por el índice de la primera capa en el grafo de planificación que contiene todos las proposiciones de C sin *mutex* (esta heurística fue denominada *set-level heuristic*) (Nguyen and Kambhampati, 2000). h_G se calcula en el proceso de expansión del grafo de planificación. El proceso de búsqueda hacia atrás de la solución en el grafo de planificación se puede ver como la aplicación de una versión del algoritmo de búsqueda IDA* (Korf, 1985), en la que se utiliza la suma del coste acumulado y el coste estimado, $h_G(n)$, para podar los nodos n cuyo coste excede el umbral actual (en realidad GRAPHPLAN no genera estos nodos).

Cuando se expande GRAPHPLAN sobre el problema relajado, el grafo de planificación que se genera no contiene *mutex*, puesto que no se tienen en cuenta los efectos negativos de las acciones. En este caso h_G coincide con h_{max}. La heurística h_{add} también se puede calcular utilizando un grafo relajado y haciendo agregación de costes vía una suma. Por lo tanto, también se puede considerar una heurística *level-based*.

Heurística de FF

Al igual que HSP, FF (Hoffmann and Nebel, 2001) es un planificador que hace búsqueda hacía delante en el espacio de estados, utilizando una heurística que se basa en el mismo tipo de relajación: ignorar los borrados de las acciones. Sin embargo, la técnica para calcular la heurística que utiliza FF difiere de la de HSP en dos detalles importantes:

- Se puede ver como la aplicación de GRAPHPLAN (Blum and Furst, 1995) al problema relajado.
- Las estimaciones heurísticas no realizan ningún tipo de suposición de independencia entre (sub)metas.

Para obtener la estimación heurística de un estado, FF expande un grafo de planificación para el problema relajado (*Relaxed Planning Graph* o *RPG*). Dado que en el problema relajado las acciones no tienen efectos negativos, el grafo de planificación no contiene *mutex*. Después de construir el *RPG*, éste se utiliza para extraer una solución al problema relajado,

$RP = \{a_1, \ldots, a_n\}$. A continuación, la heurística, $h_{ff}(s)$, se calcula como la longitud de la solución del problema relajado:

$$h_{ff}(\mathcal{G}, s) = |RP| \tag{2.1}$$

La solución óptima del problema relajado proporciona una heurística *admisible* para el problema original. Sin embargo, como ya se ha mencionado anteriormente, extraer el plan secuencial óptimo para el problema relajado es *NP-hard*. Por este motivo, el algoritmo de extracción que aplica FF no extrae la solución óptima, sino una solución cercana a la óptima que da lugar a una heurística no admisible. De hecho, por la forma de construir el *RPG* (en paralelo y parando en el primer nivel en el que se encuentran las metas) hay ocasiones en que éste podría no contener el plan relajado óptimo.

La Figura 2.3 muestra el algoritmo para expandir el grafo de planificación en problemas relajados. El algoritmo recibe como entradas el conjunto de metas $\mathcal{G}$, el estado a evaluar s y el conjunto de acciones relajadas $\mathcal{A}^+$. El grafo de planificación se representa mediante una secuencia P_0, A_0, ...,P_{t-1}, A_{t-1}, P_t de conjuntos de proposiciones y de acciones consecutivos. Estos conjuntos se construyen de forma incremental empezando por el estado a ser evaluado, s. Así, a partir de la primera capa $P_0 = s$ se van insertando iterativamente los efectos positivos de todas las acciones aplicables. El algoritmo falla si en un determinado punto antes de alcanzar la metas no se pueden insertar más proposiciones. Esto ocurre únicamente cuanto el problema relajado es irresoluble. En este caso la evaluación heurística devuelve infinito, ya que si el problema relajado es irresoluble también lo es el problema original. En caso de que la expansión acabe con éxito (es decir, se consigan todas las metas), se ejecuta el algoritmo de extracción del plan relajado que se muestra en la Figura 2.4.

function expansion_RPG($\mathcal{G}$,s,$\mathcal{A}^+$)
let $t = 0$;
let $P_0 = s$;
while $\mathcal{G} \nsubseteq P_t$ **do**
  $A_t = \{a \in \mathcal{A} \mid pre(a) \subseteq P_t\}$
  $P_{t+1} = P_t \cup add(a), \forall a \in A_t$
  if $P_{t+1} = P_t$ **then** fail **end if**
  $t = t + 1$
end while
final_layer$= t, succeed$

Figura 2.3: Expansión del grafo de planificación relajado en FF.

Para extraer la heurística, la única información que se necesita sobre el grafo de planificación relajado es el número de la primera capa del grafo en que aparece cada hecho y acción. Este número es lo que Hoffmann (Hoffmann and Nebel, 2001) denominó *layer-membership* de cada hecho y acción. Las entradas del algoritmo de extracción son en conjunto de metas $\mathcal{G}$ y las *layer-memberships* de todos los hechos y acciones que contiene el *RPG*. En este algoritmo, cada meta g se introduce en el conjunto de metas G_i de la capa i en la que

function extraccion_RP($\mathcal{G}$,*layer-memberships*)
let $RP = \emptyset$
for $i := 1, \ldots, m$ **do**
  $G_i := \{g \in \mathcal{G} |$ *layer-membership*$(g) = i\}$
end for
for $i := m, \ldots, 1$ **do**
  forall $g \in G_i$, no marcada como **true** en la capa i **do**
    seleccionar una acción a con $g \in add(a)$, *layer-membership*$(a) = i - 1$,
    y con dificultad mínima
    $RP = a + RP$
    forall $f \in pre(a)$, *layer-membership*$(f) \neq 0$, f no marcada como **true** en la capa $i - 1$ **do**
      $G_{layer-membership(f)} := G_{layer-membership(f)} \bigcup \{f\}$
    end for
    forall $f \in add(a)$ **do**
      marcar f como **true** en las capas i e $i - 1$
  end for
end for
return RP

Figura 2.4: Extracción del plan relajado según FF.

aparece por primera vez g. Una vez hecho esto, y comenzando por la última capa, en cada capa i se selecciona una acción con *layer-membership* igual a $i - 1$ para conseguir cada uno de los hechos en el conjunto de metas correspondiente. Si existe más de una acción para conseguir una misma meta, se selecciona la que se considera más apropiada de acuerdo con la heurística de la dificultad (explicada debajo). Las precondiciones de la acción seleccionada se introducen en el conjunto de metas correspondiente a su *layer-membership* (ya que son submetas). Cada vez que se selecciona una acción, todos sus efectos positivos se marcan como ciertos en las capas i e $i - 1$. Marcarlos como ciertos en la capa i previene que se seleccionen acciones para conseguir (sub)metas que ya son ciertas. Marcarlos como ciertos en la capa $i - 1$ asume que las acciones se linearizan en el orden en que son seleccionadas: una (sub)meta (precondición) que fue conseguida por una acción posterior no se considera como una nueva meta.

La heurística de la dificultad mide el coste de conseguir las precondiciones de una determinada acción, de forma que se prefiere siempre la acción cuya dificultad es menor. Se calcula como sigue:

$$\text{dificultad}(a) = \sum_{p \in pre(a)} \text{mín}\{i | p \in P_i\} \tag{2.2}$$

Es decir, es la suma de los índices de la primera capa (es decir, la que tiene índice mínimo) en que aparece cada precondición.

Heurística del grafo causal

La heurística el grafo causal fue propuesta por Helmert (Helmert, 2004). Para calcular esta heurística es necesario representar los problemas de planificación clásica con variables de estado multivaluadas (lenguaje SAS$^+$ explicado en la sección 2.3.2.4, Página 20). Mediante la representación SAS$^+$ no es necesario representar algunos efectos negativos de las acciones porque son una consecuencia directa de los efectos positivos, dado que las variables sólo pueden tomar un único valor en un momento dado. Así, la propia representación tiene en cuenta las interacciones negativas entre metas. Este tipo de representación es el que utiliza el planificador FAST DOWNWARD (Helmert, 2006).

Dado un problema SAS$^+$, su *grafo causal*, $CG(\Pi)$, es un grafo dirigido que contiene un nodo por cada variable en $\mathcal{V}$ y contiene un arco dirigido por cada operador que tiene en sus efectos la variable destino y en sus precondiciones o efectos la variable origen.

Por otro lado, por cada variable de estado en SAS$^+$, se puede construir un grafo dirigido que capture cómo la variable cambia su valor mediante la aplicación de operadores. Este grafo se denomina *grafo de transiciones en el dominio*, $DTG(v)$. El grafo de transiciones contiene un nodo por cada valor posible de la variable. Un arco dirigido entre dos nodos representa la existencia de un operador para cambiar el valor de la variable del valor que representa el nodo fuente al que representa el nodo destino.

La heurística del grafo causal $h_{cg}(s_\star, s)$ proporciona una estimación del número de acciones que se necesitan para alcanzar la meta desde un estado s en términos de lo que cuesta cambiar el valor en s de cada variable que aparece en las metas al valor de esa variable en $s_\star$ ($s_\star$ es una asignación parcial de valores a las variables y define las metas del problema).

$$h_{cg}(s_\star, s) = \sum_{v \in s_\star} coste_v(v_s, v_\star) \tag{2.3}$$

Los costes de esta ecuación, $coste_v(d, d')$, se definen en términos del grafo causal, $CG(\Pi)$, y de los grafos de transiciones en el dominio, $DTG(v)$. La definición asume que el grafo causal es acíclico. Cuando esto no ocurre el procedimiento relaja el grafo causal eliminando algunos arcos y define los costes tomando el grafo relajado acíclico.

Cuando v es una variable a la que no llega ningún arco en el grafo causal, $coste_v(d, d')$ es la longitud del camino más corto entre d y d' en el $DTG(v)$, o infinito en caso de que no exista este camino. En otro caso, los valores $coste_v(d, d')$, denotan el coste de un plan π que resuelve el subproblema $\Pi_{v,d,d'}$ con estado inicial $s[v = d]$ (que denota un estado como s, pero en el que la variable v toma el valor d) y meta $v = d'$. Este subproblema incluye únicamente la variable v y las variables de las que salen arcos hacia v en el grafo causal. El procedimiento para resolverlo no devuelve el coste óptimo, dado que no es tratable calcularlo. En su lugar, se aplica un algoritmo similar a Dikjstra, que hace un recorrido topológico comenzando en las variables a las que no llega ningún arco (variables raíz) en el grafo causal.

Heurísticas basadas en Bases de Datos de Patrones (PDBs)

Las Bases de Datos de Patrones (PDBs) son funciones heurísticas basadas en un uso extensivo de memoria. Se obtienen abstrayendo algunas variables del problema de manera que el problema que queda (el patrón) es suficientemente pequeño como para ser resuelto de manera óptima utilizando una búsqueda exhaustiva. Los resultados, que se almacenan en memoria como una tabla, constituyen una PDB para el problema original. Dado un estado, su valor heurístico se obtiene haciendo un mapeo a un estado abstracto y leyendo su valor asociado en la tabla, lo que proporciona un límite inferior del valor real del estado y por lo tanto una heurística admisible.

El primer trabajo que relaciona PDBs con planificación independiente del dominio es el de Edelkamp (Edelkamp, 2001). Lo que se propone en este trabajo, con la idea de hacer un uso efectivo de la memoria, es construir los patrones a partir de variables multivaluadas (implícitas en muchos problemas STRIPS) y hacer uso de cierta independencia entre estas variables, de manera que las estimaciones que se consiguen a través de distintas PDBs se puedan sumar manteniendo la admisibilidad de la heurística.

Uno de los problemas que tienen las PDBs es que dependiendo de cómo se seleccionen los patrones, las estimaciones serán mejores o peores. Existen trabajos que tratan de orientar los patrones hacia mejores estimaciones, como son (Edelkamp, 2001), (Holte et al., 2004) y (Haslum et al., 2005).

Heurísticas mejoradas

Las heurísticas basadas en la relajación que supone ignorar los efectos negativos se pueden ajustar, teniendo en cuenta de alguna forma las interacciones negativas entre proposiciones (Bryce and Kambhampati, 2006), para conseguir heurísticas más precisas. Existen varias propuestas para hacer esto:

- Considerar los *mutex* binarios estáticos, como se hacía en HSP-R (Bonet et al., 1997; Bonet and Geffner, 1999). En la expansión del grafo relajado esto se traduce en que si cualquier par de metas son *mutex* en un nivel, no se consideran alcanzables en ese nivel. Nguyen y Kambhampati proponen utilizar más tipos de *mutex* (Nguyen and Kambhampati, 2000).
- Añadir un factor de penalización que represente las interacciones negativas que no se tienen en cuenta (por ejemplo la diferencia entre h_G y h_{max}). Este es el caso también del *occlusion penalty*, en (Baier, 2007). El *occlusion penalty* es una penalización que se suma al valor heurístico y estima el número de acciones que faltan en el plan relajado. Se calcula sumando uno por cada precondición de una acción del plan relajado que es borrada por alguna acción previa sin que entre medias haya ninguna otra acción que la añada.
- Aprender de forma automática a ajustar la heurística en función de una serie de características (incluyendo las interacciones negativas entre las acciones del plan relajado),

como se propone en (Yoon et al., 2006). La idea de este trabajo es hacer una regresión lineal que represente en función de estas características cuál es la diferencia entre la longitud del plan relajado y el coste real de alcanzar la meta.

2.3.3.2. Heurísticas de poda/ordenación

Las heurísticas de poda permiten, dependiendo de cómo se utilicen, eliminar u ordenar nodos o arcos durante el proceso de búsqueda. Cuando eliminan nodos, por lo general, no se preserva la completitud del algoritmo.

Acciones Helpful

Encontrar una solución al problema relajado no sólo es útil para generar heurísticas numéricas, sino que también se pueden generar otras heurísticas analizando las acciones que contiene el plan relajado, sus precondiciones y/o efectos. El ejemplo más conocido es la heurística de las acciones útiles o acciones *helpful*, que se introdujo por primera vez en el planificador FF (Hoffmann and Nebel, 2001).

Dado un plan relajado, de entre todas las acciones aplicables en un estado, se consideran útiles aquéllas que generan algún hecho que sea o bien una precondición de alguna de las acciones del plan relajado o una meta del problema. Es decir, son todas las acciones aplicables que consiguen hechos que, siempre según el plan relajado, se van necesitar.

En la definición original, el conjunto de proposiciones que el plan relajado requiere (bien por ser precondiciones de alguna de sus acciones o metas) en cada nivel del grafo relajado se denomina *level goals* del nivel i. Así, si las *level goals* de cada nivel son $G_0, G_1, \ldots, G_{capa\_final}$, el conjunto de acciones *helpful* en el estado s, $H(s)$, se define como:

$$H(s) = \{a \in A \mid \mathit{eff}^+(a) \cap G_1 \neq \emptyset\}$$

Poda conmutativa y simetrías

En planificación, si dos acciones no interfieren entre sí (una no borra precondiciones ni añadidos de la otra ni viceversa), da igual el orden en que estas acciones se ejecuten en el plan final. El algoritmo de búsqueda normalmente no detecta este tipo de *simetrías*. La poda conmutativa (Haslum and Geffner, 2000) se basa en fijar *a priori* un orden para estas acciones. De esta manera, se puede podar una rama del árbol de búsqueda en el momento en que aparecen dos acciones de este tipo en un orden distinto al prefijado. Es decir, sólo se continúa la búsqueda por las ramas en las que estas acciones aparecen en el orden preestablecido. Otros tipos de simetrías entre estados se han estudiado en (Fox and Long, 1999).

Heurísticas aprendidas automáticamente

Algunos ejemplos de heurísticas de poda/ordenación que se generan tras un proceso de aprendizaje automático son las reglas de control, que son estructuras `Si ... Entonces` que permiten seleccionar o rechazar operadores en función de las características de un determinado meta-estado (que incluye información relacionada con el estado actual, las metas pendientes, etc.). Algunos ejemplos de sistemas capaces de generar reglas de control son: PRODIGY-EBL (Minton, 1988), STATIC (Etzioni, 1993) y HAMLET (Borrajo and Veloso, 1997). Otro ejemplo, son los árboles de decisión relacionales, que permiten realizar una ordenación de los sucesores de cada nodo (de la Rosa et al., 2008).

2.3.3.3. Heurísticas *lookahead*

Las heurísticas *lookahead* consisten en aplicar sucesivamente las acciones que marca una política (es decir, una secuencia de acciones), previamente obtenida. De esta forma el algoritmo de búsqueda avanza rápidamente hasta estados profundos que, si la política es adecuada, deberían ser más cercanos a una solución. Aunque existen varias formas posibles de obtener una política, la más conocida es tomar como política el plan relajado, lo cual fue propuesto por primera vez por Vicent Vidal e implementado en el planificador YAHSP (Vidal, 2004). El uso del plan relajado como política *imperfecta* esta motivado por el hecho de que el plan relajado contiene acciones útiles para resolver el problema, sobre todo en su parte inicial. La forma concreta de aplicar esta política se describe posteriormente, en la sección 2.3.4.2.

Otra aproximación para generar estados *lookahead* es la planteada por Yoon (Yoon et al., 2008). En este caso, la política se aprende automáticamente.

2.3.4. Algoritmos de búsqueda heurística en planificación clásica

Los algoritmos de búsqueda progresiva que se han venido utilizando en planificación clásica se pueden dividir en dos grupos:

- Aquellos que aplican algoritmos de búsqueda local, y
- Aquellos que aplican algoritmos de tipo *mejor primero*.

2.3.4.1. Búsqueda local

La búsqueda local funciona de la siguiente forma: de entre la vecindad de un único nodo actual, elige un nodo que pasa a ser el nuevo nodo actual y el proceso se repite de forma iterativa hasta que se cumpla un criterio de terminación. La búsqueda local se caracteriza porque no tiene memoria y no revisa las decisiones una vez que éstas se han tomado, es decir, es un procedimiento irrevocable. El criterio de terminación normalmente se cumple cuando se ha encontrado una solución, cuando se ha realizado un determinado número de iteraciones y aparentemente el proceso de búsqueda está estancado o cuando no se puede continuar a partir el nodo actual.

A continuación se describen algunos tipos de búsqueda local que se han aplicado en planificación.

Hill-Climbing (HC)

La búsqueda *Hill-Climbing* o búsqueda en escalada es uno de los procedimientos más conocidos de búsqueda local. En este caso, el siguiente nodo de cada iteración se elige de entre los que tienen una mejor evaluación heurística de entre todos los sucesores. En caso de que el nodo elegido tenga una evaluación heurística peor que el padre, la búsqueda temina.

El funcionamiento de la búsqueda *HC* depende de la topología del espacio de búsqueda. Cuando se encuentra un *óptimo local*, es decir, un estado para el que todos los sucesores tienen peor evaluación, la búsqueda termina sin encontrar solución alguna. Cuando se encuentra un *plateau*, es decir, un área del espacio de búsqueda para el que todos los estados visibles tienen la misma evaluación heurística, la búsqueda se comportará como un *random walk*, si la forma de elegir entre los mejores sucesores es estocástica, lo que puede suponer no avanzar hacia la meta.

El planificador HSP utiliza un esquema de búsqueda *HC*, con algunas variantes:

- Los conflictos entre sucesores con la misma evaluación heurística se resuelven de forma arbitraria.
- Se cuenta el número de movimientos en *plateaus* de manera que cuando este número excede un umbral la búsqueda se reinicia.
- Los estados generados se almacenan en una memoria rápida (tabla hash) para evitar estados repetidos en la búsqueda.

Enforced Hill-Climbing (EHC)

La búsqueda *Enforced Hill-Climbing (EHC)* es una variante de *Hill-Climbing* en la que se generan sucesores utilizando búsqueda en amplitud, hasta encontrar un sucesor (que puede ser indirecto) con una evaluación heurística mejor que el nodo actual. En este caso el nuevo nodo actual es este sucesor. La búsqueda en amplitud es en realidad un método para escapar de los *plateaus*.

EHC es el procedimiento de búsqueda que utiliza el planificador FF (Hoffmann and Nebel, 2001). En el caso de FF el algoritmo, para evitar ciclos, poda los estados repetidos (manteniendo una tabla *hash*) y no expande los estados que la función heurística reconoce como estados sin salida o *dead-ends*. Además FF aplica EHC utilizando la poda de las acciones *helpful* (es decir, sólo se consideran los sucesores debidos a acciones *helpful*), lo que hace la búsqueda aún más incompleta. En algunos casos, esto puede llevar al algoritmo a estados a partir de los cuales no se puede alcanzar un estado meta. Para aliviar este problema, en FF, cuando dado el nodo raíz actual el algoritmo EHC falla, se anula la poda por acciones *helpful* y se vuelve lanzar una búsqueda EHC para ese nodo considerando todos

los sucesores. Si esto también falla, FF inicia una búsqueda *mejor primero*, con función de evaluación $f(n) = \omega \times h(n) + g(n)$, desde el principio.

2.3.4.2. Búsqueda *mejor primero*

La búsqueda *mejor primero* constituye un esquema de búsqueda global, que se caracteriza por elegir el siguiente nodo a expandir basándose en una función de evaluación $f(n)$. Normalmente se consideran mejores los nodos con un menor valor para la función de evaluación. La búsqueda *mejor primero* se suele implementar utilizando una lista ordenada en orden creciente de $f(n)$, la lista ABIERTA, de manera que el siguiente nodo a expandir es siempre el primer elemento de la lista. Cuando un nodo se expande, se generan todos sus sucesores, y éstos se introducen en la lista ABIERTA. El nodo expandido se introduce en una lista CERRADA. El algoritmo finaliza cuando el nodo que se extrae de la lista ABIERTA es un nodo solución.

El planificador HSP2 (Bonet and Geffner, 2001) utiliza un esquema de búsqueda *mejor primero* con función de evaluación $f(n) = \omega \times h(n) + g(n)$, donde el parámetro ω toma un valor de 5. Bonet y Geffner (Bonet and Geffner, 2001) mencionan que experimentalmente valores entre 2 y 10 para este parámetro no producen diferencias significativas.

El comportamiento de la búsqueda *mejor primero* depende en gran medida de la precisión de la función heurística, $h(n)$, que se utilice y de las propiedades de la misma. En el caso de una heurística *admisible*, es decir, cuyos valores nunca sobre-estiman el coste del camino óptimo, $h^*(n)$, está garantizado que el primer nodo meta que se selecciona para su expansión corresponde a la solución óptima. Si la heurística además es *consistente*, es decir, para cualquier par de nodos n y n' se cumple que $h(n) \leq c(n, n') + h(n')$, donde $c(n, n')$ es el coste óptimo entre n y n', el coste $g(n)$ de cualquier nodo es siempre óptimo en el momento de su expansión (es decir, cuando es el primer nodo de la lista ABIERTA). Consistencia implica admisibilidad (aunque no al contrario) mientras que no admisibilidad implica inconsistencia.

Cuando se trabaja con heurísticas admisibles, las propiedades teóricas permiten establecer que una heurística h_1 es *más informada* que otra h_2 cuando para todos los nodos $h_1(n) \geq h_2(n)$. Es decir, la heurística que proporciona valores mayores es más informada. El uso de una heurística admisible más informada garantiza un menor o igual número de nodos expandidos para encontrar la solución.

Cuando se trabaja con heurísticas inconsistentes es posible que después de haber expandido un nodo se pueda encontrar un camino de menor coste hasta él (es decir, un camino con menor $g(n)$). En este caso, el nuevo coste se debe propagar a los sucesores del nodo. Una forma de llevar esto a cabo es *reabrir* el nodo, que ya está en la lista CERRADA, volviéndole a incluir en *ABIERTA*. Cuando el nodo se vuelva a expandir, su nuevo coste $g(n)$ se propagará a sus sucesores. Este procedimiento implica que el mismo nodo se pueda expandir varias veces.

Por otro lado, el peso $\omega \geq 1$ de la función heurística permite que se pueda establecer un balance entre la calidad de la solución y el tiempo que conlleva encontrarla. Cuando $\omega > 1$ y se está utilizando una heurística admisible, se encontrará una solución subóptima

cuya suboptimalidad está limitada: el coste de la solución no será mayor que ω veces el de la solución óptima (Davis et al., 1988). El uso de una ω mayor provoca que se consideren más atractivos los nodos que se estiman más cercanos a la meta. Por lo tanto, la búsqueda se parece más a una búsqueda en profundidad (es más avara) y se acelera el proceso de encontrar la solución. Se han estudiado distintas variaciones. Por ejemplo, existe una aproximación denominada *dynamic weighting* en la que ω se va ajustando a medida que avanza la profundidad (Polh, 1973).

Existen planificadores que utilizan esquemas derivados de *mejor primero*, como los que se explican a continuación.

Búsqueda IDA*

Haslum y Geffner (Haslum and Geffner, 2000) proponen utilizar la heurística h^2 (admisible) en el contexto de una búsqueda IDA* (*Iterative-Deepening* A^* (Korf, 1985)). El algoritmo IDA* simula la idea de un algoritmo *mejor primero* (A*), pero utilizando memoria lineal. Consiste en realizar series de iteraciones, de manera que cada iteración es una búsqueda en profundidad con un umbral de coste asociado. Para cada nodo generado en la búsqueda en profundidad, se calcula su coste $f(n) = h(n) + g(n)$. Si este coste es menor o igual que el umbral para la iteración, la búsqueda en profundidad continúa expandiendo el nodo. Sin embargo si el coste es mayor que el umbral, la rama se poda y la búsqueda hace *backtracking*.

Cuando una iteración se completa sin haber encontrado solución, se comienza la siguiente iteración con un umbral mayor. El umbral de coste para cada iteración se calcula como el mínimo coste de todos los nodos generados y no expandidos (podados) en la iteración previa. En la primera iteración el umbral de coste se fija al valor heurístico, $h(n)$, del estado inicial.

Adicionalmente, en el algoritmo propuesto por Haslum y Geffner se utiliza poda conmutativa.

Búsqueda *mejor primero* avara

En el planificador FAST DOWNWARD (Helmert, 2006) se utiliza un esquema *mejor primero*, pero avaro (*Greedy Best-First Search*) (Russell and Norvig, 2002), es decir con función de evaluación $f(n) = h(n)$. La función heurística $h(n)$ que utiliza FAST DOWNWARD es la heurística del grafo causal. En el caso de este planificador, el algoritmo de búsqueda original se modifica en los siguientes aspectos:

- Utiliza operadores preferidos o *preferred operators*, que se corresponden con las acciones *helpful*. Para esto mantiene dos listas ABIERTAS: una en la que introduce **todos** los sucesores de cada nodo y otra en la que introduce los sucesores debidos a *preferred operators*. El siguiente nodo a expandir se selecciona alternativamente de estas dos listas. Esto supone que los sucesores debidos a *preferred operators* se seleccionen normalmente mucho antes que los otros, dado que este número es normalmente más

pequeño que el numero total de sucesores. Esta aproximación implica evaluar todos los sucesores de cada nodo después de haberlos generado, lo cual puede ser demasiado costoso.

- Realiza una evaluación heurística retrasada. La aproximación del punto anterior implica evaluar todos los sucesores de cada nodo después de haberlos generado, lo cual puede ser muy costoso en términos de tiempo. Por este motivo FAST DOWNWARD, en lugar de evaluar todos los sucesores de cada estado, los sitúa en la lista *ABIERTA* de forma consecutiva, asumiendo que tienen la misma evaluación heurística que su padre. Así, Los sucesores se evalúan al ser expandidos. Esto permite ahorrar un número considerable de evaluaciones heurísticas.

Búsqueda con estados *lookahead*

En el planificador YAHSP (Vidal, 2004) también se emplea un esquema de búsqueda *mejor primero*, pero combinado con la aplicación de una heurística *lookahead*. En este trabajo se demuestra que la inclusión de estados *lookahead* en un proceso de búsqueda heurística hacia delante, reduce significativamente el número de estados evaluados, y por lo tanto el tiempo de búsqueda, aunque normalmente implica que la longitud del plan se incremente. Los estados *lookahead* en este trabajo se obtienen mediante la aplicación sucesiva de las acciones del plan relajado del estado actual. El éxito de esta aproximación depende fuertemente de la calidad del plan relajado. Es decir, de la similitud entre el plan relajado y el plan real, que depende del dominio. Se ha demostrado experimentalmente que los planes relajados tienen menos calidad en dominios en los que hay muchas interacciones entre metas o submetas. Normalmente en este tipo de dominios la tarea de planificación es compleja para todos los planificadores existentes.

A la hora de plantear la búsqueda con estados *lookahead* hay tres aspectos a tener en cuenta:

- Qué plan relajado se aplica: dado un estado pueden existir varios planes que resuelven el problema relajado, por lo tanto hay que determinar qué procedimiento se utiliza para conseguir un plan relajado de buena calidad.
- Cómo se aplica el plan relajado, es decir, en qué orden se aplican las acciones que el plan relajado contiene para conseguir el estado *lookahead*.
- Qué algoritmo de búsqueda se utiliza y cómo en este algoritmo se incluyen los estados *lookahead*.

Los dos primeros aspectos se resuelven aplicando criterios heurísticos y considerando como base el plan relajado que se obtiene con el planificador FF. Estos criterios son los siguientes:

1. Los planes relajados se obtienen ignorando todas las acciones que borran las metas del problema. Si ignorar estas acciones hace imposible encontrar un plan relajado, el

plan relajado se contruye como se hace habitualmente, es decir, considerando todas las acciones del dominio.

2. En el proceso de extracción del plan relajado a partir del grafo de planificación relajado, se permite conseguir una (sub)meta (meta del problema o precondicion de una acción del plan relajado) mediante una acción del mismo nivel. Para llevar esto a cabo, las (sub)metas se marcan como ciertas en el nivel en que aparece la acción que las requiere. Esto permite que el rango de acciones que se puede seleccionar para conseguir cada (sub)meta sea más amplio. Esto modifica el proceso de extracción del planificador FF, ya que en este caso las (sub)metas se marcan como ciertas en el primer nivel en que aparecen.
3. Las acciones del plan relajado están ordenadas según fueron seleccionadas en el proceso de extracción: las acciones que añaden una (sub)meta en el nivel k se sitúan después de las acciones que añaden (sub)metas en niveles $i < k$. Si dos acciones producen (sub)metas en el mismo nivel, se sitúa después aquella que produce la (sub)meta para una acción en un nivel posterior. En este caso de dos acciones que producen (sub)metas en el mismo nivel, se sitúa antes la acción que no borra ninguna precondición de la otra.
4. El estado *lookahead* se genera utilizando tantas acciones como es posible del plan relajado.
5. Cuando no se puede aplicar ninguna acción más del plan relajado, se sustituye una acción del mismo por otra acción del dominio que no pertenece al plan relajado y que cumple: (1) que es aplicable en el estado *lookahead* actual s', y (2) que produce al menos una precondición que no está en s' de una acción del plan relajado.

Respecto al tercer aspecto, el algoritmo de búsqueda es un algoritmo *mejor primero* completo, con función de evaluación $f(n) = \omega \times h(n) + g(n)$, pero modificado para considerar los estados *lookahead* y dar prioridad a las acciones *helpful*, como se muestra en la Figura 2.5. En la publicación original este algoritmo se llama *lookahead optimistic best first search*. Concretamente, el algoritmo mantiene una lista ABIERTA ordenada según $f(n)$. Cuando se genera un nodo, se incluyen en la lista ABIERTA dos instancias del mismo en este orden: la primera tendrá por sucesores sólo aquéllos generados mediante acciones *helpful*, y la segunda el resto. Desde el punto de vista de la implementación, cada nodo almacena tanto su evaluación heurística como las acciones aplicables dado el estado asociado al nodo. Así la primera instancia sólo almacena las acciones *helpful* y la segunda las demás. Ambos nodos tienen el mismo valor para la función de evaluación, puesto que representan el mismo estado. Como se sitúan en ABIERTA de forma consecutiva, forzando a que el primero sea el que contiene las acciones *helpful*, se generarán siempre primero los sucesores que vienen dados por acciones *helpful*, puesto que el nodo con estas acciones se expande siempre antes. El nodo con las acciones no *helpful* se extraerá de la lista ABIERTA cuando se hayan expandido todos los estados del subespacio de estados al que da lugar la aplicación de acciones *helpful*, que tengan un valor para la función de evaluación menor que él.

Cuando se expande un nodo y después de introducir en ABIERTA sus sucesores, se calcula el estado *lookahead* del mismo. Una vez calculado (y antes de continuar extrayendo

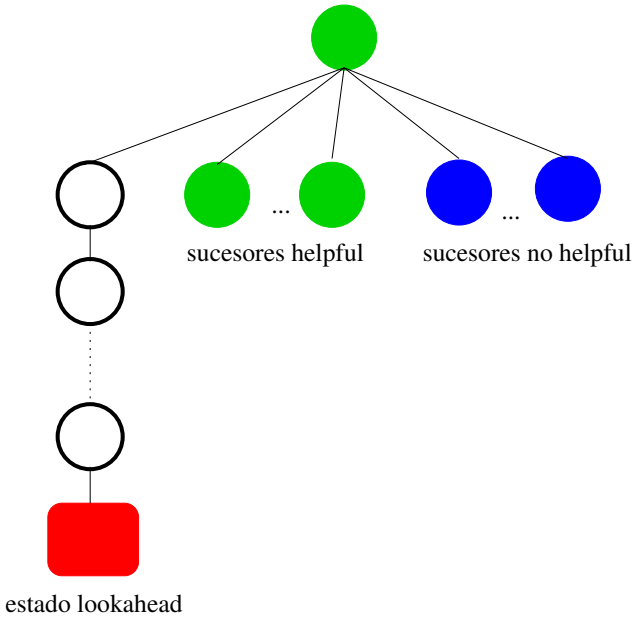

Figura 2.5: Sucesores de un nodo en la búsqueda con estados *lookahead*.

elementos de la ABIERTA) se realiza un ciclo en el que el estado *lookahead* se evalúa, se introduce en la ABIERTA (las dos instancias correspondientes) y se genera el siguiente estado *lookahead*. El ciclo termina cuando no se pueden generar más estados *lookahead* o se alcanza una solución. Después, se continúa expandiendo el mejor nodo de la lista ABIERTA.

La heurística que utiliza el algoritmo es, como en el caso de FF, el número de acciones del plan relajado. Tanto el plan relajado, como el valor heurístico y las acciones *helpful* se calculan en el mismo proceso.

Los resultados experimentales muestran que el uso de estados *lookahead* combinados con el tipo de búsqueda que se ha explicado mejoran notablemente el número de problemas resueltos y el tiempo, aunque por lo general incrementan el número de operadores de la solución. La mejora en escalabilidad se debe al ahorro de evaluaciones en dominios en los que el plan relajado es de calidad suficiente como para que se encuentre la solución a partir de los estados *lookahead* que genera. Su aplicación es más adecuada en dominios que no presentan interacciones complejas entre (sub)metas, porque en estos dominios el plan relajado se aproxima más al real. Recientemente, se ha tratado de mejorar el comportamiento en este tipo de dominios, tratando de extraer del *RPG* planes relajados en los que se minimizan los conflictos (Baier and Botea, 2009).

2.4. Planificación basada en costes

En muchas ocasiones es difícil justificar desde el punto de vista del usuario la métrica que determina la calidad de los planes en planificación clásica (la longitud del plan), porque hay

dominios en los que las únicas métricas razonables son las orientadas a costes. Es decir, el objetivo de la planificación es encontrar el plan con coste mínimo, por ejemplo en términos de consumo de combustible, distancia, tiempo, etc. El modelo de costes más sencillo es aquél en el que el coste de un plan es exactamente la suma de los costes de las acciones individuales que aparecen en él. Este modelo es más general que el clásico, ya que considerar como métrica la longitud del plan supone asumir coste unitario para todas las acciones. La planificación basada en costes se distingue de la planificación clásica en que el coste de las acciones no tiene por qué ser uniforme. En planificación basada en costes, las heurísticas numéricas deben estimar la distancia entre el estado fuente y un estado que contiene las metas en términos de coste.

En este apartado se presenta primero el modelo de planificación con costes, a continuación se introducen las modificaciones en el lenguaje de representación que implica este modelo, y posteriormente las heurísticas y los planificadores más conocidos en planificación basada en costes.

2.4.1. Modelo de planificación

El modelo más simple de planificación basada en costes es un modelo de planificación clásica en el que las acciones pueden tener **coste no uniforme**, y además se considera que el coste de un plan π es la suma de los costes de todas las acciones que contiene:

$$coste(\pi) = \sum_{a_i \in \pi} coste(a_i)$$

Un plan es **óptimo** si no existe ningún otro plan que resuelva el problema con menor coste.

Es usual que en este modelo se impongan dos restricciones adicionales:

- Que el coste de las acciones sea independiente del estado.
- Que el coste de las acciones sea no negativo.

Cuando el coste de las acciones es independiente del estado, el problema es más sencillo. En este caso, las acciones siempre tienen el mismo coste (su coste es un valor fijo), que además se conoce antes del inicio de la planificación en sí, y si no se conoce directamente se puede calcular en una etapa de preproceso. Cuando la planificación se resuelve mediante búsqueda heurística, el hecho de que el coste de las acciones sea o no independiente del estado no influye más que en el cálculo de las heurísticas. Aunque no es complicado desarrollar alguna aproximación en la que las heurísticas consideren de alguna forma que el coste de las acciones depende del estado en que éstas se aplican, por lo general las heurísticas asumen que esto no ocurre. Concretamente en las heurísticas basadas en grafos de planificación relajados no se mantienen estados, sino que cada capa de proposiciones es un superconjunto de todos los estados que se pueden generar hasta ese momento. Dado este superconjunto es posible que no se pueda determinar un coste único para cada acción aplicable. Una posible aproximación es tomar el mínimo.

2.4.2. Lenguaje de representación

La planificación basada en costes no supone muchos cambios respecto al lenguaje de representación explicado en la sección 2.3.1 (Página 14). Básicamente se necesita dotar al lenguaje de la expresividad suficiente para definir los costes de las acciones.

En la versión 2.1 de PDDL (Fox and Long, 2003)[3] se introdujeron por primera vez los cambios que permiten usar este lenguaje para planificación basada en costes. Concretamente las ampliaciones que se hicieron permiten:

- Definir variables de estado numéricas. Estas variables, en realidad son funciones que devuelven un número real y que pueden tomar como argumentos los distintos objetos definidos en el dominio. Para definirlas hay que declarar en los *requirements* del dominio que es un dominio con *fluents*, definir las cabeceras de estas variables con los tipos de sus argumentos en el apartado *functions* y definir en el estado inicial de los problemas su valor inicial. Los dos ejemplos siguientes muestran como declarar el valor de una variable numérica en el estado inicial del problema.

```
(= (time-to-drive ciudad-1 ciudad-2) 40)
(= (driven) 0)
```

- Que los efectos de las acciones modifiquen las variables de estado numéricas; es decir, la posibilidad de definir efectos numéricos, como se muestra en el siguiente ejemplo:

```
(:action drive-truck
  :parameters (?truck - truck ?loc-from - location
               ?loc-to - location ?driver - driver)
  :precondition (and (at ?truck ?loc-from)
                     (driving ?driver ?truck)
                     (link ?loc-from ?loc-to))
  :effect (and (not (at ?truck ?loc-from))
               (at ?truck ?loc-to)
               (increase (driven) (time-to-drive ?loc-from ?loc-to))))
```

  Aunque para el modelo de planificación basada en costes definido, es suficiente con el efecto numérico `increase`, el lenguaje también permite decrementar el valor de una variable numérica con `decrease` y hacer asignaciones con `assign` y multiplicaciones con `scale-up` y `scale-down`

- Que el usuario pueda expresar la métrica de calidad de cada problema en función de estas variables numéricas:

```
(:metric minimize (+ (* 2 (driven)) (* 3 (walked))))
```

Bien es cierto que algunos planificadores que hacen planificación basada en costes, como METRIC-FF suelen estar preparados para trabajar con otras características numéricas (además de las mencionadas previamente), como son:

[3] http://zeus.ing.unibs.it/ipc-5/

- Precondiciones numéricas, del tipo:

```
(>= (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))
```

- Metas numéricas, del tipo:

```
(< (total-fuel-used) 1200)
```

Sin embargo, estas caraterísticas no pertenecen al modelo de planificación basada en costes definido, sino a un modelo con menos restricciones de planificación numérica.

PDDL permite definir el modelo de planificación basada en costes. Sin embargo, normalmente el coste de las acciones no se define explícitamente en el dominio ni el problema de planificación. Como se ha explicado, lo que se define en PDDL es una métrica a ser minimizada que a su vez se define en función de una o varias variables numéricas. Por otro lado, las acciones modifican el valor de las variables numéricas que forman parte de la métrica a través de efectos de tipo `increase`.

A partir de las descripciones PDDL del dominio y del problema se extrae el coste de cada acción. Además, para que el modelo sea coherente con la forma de expresarlo en PDDL el resultado de la expresión de la métrica en el estado final debe coincidir con la suma de los costes de las acciones del plan. Para esto, es necesario que la métrica se pueda descomponer como una suma de costes de acciones. Esta descomposición sólo es posible si la expresión de la métrica es una combinación lineal de variables numéricas con pesos w_i positivos. Así, si M es la expresión de la métrica, y f_i son variables numéricas.

$$M = \sum_{i=1}^{n} w_i f_i$$

Dado un plan π, el valor de una variable numérica f_i en el estado final, se calcula como la suma de todos los efectos `increase` sobre las variables numéricas que forman parte de la métrica de todas las acciones del plan:

$$f_i = \sum_{a_j \in \pi} \Delta_{f_i}^{a_j}$$

donde $\Delta_{f_i}^{a_j}$ representa el incremento de la acción a_j sobre la variable numérica f_i, y es cero cuando la acción no tiene ningún efecto para incrementar la variable numérica f_i .

El coste del plan viene dado por el valor de la expresión de la métrica en el estado final. Cuando la métrica es una combinación lineal de variables numéricas, se pueden aplicar las propiedades distributiva y asociativa para expresar la métrica como la suma de los costes de las acciones del plan:

$$coste(\pi) = \sum_{i=1}^{n} w_i f_i = w_1 \sum_{\forall a_j \in \pi} \Delta_{f_1}^{a_j} + \cdots + w_n \sum_{\forall a_j \in \pi} \Delta_{f_n}^{a_j} = \sum_{\forall a_j \in \pi} \sum_{i=1}^{n} w_i \Delta_{f_i}^{a_j}$$

Así, el coste de cada acción se calcula como:

$$coste(a_j) = \sum_{i=1}^{n} w_i \Delta_{f_i}^{a_j}$$

Lo cual hace que la expresión de la métrica sea coherente con el modelo de planificación basada en costes, donde $coste(\pi) = \sum_{\forall a_j \in \pi} coste(a_j)$.

Para mantener la restricción de que los costes de las acciones sean no negativos, se obliga a que las acciones únicamente puedan aplicar efectos `increase` sobre las variables numéricas que forman parte de la métrica.

En general las técnicas que se utilizan en planificación basada en costes son independientes del lenguaje que se utiliza para expresar el modelo, es decir, asumen que cada acción a tiene un coste $coste(a)$. Sin embargo, PDDL es el lenguaje estándar utilizado por la comunidad de planificación y muchos planificadores basados en costes calculan el coste de cada acción como se ha explicado.

El problema de optimización al que da lugar el modelo de planificación basada en costes aquí descrito es un problema mono-objetivo, en el que aunque el coste de las acciones dependa de distintos criterios, siempre se puede sintetizar en un único valor de manera que el coste del plan se calcule de forma aditiva a partir del coste de las acciones que lo componen. Sin embargo, se podría plantear un modelo más complejo, de optimización multi-objetivo. En este caso el coste de las acciones se debería representar como un vector, ya que la ejecución de una misma acción podría mejorar un criterio y empeorar otro. En este caso habría que definir también el criterio que permite determinar que una solución es mejor que otra. El problema de planificación con optimización multi-objetivo se ha trabajado poco. Un ejemplo es el planificador MO-GRT (Refanidis and Vlahavas, 2003), en el que hay que definir la función objetivo como una jerarquía con pesos de criterios definidos por el usuario. En este caso, el coste de las acciones no se sintetiza en un único valor, sino que por cada acción se mantiene un vector de costes que tiene un elemento por cada variable numérica implicada en la métrica.

2.4.3. Técnicas de planificación y planificadores

En planificación basada en costes existen, por un lado, planificadores que utilizan técnicas de planificación clásica (o alguna versión extendida de las mismas) y por otro lado, también existen algunos planificadores cuyos algoritmos difieren significativamente de los que se han venido utilizando en planificación clásica. Dado que no hay un conjunto de técnicas que se apliquen de forma general en planificación basada en costes, en esta sección se describen los principales planificadores.

METRIC-FF (Hoffmann, 2003) es un planificador que hace búsqueda heurística progresiva en el espacio de estados inducido por el problema de planificación. Utiliza una heurística basada el ignorar los efectos negativos de las acciones que se calcula utilizando un grafo

de planificación relajado. En caso de que se defina una métrica de calidad en un problema, METRIC-FF, utiliza un algoritmo de búsqueda *mejor primero* estándar, con función de evaluación $f(n) = \omega \times h(n) + g(n)$. Por defecto, el valor de ω que utiliza es 5. METRIC-FF es un planificador que combina el modelo de planificación basada en costes con un modelo de planificación con recursos contínuos (esta combinación habitualmente se denomina planificación numérica). Una de las principales aportaciones de este planificador es que extiende el procedimiento para calcular la heurística de FF para que sea capaz de trabajar con variables numéricas. Recientemente, se ha aplicado programación lineal también con este propósito (Coles et al., 2008).

El planificador SAPA (Do and Kambhampati, 2003) fue contruído para un modelo de planificación que combina planificación basada en costes y planificación temporal. Utiliza un algoritmo *mejor primero* cuya función de evaluación es sensible tanto a los costes como a la duración total del plan considerando acciones concurrentes (*Makespan*). El espacio de estados en el caso de SAPA se compone de estados que incluyen información sobre el tiempo relevante para la planificación temporal[4].

El planificador SIMPLANNER (Onaindia et al., 2001; Sapena et al., 2004) utiliza un algoritmo de búsqueda que es una variación de *Hill-Climbing*. Esto significa que, de alguna forma, sacrifica la calidad del plan encontrado para encontrar soluciones más rápidas. SIMPLANNER es un planificador reactivo que utiliza la heurística para calcular la siguiente acción a ejecutar en el plan. Para esto, primero calcula la heurística para cada meta del problema de forma independiente, y luego combina los planes relajados obtenidos en cada caso para identificar qué acción es más útil aplicar en el estado actual.

El planificador MO-GRT (Refanidis and Vlahavas, 2003) es una versión del planificador GRT (Refanidis and Vlahavas, 2001) que es capaz de considerar a la vez varios criterios para evaluar la calidad de los planes. Utiliza una estrategia de búsqueda *mejor primero* hacia delante con función heurística multiobjetivo que se calcula en función de una serie de criterios con pesos definidos por el usuario. Su cálculo se basa en conjuntos de vectores que se asignan a los hechos del problema, y permiten estimar el coste total de conseguir ciertos hechos a partir de las metas (la heurística se calcula de forma regresiva). Al igual que en GRT, existe una etapa de preproceso en la que se estiman los valores heurísticos que se almacenan en una tabla[5].

El planificador LPG-td (Gerevini et al., 2004, 2008) (*Local search for Planning Graphs*) se basa en búsqueda local estocástica y grafos de planificación. El espacio de búsqueda consiste en "*grafos de acciones*" (*action graphs*), que son subgrafos del grafo de planificación. En cada paso de búsqueda se transforma un grafo de acciones en otro. LPG utiliza una representación compacta del grafo de planificación para poder definir la vecindad de un nodo en la búsqueda y para evaluarlo. En la evaluación se utiliza una función parametrizada cuyos parámetros dan peso a los diferentes tipos de inconsistencias que hay en el plan parcial actual. Estos parámetros se evalúan dinámicamente durante la búsqueda utilizando multiplicadores de Lagrange discretos[6] (Vapnyarskii, 2001). La función de evaluación utiliza heurísticas para

[4] *Time-stamped states.*
[5] Denominada *greedy regression table.*
[6] Los multiplicadores de Lagrange permiten optimizar una función de varias variables sujeta a restricciones.

estimar el coste de búsqueda y el coste de ejecución para conseguir una precondición (que puede ser numérica). Las cantidades numéricas también están modeladas en la función de evaluación. El sistema puede producir planes de buena calidad porque utiliza un proceso *anytime* que produce una secuencia de planes, donde cada plan mejora los anteriores en términos de calidad. Está integrado con un algoritmo de búsqueda hacia delante *mejor primero*, al que cambia automáticamente cuando se han llevado a cabo una serie de pasos de búsqueda y reinicios.

El planificador SGPLAN (Chen et al., 2006) divide los problemas de planificación en subproblemas, cada uno con sus propias submetas. Después ordena los subproblemas de acuerdo a una resolución secuencial de sus submetas y encuentra un plan válido para cada meta, asegurándose de que la resolución de una meta posterior es consistente con las de las submetas previas. Este planificador implementa métodos para la detección de un orden razonable entre submetas, para hacer una descomposición jerárquica de los subproblemas, para reducir el espacio de búsqueda eliminando acciones irrelevantes en subproblemas y una estrategia para llamar al mejor planificador para resolver los subproblemas básicos. La implementación actual utiliza como planificador básico un METRIC-FFampliado para tratar todas las características de PDDL2.2, planificación paralela, y planificación con las restricciones mutex particionadas.

Otros planificadores, como los derivados de HSP y FAST DOWNWARD también se pueden utilizar para planificación basada en costes, modificando la heurística para que tenga en cuenta los costes de las acciones, aunque se ha realizado muy poca experimentación en este sentido.

Existen también planificadores capaces de trabajar con métricas que siguen otras estrategias de planificación que requieren de información dependiente del dominio, como son: SHOP2 (Nau et al., 2003) y TLPlan (Bacchus and Kabanza, 2000). SHOP2 es un planificador HTN, que por lo tanto, necesita una base de conocimiento dependiente del dominio que le proporcione los métodos. TLPlan utiliza información del dominio para controlar una búsqueda hacia delante. Esta información se expresa de forma declarativa utilizando una lógica temporal de primer orden.

2.4.4. Heurísticas en planificación basada en costes

En esta sección se describen de forma general las heurísticas numéricas que se han venido utilizando en planificadores basados en costes que realizan búsqueda en el espacio de estados.

Aunque las heurísticas de la familia h^m (descritas en la sección 2.3.3.1) no se han utilizado en planificación basada en costes, su definición incluye el caso de que el coste de las acciones sea distinto de 1, por lo que su aplicación es directa.

2.4.4.1. Heurísticas de HSP con costes

Las heurísticas del planificador HSP, como se explicó en la sección 2.3.3.1, fueron diseñadas inicialmente para planificación STRIPS, aunque su definición se puede generalizar

fácilmente para el caso de planificación con costes. Esta generalización, que consiste en incluir en las fórmulas el coste de las acciones, ya se propuso en (Haslum and Geffner, 2001) para planificación temporal, donde los costes son duraciones.

La definición formal generalizada de $h(p, s)$ para una proposición p en un estado s, es:

$$h(p,s) = \begin{cases} 0 & \text{si } p \in s \\ \min\limits_{a \in A(p)} \{coste(a) + h(pre(a), s)\} & \text{en otro caso} \end{cases}$$

donde $A(p)$ es el conjunto de acciones que añaden p, y $h(pre(a), s)$ es el valor de la heurística para el conjunto de precondiciones de a. Cuando $A(p) = \emptyset$, el valor de la heurística es infinito. Al igual que en el caso de STRIPS, la idea es estimar el coste de un conjunto de proposiciones como el coste del átomo con máximo coste. Así la definición de $h(C, s)$ para un conjunto de proposiciones C en un estado s es:

- Para la heurística del máximo:

$$h_{max}(C,s) = \max_{p \in C} \{h_{max}(p,s)\}$$

- Para le heurística aditiva:

$$h_{add}(C,s) = \sum_{p \in C} h_{add}(p,s)$$

Al igual que en planificacicón clásica, la heurística del máximo es admisible, mientras que la heurística aditiva no lo es.

2.4.4.2. Heurística de METRIC-FF

METRIC-FF (Hoffmann, 2003) expande el grafo de planificación y extrae una solución prácticamente como lo hace FF (Hoffmann and Nebel, 2001), con las ampliaciones que suponen las características de la planificación numérica:

- Incluye el tratamiento de precondiciones y metas numéricas.
- Los efectos numéricos se relajan ignorando aquéllos que suponen decrementos (`decrease`).

Para que el hecho de ignorar los efectos que decrementan suponga una verdadera simplificación del problema, es necesario que la tarea de planificación relajada sea *monótona*, que es lo que sucede en tareas STRIPS al ignorar las listas de borrados: si s y s' son dos estados sucesivos, $s \subseteq s'$. La monotonicidad numérica se consigue si todas las metas y precondiciones numéricas comparan variables con constantes utilizando exclusivamente los operadores $>$ ó $\geq$. De esta manera, una precondición cierta en un estado s lo será también el un estado posterior s'. La heurística de METRIC-FFestá pensada para dominios en los que se cumple esta propiedad o se puede hacer cumplir mediante una etapa de preproceso

o traducción, como por ejemplo en *tareas lineales*, en las que las variables numéricas sólo aparecen en funciones lineales.

La inclusión del tratamiento de metas y precondiciones numéricas en el algoritmo que genera el grafo de planificación relajado consiste en lo siguiente:

- Por cada variable numérica, en cada capa de proposiciones, se incluye el máximo valor que se ha conseguido hasta el momento para esa variable (a través de la aplicación de los incrementos de los efectos de las acciones). En la capa inicial este valor es el que aparece en el estado inicial.
- Las metas se alcanzan cuando las metas proposicionales aparecen en una capa y las metas numéricas son ciertas teniendo en cuenta los valores máximos propagados para las variables numéricas.
- Se llega al punto fijo cuando no se pueden conseguir más proposiciones y los valores máximos propagados para las variables numéricas son mayores de lo que se necesita o no se pueden superar.
- El proceso de extracción de la solución supone que una meta numérica se cumple en el primer nivel en que la condición que se impone sobre la correspondiente variable numérica es cierta. Por ejemplo, si la meta numérica es `(fuel>100)`, se considera que ésta se cumple en el primer nivel de proposiciones en el que el valor de la variable numérica `fuel` sea mayor que 100. El valor de una variable numérica, como se menciona en el primer punto, es el máximo conseguido hasta el momento.

Dejando de lado el tratamiento de las características numéricas que hace METRIC-FF no relacionadas directamente con el modelo de planificación basada en costes, la heurística de METRIC-FF se calcula de forma muy similar a la heurística de FF, el planificador previo para planificación STRIPS, descrita en la sección 2.3.3.1. El grafo de planificación relajado es exactamente el mismo, y el plan relajado se extrae de manera similar. La diferencia principal es que en el caso de METRIC-FF el valor heurístico es la suma de los costes de las acciones del plan relajado, en lugar de la longitud de éste.

La Figura 2.6 muestra el proceso de extracción. El nivel de cada (sub)meta se refiere al primer nivel proposicional del grafo en el que la (sub)meta aparece. El algoritmo construye inicialmente los conjuntos G_i que contienen las metas de cada nivel. A continuación comienza un proceso hacia atrás avaro, en el que empezando por la última capa G_i de metas, se seleciona para cada meta una acción de la capa $i-1$ que la consiga, y sus precondiciones se añaden al conjunto de metas G_j, que se corresponde con el nivel de la precondición.

En este caso, el valor final de la heurística se calcula como la suma los costes de las acciones del plan relajado, mientras que en FF se calculaba como el número de acciones del plan relajado.

$$h_{mff}(\mathcal{G}, s) = \sum_{a_i \in RP} coste(a_i) \tag{2.4}$$

function extraccion_RP($\mathcal{G}$,RPG)
for $i := 1, \ldots, final\_layer$ **do**
  $G_i := \{g \in \mathcal{G} | level(g) = i\}$
end for
for $i := final\_layer, \ldots, 1$ **do**
  forall $g \in G_i$ **do**
    seleccionar una acción a con $g \in add(a)$ y $level(a) = i - 1$,
    forall $f \in pre(a)$ **do**
      $G_{level(f)} = G_{level(f)} \bigcup \{f\}$
    end for
  end for
end for

Figura 2.6: Extracción del plan relajado en METRIC-FF.

2.4.4.3. Heurísticas de SAPA

El planificador SAPA (Do and Kambhampati, 2003) fue el primero en introducir la idea de un *RPG* con propagación de costes. Los *RPGs* que construye este planificador están ideados para problemas de planificación temporales, numéricos y con métrica de calidad, porque la idea es llegar a un buen balance entre el coste del plan y su *makespan*. El algoritmo básico (Bryce and Kambhampati, 2007) para la problemas con métrica de calidad, difiere del algoritmo de METRIC-FF en que realiza propagación de costes en el *RPG*. La idea es propagar el mejor coste con el que se consigue cada proposición en cada nivel. Para llevar esto a cabo, se asocia a cada proposición un valor que indica su coste. Además, a cada acción también se le asocia un valor que indica el coste total de aplicarla, es decir, el coste de sus precondiciones más el coste propio de la acción. Los costes de las proposiciones y de las acciones no están relacionados con el nivel del *RPG* en el que aparecen.

El grafo de planificación se construye de forma similar al de METRIC-FF, pero con la propagación de costes. Además de ésta, otra diferencia significativa es que el grafo de planificación relajado se debe expandir hasta que no se puedan aplicar más acciones, en lugar de parar el proceso en el primer nivel en que se encuentran las metas. Esto es así para garantizar que cuando termina el proceso de expansión, el coste asociado a cada proposición es mínimo.

El algoritmo de expansión del grafo de planificación se muestra en la Figura 2.7. Bryce y Kambhampati (Bryce and Kambhampati, 2007) realizan únicamente una descripción textual. El algoritmo que se muestra aquí se ha escrito siguiendo esa descripción. En el proceso de propagación de costes:

- $coste\_actual(p)$ y $coste\_actual(a)$ representan el valor actual de los costes de la proposición p y de la acción a respectivamente.
- $coste(p, i)$ representa el coste de la proposición p en la capa P_i.

- $coste(a, i)$ representa el coste de la acción a en la capa A_i.

Todas las proposiciones del estado inicial tienen coste cero, mientras que las que no están en el estado inicial, tienen un coste inicial infinito. En cada capa, a cada acción se le asigna la suma del coste de la acción, $coste(a)$, más el coste de alcanzar sus precondiciones. Este último coste viene dado por la función `Agregar`. El coste de las precondiciones de una acción se puede agregar vía un máximo o vía una suma. Estas dos formas de agregar los costes de las precondiciones dan lugar a dos posibles definiciones de la función `Agregar`, que generan dos heurísticas distintas. Denominaremos a estas heurísticas $h_{sapa\text{-}max}$ y $h_{sapa\text{-}add}$. En el caso de $h_{sapa\text{-}max}$ los costes de las precondiciones se agrega vía un máximo, mientras que en el caso de $h_{sapa\text{-}add}$ los costes de las precondiciones se agregan vía una suma.

En cada capa, el coste de cada proposición se calcula como el mínimo entre el coste actual de la proposición y los costes de las acciones de la capa previa que consiguen la proposición. De esta forma, los costes de las proposiciones se decrementan con las capas. Smith (Smith, 2004) utiliza también un proceso de propagación de costes similar a éste.

Como en el caso de METRIC-FF, una vez terminado el *RPG*, se extrae del mismo un plan relajado (*RP*). Después, se calcula la heurística, h_{sapa}, como el coste de las acciones en ese plan.

$$h_{sapa}(\mathcal{G}, s) = \sum_{a_i \in RP} cost(a_i) \tag{2.5}$$

El proceso de extracción es parecido al de METRIC-FF. Es un proceso hacia atrás avaro, que elige la acción cuyo coste actual sea más bajo para resolver cada (sub)meta.

2.4.4.4. Procedimiento de SIMPLANNER

El planificador SIMPLANNER también utiliza *RPGs*. En este caso, los *RPGs* no se utilizan para extraer un valor heurístico del estado actual, sino que SIMPLANNER utiliza la información en el *RPG* para determinar, aplicando ciertos criterios heurísticos, cuál es la siguiente acción a ejecutar. En el *RPG* que genera SIMPLANNER los niveles del grafo no representan pasos en el tiempo sino costes de acuerdo con la métrica definida para el problema. La justificación de que esto sea así es que para estimar costes parece más adecuado construir el grafo de planificación de una forma orientada a hacer esta estimación. La heurística de METRIC-FF calcula la estimación para un problema de planificación basado en costes a partir de un grafo de planificación que por definición está construido para minimizar la longitud de los planes.

En el planificador SIMPLANNER (Onaindia et al., 2001) el grafo de planificación relajado se expande como muestra el algoritmo de la Figura 2.8. Según este algoritmo se mantiene una lista de proposiciones programadas, *prog_prop*, que contiene para cada proposición el mínimo coste con que ésta se ha conseguido hasta el momento. Inicialmente *prog_prop* contiene las proposiciones del estado inicial (cuyo coste es 0). Las proposiciones que no aparecen en *prog_prop* tienen coste ∞. Una vez generada una capa de proposiciones, P_i, se utiliza el coste de la proposición en *prog_prop* con coste mínimo (c) para limitar las

function expansion_RPG_SAPA($\mathcal{G}$,s,$\mathcal{A}^+$,$\mathcal{P}$)
let $i = 0$;
let $P_0 = s$;
let $coste\_actual(p) = \begin{cases} 0 & \text{if } p \in P_0 \\ \infty & \text{en otro caso} \end{cases}, \forall p \in \mathcal{P}$;
let $coste(p, 0) = 0, \forall p \in P_0$;
let $coste\_actual(a) = \infty, \forall a \in \mathcal{A}$;
while `true` **do**
/* nivel de acciones */
$A_i = \{a \in \mathcal{A} \mid pre(a) \subseteq P_i \wedge [coste(a) + \text{Agregar}_{\forall p \in pre(a)}(coste\_actual(p))] < coste\_actual(a)\}$

/* actualización de costes (acciones)*/
forall $a \in A_i$ **do**
$coste\_actual(a) = \text{mín}\{coste\_actual(a), coste(a) + \text{Agregar}_{\forall p \in pre(a)}(coste\_actual(p))\}$;
$coste(a, i) = coste\_actual(a)$
end for

/* nivel de proposiciones*/
$P_{i+1} = P_i \cup_{a \in A_i} add(a)$

/* actualización de costes (proposiciones) */
forall $p \in P_{i+1}$ **do**
$coste\_actual(p) = \underset{a \in A_i, p \in add(a)}{\text{mín}} \{coste\_actual(p), coste\_actual(a)\}$;
$coste(p, i + 1) = coste\_actual(p)$
end for

if $P_{i+1} = P_i$ and $\forall p \in P_{i+1}, coste(p, i + 1) = coste(p, i)$ **then**
if $\mathcal{G} \subseteq P_i$ **then return** $P_0, A_0, ..., A_{i-1}, P_i$ /* condición de éxito*/
else return fail /* condición de fallo */
end if
end if
$i = i + 1$;
end while

Figura 2.7: Algoritmo de SAPA para construir el *RPG*.

acciones del siguiente nivel de acciones, A_i, de manera que sólo se pueden aplicar aquellas acciones cuyas precondiciones tienen coste menor o igual que c. Una vez en este punto, se eliminan de *prog_prop* las proposiciones con coste igual a c. Cuando una acción, a, consigue una proposición, p, su coste se actualiza en *prog_prop* de la siguiente forma:

$$coste(p) = min\{coste(p), cost(a) + \sum_{q \in prec(a)} coste(q)\}$$

function expansion_RPG_Simplanner($\mathcal{G}$,s,$\mathcal{A}^+$,$\mathcal{P}$)
let $prog\_prop = s$;
let $coste(p) = \begin{cases} 0 & \text{si } p \in s \\ \infty & \text{en otro caso} \end{cases} \quad \forall p \in \mathcal{P}$
while $\exists g \in \mathcal{G}, coste(g) = \infty$ **do**
  if $prog\_prop = \emptyset$ **then** fail **end if**
  $c = \min_{p \in prog\_prop} \{coste(p)\}$
  $P_c = \{p \mid p \in prog\_prop \wedge coste(p) = c\}$
  $prog\_prop = prog\_prop \setminus P_c$
  $A_c = \{a \in \mathcal{A}^+ \mid \forall p \in pre(a), coste(p) \leq c\}$
  for all $a \in \mathcal{A}^+$ **do**
    $coste\_de\_alcanzar(a) = \sum_{p \in pre(a)} coste(p)$
    for all $p \in add(a) \wedge (coste\_de\_alcanzar(a) + coste(a) < coste(p))$ **do**
      $prog\_prop = prog\_prop \cup \{p\}$
      $coste(p) = coste\_de\_alcanzar(a) + coste(a)$
    end for
  end for
end while
succeed

Figura 2.8: Expansión del grafo de planificación relajado en SIMPLANNER.

2.4.4.5. Heurísticas de reciente aparición

Recientemente han aparecido algunas heurísticas para planificación basada en costes. Entre ellas se encuentran:

- La heurística *set-additive* (Keyder and Geffner, 2007).
- La heurística h_a (Keyder and Geffner, 2008).
- La heurística h_{pmax} (Mirkis and Domshlak, 2007).
- Heurísticas derivadas del análisis de *landmarks* (Richter et al., 2008).

La heurística *set-additive* se obtiene haciendo una modificación en la formulación de la heurística aditiva, para propagar acciones en lugar de escalares. Según la heurística aditiva, el valor heurístico de una proposición viene dado por la suma del coste de una acción que genera la proposición más la suma de los costes de sus precondiciones. La acción que se elige de entre todas las que generan la proposición es la que minimiza esta suma, que se denomina *best supporter*. Así, la heurística aditiva se puede reformular de la siguiente manera:

$$h_{add}(p,s) = \begin{cases} 0 & \text{si } p \in s \\ h_{add}(a_p, s) & \text{en otro caso} \end{cases}$$

donde

$$a_p \in \arg\min_{a \in A(p)} h_{add}(a, s)$$

es el *best supporter* de p en s.

En la heurística aditiva el valor $h(a_p, s)$ del *best supporter* a_p se propaga para obtener el valor heurístico de $h(p, s)$. En la heurística *set-additive* lo que se propaga es el propio *best supporter*, resultando en una función $\pi(p, s)$ que representa un conjunto de acciones:

$$\pi(p, s) = \begin{cases} \emptyset & \text{si } p \in s \\ \pi(a_p, s) & \text{en otro caso} \end{cases}$$

donde

$$a_p \in \arg\min_{a \in A(p)} Coste(\pi(a, s))$$

$$\pi(a, s) = \{a\} \bigcup \{ \bigcup_{q \in pre(a)} \pi(q, s) \}$$

$$Coste(\pi(a, s)) = \sum_{a' \in \pi(a,s)} coste(a')$$

Además, en lugar de realizar una suma de valores, los *best supporters* de las precondiciones se combinan mediante una unión. La heurística *set-additive* se calcula de la siguiente forma:

$$h_{set-additive}(\mathcal{G}, s) = Coste(\pi(G, s))$$

donde se está asumiendo que existe una acción ficticia *End* con precondiciones las metas $\mathcal{G}$ que genera el efecto ficticio G.

Las acciones en $\pi(p, s)$, ordenadas respetando el orden causal, constituyen una solución al problema relajado con estado inicial s y meta p.

La heurística h_a sigue la misma idea que la *set-additive*, pero es aún más similar a la heurística aditiva, ya que en el conjunto $\pi(p, s)$ se recolectan los *best supporters* según la heurística aditiva. Su formulación es:

$$\pi(p, s) = \begin{cases} \emptyset & \text{si } p \in s \\ \{a_p\} \bigcup \{ \bigcup_{q \in pre(a)} \pi(q, s) \} & \text{en otro caso} \end{cases}$$

donde

$$a_p \in \arg\min_{a \in A(p)} coste(a) + \sum_{q \in pre(a)} h_{add}(q, s)$$

La diferencia con la heurística *set-additive* radica en la forma de seleccionar los *best supporters*.

La heurística $h_{\max}$ se calcula propagando en el grafo de planificación relajado vectores de coste con un elemento por proposición. La Figura 2.9 muestra cómo se realiza esta propagación. El vector de coste para una acción contiene para cada proposición el máximo

valor para esa proposición en los vectores de coste de sus precondiciones. El vector de coste para una proposición es el vector de coste de la acción que genera esa proposición para la cual la suma de todos los elementos de su vector de costes más el coste de la acción es mínima. Los vectores de costes de las proposiciones en el estado inicial son vectores de ceros. Además, el coste de cada acción se calcula como el propio coste de la acción dividido entre el número de sus efectos positivos. El valor final de la heurística se calcula como la suma de los valores en el vector de costes de una acción ficticia que tiene como precondiciones las metas.

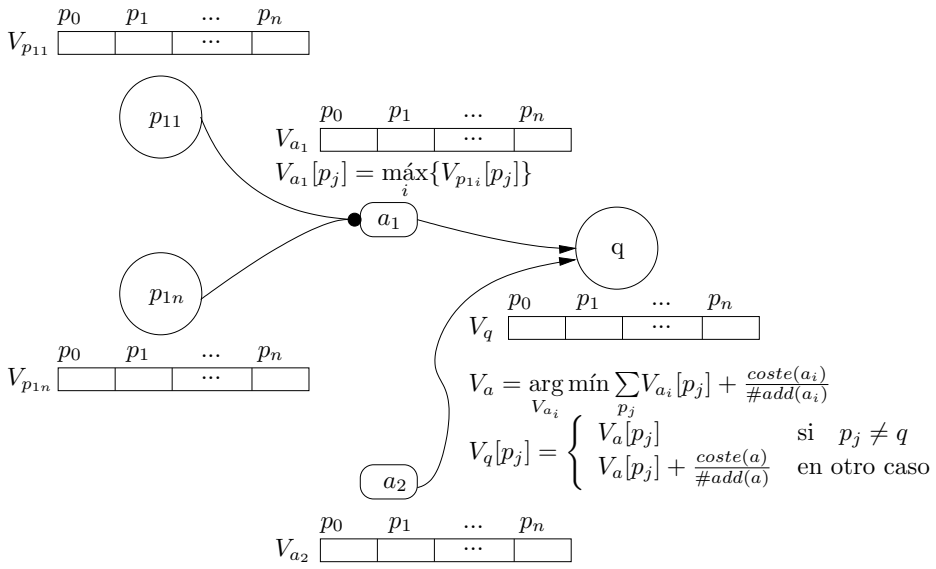

Figura 2.9: Propagación de vectores en $h_{\max}$.

La heurística $h_{\max}$ se deriva de una generalización de las heurísticas para planificación basada en costes que se calculan mediante propagación de elementos en un grafo de planificación relajado. En esta generalización, los grafos de planificación se representan como grafos AND/OR con pesos (*WAODAGs*[7]), y el proceso general de propagación define cómo se propaga la información a través de los nodos y arcos de estos grafos. La generalización deja abierto el tipo de información que se propaga.

Por último, los *landmarks* (Porteous et al., 2001; Richter et al., 2008) en una tarea de planificación son las proposiciones que deben ser ciertas en el algún momento en todos los planes solución. Dado un estado, los *landmarks* se pueden estimar a partir de un *RPG*. Últimamente se ha propuesto utilizar el número de *landmarks* de un estado como su valoración heurística. Entre otras, esta heurística se utiliza en el planificador LAMA (Richter and Westphal, 2008), ganador de la última IPC.

[7]Weighted And-Or Graphs.

2.4.4.6. Heurísticas no numéricas

En planificación basada en costes también se pueden aplicar otras heurísticas no numéricas, como las descritas en la parte de planificación clásica, aunque su aplicación se ha evaluado mínimamente. Por otro lado, existen algunos trabajos en los que se hace un esfuerzo por aprender automáticamente conocimiento que permita obtener planes de calidad, como (Pérez, 1996; Upal and Elio, 2000; Borrajo and Veloso, 1997). Sin embargo, es difícil generalizar información numérica, por lo que resulta complicado representar explícitamente información relacionada con los costes en el conocimiento que se aprende.

2.5. Resumen

En este capítulo se ha presentado una revisión del estado de la cuestión en planificación automática independiente del dominio. En esta revisión se ha hecho especial énfasis en los trabajos relacionados con planificación mediante búsqueda heurística progresiva, dado que las técnicas desarrolladas para este tipo de planificación conforman la base del trabajo desarrollado en esta tesis.

La planificación mediante búsqueda heurística progresiva se basa en la aplicación de un algoritmo de búsqueda genérico que se combina con distintos mecanismos heurísticos independientes del dominio orientados a guiar la búsqueda. El espacio de estados es el que induce directamente el problema de planificación. En los últimos años, este esquema ha demostrado ser bastante efectivo, escalando mejor que técnicas anteriores que llevaban a cabo procesos de búsqueda no informados o hacían búsqueda en otros espacios de estados.

El éxito de la planificación mediante búsqueda heurística se debe en gran medida al desarrollo de nuevas heurísticas con la potencia suficiente para guiar con más éxito al algoritmo de búsqueda. Principalmente se han desarrollado heurísticas de tres tipos: heurísticas numéricas, heurísticas de poda/ordenación y heurísticas *lookahead*, que consisten en la aplicación sucesiva de las acciones que marca una política por lo general imperfecta, con el objetivo avanzar rápidamente hacia estados más cercanos a un estado meta. Normalmente, varias heurísticas de distintos tipos se combinan en un mismo algoritmo de búsqueda. Respecto a los algoritmos de búsqueda, los esquemas con más éxito aplican búsqueda local o variaciones de búsqueda tipo *mejor primero*.

La efectividad de la búsqueda heurística en planificación clásica se ha venido demostrando en las distintas IPCs, que se vienen organizando desde el año 1998. Hoy en día se puede afirmar que unos de los mejores esquemas para realizar planificación sub-óptima es la búsqueda heurística. Sin embargo, para planificación óptima respecto a la longitud de los planes parecen más adecuados los planificadores SAT.

La búsqueda heurística se puede aplicar tanto en planificación clásica como en planificación basada en costes. En los últimos años se han desarrollado algunas heurísticas numéricas que tienen en cuenta los costes de las acciones. En realidad, desde el punto de vista conceptual, el espacio de estados es exactamente el mismo en ambos casos. Sin embargo, el problema a resolver es ligeramente distinto, ya que en planificación basada en costes el objetivo es minimizar el coste total del plan, y no el número de operadores que

contiene. Desde el punto de vista de la representación, el hecho de utilizar PDDL, da lugar a espacios de estados distintos, ya que el coste acumulado del camino se representa en cada estado como una variable numérica, junto con todas las variables numéricas que forman parte de la métrica, y cuya combinación da lugar a este coste acumulado. Sin embargo, no es realmente necesario representar el coste acumulado en los estados, ya que se puede calcular sin más que sumar el coste de las acciones del camino desde el estado inicial al estado correspondiente.

Respecto a la planificación basada en costes, la aplicación de distintos algoritmos de búsqueda se ha evaluado mínimamente. En relación con las heurísticas desarrolladas, algunas, como la de METRIC-FF son claramente mejorables, dado que no tienen en cuenta los costes hasta el final. Por otro lado, la definición de heurísticas no sigue un procedimiento común. Algunas heurísticas numéricas se han definido matemáticamente, mientras que otras se han definido proceduralmente, lo cual hace relativamente complicado entender las diferencias entre ellas. Finálmente, el éxito de la búsqueda heurística se debe a una combinación acertada de varias heurísticas con un algoritmo de búsqueda. Sería útil poder determinar qué heurísticas son más útiles y qué algoritmos son más adecuados. En esta tesis se tratará de aportar un poco de luz a estos aspectos.

Parte II

Heurísticas

Capítulo 3

La heurística h_{level}

En este capítulo se presenta la heurística h_{level}. Esta heurística está diseñada para problemas de planificación basada en costes y se basa en una construcción del grafo de planificación relajado en niveles incrementales de coste. Todas las heurísticas basadas en grafos de planificación tienen características comunes. Por este motivo, el capítulo comienza con una descripción general de los procedimientos que se utilizan para calcular heurísticas a partir de grafos de planificación relajados (*RPGs*). Esta generalización permitirá determinar posteriormente cuál es la diferencia entre el algoritmo para calcular h_{level} y los algoritmos que utilizan las heurísticas previas que también se basan en *RPGs*.

3.1. Generalización de los algoritmos basados en $RPGs$

En general, todas las heurísticas que utilizan algoritmos basados en *RPGs* deben definir los siguientes aspectos:

1. Aspectos relacionados con la expansión del *RPG*:
   - **Niveles de proposiciones**: cómo se calcula cada nivel de proposiciones.
   - **Niveles de acciones**: cómo se calcula cada nivel de acciones.
   - **Condición de éxito**: bajo qué condiciones el algoritmo acaba con éxito.
   - **Condición de fallo**: bajo qué condiciones el algoritmo acaba con fallo.
2. Aspectos relacionados con la extracción del plan relajado: cómo se extrae el plan relajado del *RPG*.

Respecto a la expansión del *RPG*, cabe destacar que en planificación clásica en el *RPG* los niveles de proposiciones contienen todas las proposiciones generadas hasta el momento, y los niveles de acciones todas las acciones aplicables dada la capa de proposiciones previa. Sin embargo, en planificación basada en costes esto no siempre es así. Hay algoritmos que en las capas de proposiciones no introducen todas las proposiciones generadas hasta el momento o en las capas de acciones no introducen todas las acciones aplicables. Respecto a

la condición de éxito ocurre lo mismo. La condición de éxito clásica es encontrar un nivel de proposiciones con las metas. Sin embargo, en planificación basada en costes hay algoritmos que continúan con el proceso de expansión aún cuando las metas ya aparecen en un nivel de proposiciones.

Respecto al algoritmo de expansión, lo usual es ir hacia atrás de forma avara (es decir, sin hacer *backtracking*) partiendo de la última capa de proposiciones y seleccionando acciones para conseguir las (sub)metas. La Figura 3.1 muestra el algoritmo generalizado. Inicialmente el plan relajado, RP, es la lista vacía, y las metas a ser satisfechas son las metas del problema. A continuación, en el paso (1) se selecciona una meta g a resolver, y en el paso (2) se selecciona una acción a que consiga o soporte la meta seleccionada, g, es decir, una acción cuyos efectos positivos añadan g. La acción elegida se inserta en el plan relajado y el conjunto de metas a resolver se actualiza eliminando g y añadiendo todas las precondiciones de la acción seleccionada a que no se encuentren en el estado inicial. El algoritmo termina cuando el conjunto de metas a resolver es vacío.

```
function extraer_RP(G, RPG, I)
let RP = [];                                        /* plan relajado inicial */
let metas = G;                                      /* metas del problema */
while metas ≠ ∅ do
    (1) Seleccionar una meta g ∈ metas
    (2) Seleccionar una acción a del RPG que soporte g
    metas = metas − {g}
    RP = {a} ∪ RP
    forall p ∈ pre(a), p ∉ I do
        metas = metas ∪ {p}                         /* (sub)metas */
    end for
end while
return RP
```

Figura 3.1: Extracción del plan relajado: algoritmo generalizado.

Cada instanciación concreta del algoritmo de la Figura 3.1 debe determinar: (a) qué meta se selecciona en el paso (1) y (b) que acción se selecciona en el paso (2). Lo usual es seleccionar la (sub)meta que aparece más tarde en el RPG y seleccionar la acción para conseguirla de forma heurística.

La decisión de hacer un proceso de extracción avaro es exclusivamente computacional. Si se contara con el plan relajado óptimo la heurística sería admisible, dado que el hecho de que la heurística se calcule a partir de un plan relajado óptimo es una condición suficiente (aunque no necesaria) para garantizar la admisibilidad de la misma. Sin embargo, como se comentó anteriormente (sección 2.3.3.1, página 25), obtener el plan relajado óptimo es un problema *NP-hard* desde el punto de vista teórico.

En cualquier caso, dada la forma usual de construir el RPG en planificación clásica, hay casos en los que ni aún haciendo *backtracking* se podría conseguir el plan relajado óptimo.

Esto ocurre por dos razones:

- En el *RPG* se incluyen en **paralelo** todas las acciones que se pueden ejecutar dada una capa de proposiciones.
- La condición de éxito se cumple cuando todas las metas están en una capa proposicional.

La Figura 3.2 muestra un ejemplo de un dominio ficticio instanciado en el que el estado inicial contiene la proposición p, y el objetivo es conseguir las proposiciones q, r y s. El grafo de planificación relajado para este problema se muestra en la Figura 3.3. Las metas se consiguen en la capa proposicional P_1, así que ésta es la capa en la que finalizaría el algoritmo de expansión. Al hacer la extracción se obtiene el plan relajado: a_1, a_2, a_3. Esté plan relajado es el único que contiene el grafo. Sin embargo, si se expandiera una capa más se podría obtener el plan relajado a_1, a_4 que tiene menor longitud y es el óptimo para este problema.

```
Acciones del dominio:
   a1:pre(a1)={p},          a3:pre(a3)={p},          Problema:
      add(a1)={q},             add(a3)={s},             I={p}
   a2:pre(a2)={p},          a4:pre(a4)={q},          G={q,r,s}
      add(a2)={r},             add(a4)={r,s},
```

Figura 3.2: Ejemplo de problema en el que el *RPG* clásico no contiene el *RP* óptimo.

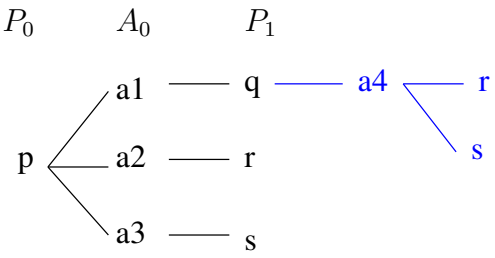

Figura 3.3: *RPG* para el ejemplo de la Figura 3.2.

Una forma de obligar a que el *RPG* contenga el plan relajado óptimo es expandirlo hasta el punto fijo, es decir hasta que dos capas proposicionales consecutivas contengan los mismos hechos. Otra forma, seguramente más eficiente, es realizar un proceso de expansión seguido de una extracción y seguir expandiendo el grafo utilizando la longitud del plan relajado obtenido como límite superior del número de capas.

Un punto importante del algoritmo generalizado de extracción en la Figura 3.1 es que cuando se selecciona una acción a, ésta se incluye en el plan relajado mediante una unión.

Esto supone que el plan relajado no tiene acciones duplicadas, lo cual tiene sentido puesto que dado el problema relajado los hechos que se han conseguido ya no se borran nunca, por lo que no tendría sentido aplicar de nuevo una acción que ya se aplicó. De esta forma el proceso captura algunas interacciones positivas entre (sub)metas: si para dos (sub)metas distintas se selecciona la misma acción, ésta y las acciones para conseguir sus precondiciones sólo aparecerán una vez.

Por otro lado, en este algoritmo, al representar el plan relajado como un conjunto se pierde el orden de las acciones que contiene. Para que este conjunto fuera en realidad un plan relajado habría que ordenarlo respetando las restricciones causales. El orden no es relevante para el cálculo de las heurísticas, puesto que para calcularlas es suficiente con conocer el número de acciones que contiene el plan relajado (o sus costes en el caso de planificación basada en costes). Sin embargo, sí es relevante para calcular las acciones *helpful*. Estas acciones en planificación clásica se definen como las acciones del dominio aplicables en el estado evaluado que consiguen hechos que son (sub)metas de la extracción a nivel P_1, es decir que consiguen hechos que son precondiciones de otras acciones del plan relajado. Por este motivo, el plan relajado se suele implementar como una lista sin duplicados. Para que en esta lista se respete el orden causal, lo que se hace seleccionar antes en el paso (1) del algoritmo aquellas (sub)metas que aparecen por primera vez en niveles superiores del *RPG*. En cualquier caso, aún representando el *RP* como un conjunto las acciones *helpful* se podrían calcular *a posteriori*.

3.2. Una heurística basada en el algoritmo Dijkstra: la heurística h_{level}

La heurística que se propone en este apartado en una heurística numérica basada en grafos de planificación relajados. En este caso, el grafo de planificación relajado se construye siguiendo la idea del algoritmo Dijsktra en el sentido de que se aplican primero las acciones cuyo coste acumulado es menor. Esto da lugar a un grafo de planificación en niveles incrementales de coste.

3.2.1. Descripción procedural

El algoritmo para construir el *RPG* en niveles incrementales de coste se detalla en la Figura 3.4. Lo que caracteriza a este algoritmo es que los niveles de acciones contienen únicamente las acciones aplicables con coste acumulado mínimo.

El algoritmo de la Figura 3.4 tiene como entradas las metas del problema ($\mathcal{G}$), el estado a evaluar (s) y las acciones del dominio relajadas ($\mathcal{A}^+$). Este algoritmo construye un *RPG* en el que los niveles llevan asociado un coste. Este coste viene dado por el coste acumulado, o coste de aplicación, *coste_app*, de las acciones que se encuentran en el nivel inmediatamente anterior, y representa una estimación de lo que cuesta conseguir las proposiciones que aparecen por primera vez en esa capa. El coste asociado a un nivel se denota como $limite\_coste_i$ (inicialmente $limite\_coste_0 = 0$). El algoritmo mantiene una lista abierta de acciones aplicables (*AbiertaApp*) ordenada de menor a mayor coste . En cada nivel i, todas las nuevas acciones aplicables se incluyen en *AbiertaApp*, junto con su coste acumulado.

function expansion_RPG_hlevel $(\mathcal{G}, s, \mathcal{A}^+)$
let $i = 0$;
let $P_0 = s$;
let $AbiertaApp = \emptyset$;
let $limite\_coste_0 = 0$;
while $G \nsubseteq P_i$ **do** /* condición de éxito */
  $AbiertaApp = AbiertaApp \cup \{a \in \mathcal{A}^+ \setminus \cup_{j<i} A_j \mid pre(a) \subseteq P_i\}$
  for cada acción nueva en $AbiertaApp$ **do**
    $coste\_app(a) = coste(a) + limite\_coste_i$
  end for
  $A_i = \{a \mid a \in \arg\min_{a \in AbiertaApp} coste\_app(a)\}$ /* niveles de acciones */
  $limite\_coste_{i+1} = \min_{a \in AbiertaApp} coste\_app(a)$
  $P_{i+1} = P_i \cup_{a \in A_i} add(a)$ /* niveles de proposiciones */
  if $AbiertaApp = \emptyset$ **then return** fail /* condición de fallo*/
  $AbiertaApp = AbiertaApp - A_i$
  $i = i + 1$
end while
return $P_0, A_0, ..., P_{i-1}, A_{i-1}, P_i$

Figura 3.4: Algoritmo de expansión de *RPG* para calcular h_{level}.

El coste acumulado de una acción es la suma del coste propio de la acción $coste(a)$ más el coste de conseguir sus precondiciones. El algoritmo considera que el coste de conseguir las precondiciones cada **nueva** acción aplicable es el límite de coste de la capa de proposiciones anterior, $limite\_coste_i$, ya que esta capa de proposiciones es la que permite por primera vez que la acción sea aplicable. En cada iteración, las acciones que tienen coste acumulado mínimo en $AbiertaApp$ se extraen de esta lista y se incluyen en el siguiente nivel de acciones, A_i. Es decir, sólo se incluyen las acciones con mínimo coste acumulado. La siguiente capa de proposiciones P_{i+1} se define como viene siendo usual en los algoritmos que generan grafos de planificación relajados: se incluyen en ella todos los efectos de las acciones de la capa A_i más todas las proposiciones previas (las que están en P_i). El proceso continúa hasta que todas las metas se encuentran en un nivel de proposiciones.

En resumen, y considerando los cuatro aspectos que hay que definir en los algoritmos de expansión del *RPG*, en el caso de h_{level}:

- **Niveles de proposiciones**: cada nivel de proposiciones contiene el conjunto de proposiciones que se ha conseguido hasta el momento: $P_{i+1} = P_i \cup add(a), \forall a \in A_i$. El primer nivel de proposiciones contiene las proposiciones en el estado que se está evaluando: $P_0 = s$.
- **Niveles de acciones**: cada nivel de acciones contiene el conjunto de acciones con coste acumulado mínimo en la lista *AbiertaApp*. Esta lista contiene las acciones aplicables dada la capa actual de proposiciones junto con las acciones aplicables en capas previas que aún no han sido aplicadas por no tener coste acumulado mínimo. En la lista *AbiertaApp* únicamente se incluyen acciones nuevas, es decir, acciones que no aparecen

en ninguna capa del *RPG*. No es necesario considerar las acciones que ya se han aplicado en niveles previos, dado que sus efectos ya estarán incluidos en alguna capa de proposiciones previa (y puesto que el problema está relajado nunca se borran) y el coste acumulado de las acciones que ya están incluidas no se puede decrementar más, porque las acciones se aplican en orden creciente de coste. Por lo tanto, cada acción aparece a lo sumo una vez en el *RPG*.

- **Condición de éxito**: el algoritmo finaliza con éxito en un nivel i cuando todas las metas del problema se encuentran en ese nivel: $\mathcal{G} \subseteq P_i$. En ese caso, P_i es el último nivel del *RPG*, que denominaremos P_{final}.
- **Condición de fallo**: el algoritmo termina con fallo cuando la lista *AbiertaApp* se queda vacía. Cuando esto ocurre, no se pueden aplicar más acciones.

Una vez ejecutado el algoritmo de expansión se pueden dar varios casos:

- Que el algoritmo falle, en cuyo caso $h_{level}(\mathcal{G}, s) = \infty$.
- Que el *RPG* sólo contenga la primera capa de proposiciones, lo que quiere decir que las metas se encuentran en el estado evaluado, por lo que $h_{level}(\mathcal{G}, s) = 0$.
- Que el *RPG* tenga más de una capa, en cuyo caso se aplica el algoritmo de extracción para conseguir un plan relajado secuencial que permita calcular la estimación heurística.

La instanciación del algoritmo de extracción generalizado (Figura 3.1, página 62) en el caso de h_{level} es prácticamente la misma que aplica METRIC-FF:

- En el paso (1) del algoritmo (selección de la (sub)meta a resolver) se seleccionan primero las (sub)metas que aparecen por primera vez en el nivel de proposiciones con **mayor** índice.
- En el paso (2) del algoritmo (selección de la acción que genera la (sub)meta seleccionada), se selecciona de entre las acciones que aparecen en el nivel de acciones con **menor** índice, aquella con menor *dificultad*. En FF esto se hace así siempre excepto en algunos casos. Concretamente, la excepción se produce cuando se dan las siguientes circunstancias, ilustradas en la Figura 3.5:
  - en una iteración anterior se seleccionó como (sub)meta a resolver g_1 en el paso (1) y la acción seleccionada en el paso (2) para resolver esa (sub)meta fue a_1, y
  - la (sub)meta actual es g_2, y
  - la acción a_1 añade g_2 (pero, de entre las acciones que generan g_2, a_1 no es la acción que aparece en el nivel de menor índice y con menor dificultad), y
  - g_2 aparece por primera vez en el *RPG* en el mismo nivel en que aparece g_1 por primera vez, o en un nivel inmediatamente anterior. Esto sólo se tiene en cuenta en FF, y no en METRIC-FF.

En este caso, la acción que se selecciona para resolver g_2 no es la de menor índice y dificultad, sino a_1. En realidad, se puede considerar que g_2 es un *efecto secundario* de haber elegido antes g_1 que g_2 en el paso (1), y de haber elegido la acción a_1 en el paso (2) para resolver g_1. La consideración de este tipo de efectos secundarios, se hace con el objetivo de tener en cuenta más interacciones positivas entre metas a la hora de generar el plan relajado, y quizás conseguir un plan relajado más cercano al óptimo.

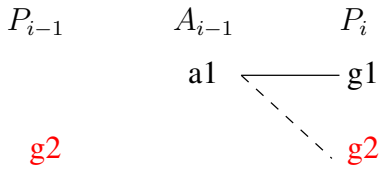

Figura 3.5: Tratamiento de los efectos secundarios en la extracción.

Si los efectos secundarios no se consideraran de ninguna manera, la decisión (1) del algoritmo de extracción influiría sólo en el orden en que las acciones aparecen en el plan relajado, dado que cada (sub)meta se resuelve de forma independiente. Como se ha mencionado anteriormente el orden de las acciones en el plan relajado es indiferente para el calculo del valor heurístico.

En el caso de h_{level} (y al contrario que en METRIC-FF), la heurística de la *dificultad* tiene en cuenta los costes de las acciones de la siguiente manera:

$$\text{dificultad}_{h_{level}}(a) = \sum_{p \in pre(a)} \text{mín}\{limite\_coste_i | p \in P_i\} \tag{3.1}$$

Es decir, la dificultad de una acción es la suma de los límites de coste del primer nivel en que aparece cada una de sus precondiciones (los límites de coste crecen con las capas, por lo tanto el mínimo es el de la capa de menor índice que contiene la precondición).

Por último, una vez extraído el RP la heurística se calcula como la suma de costes de las acciones que éste contiene:

$$h_{level}(\mathcal{G}, s) = \sum_{a_i \in RP} coste(a_i)$$

Una posibilidad a la hora de diseñar una heurística de este tipo es utilizar un proceso de extracción que no sea avaro. Por ejemplo, se podría plantear hacer *backtracking* con el objetivo de encontrar el plan relajado óptimo o uno cercano a éste. Esta opción se ha descartado por varios motivos. Como se explicó en la sección 3.1 (página 63), no está siempre garantizado que el RPG contenga el plan relajado óptimo terminando en la primera capa que contiene las metas. Por esto, en algunas ocasiones, ni aún haciendo *backtracking* se podría conseguir este plan, con lo que no se podría garantizar siempre la admisibilidad de la heurística. El principal motivo es que el grafo relajado y el proceso posterior de extracción no sólo ignoran las interacciones negativas entre (sub)metas, sino también algunas de las

interacciones positivas. Las soluciones comentadas en la sección 3.1 (página 63) para forzar a que el *RPG* contenga siempre el plan relajado óptimo incrementan el tiempo de cálculo de la heurística. En el peor caso, la tarea es *NP-hard* (Bylander, 1994). Por otro lado, si no se toma ninguna de estas alternativas, incrementar el tiempo de cálculo de la heurística haciendo *backtracking* no parece tener mucho sentido si no se espera obtener una heurística ni admisible, ni más precisa. Aunque se podría valorar experimentalmente, la heurística seguiría ignorando interacciones negativas y algunas de las positivas, con lo que parece difícil justificar más precisión.

3.2.2. Ejemplo: cálculo de h_{level}

En este apartado se muestra mediante un ejemplo el procedimiento para calcular h_{level}. La Figura 3.6 muestra un dominio artificial con seis acciones instanciadas, su coste, precondiciones y efectos positivos. También muestra un problema en este dominio en el que el estado inicial contiene la proposición p y las metas son $\{t, k\}$.

```
Acciones del dominio:
   a1:pre(a1)={p},
      add(a1)={q, r},
      coste(a1)=15

   a2:pre(a2)={p},
      add(a2)={s},
      coste(a2)=20

   a3:pre(a3)={q, r, s},
      add(a3)={t},
      coste(a3)=10

   a4:pre(a4)={t},
      add(a4)={k},
      coste(a4)=2

   a5:pre(a5)={q, r},
      add(a5)={k},
      coste(a5)=20

   a6:pre(a6)={s},
      add(a6)={k},
      coste(a6)=60
Problema:
   I={p}
   G={t, k}

Coste óptimo: 47
```

Figura 3.6: Ejemplo artificial de un problema en un dominio relajado.

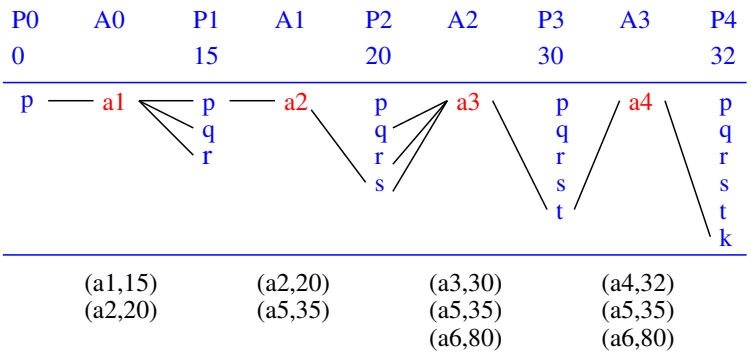

Figura 3.7: Ejemplo del *RPG* para h_{level}.

La Figura 3.7 muestra cómo se expande el *RPG* correspondiente a h_{level} para el ejemplo de la Figura 3.6. El número asociado a cada capa de proposiciones P_i es el límite de

coste de esa capa, $limite\_coste_i$. Debajo de cada capa se muestra el contenido de la lista abierta, *AbiertaApp*, para esa iteración. La lista abierta se inicializa al conjunto vacío, y $limite\_coste_0$ se inicializa a cero. La primera capa de proposiciones, P_0, contiene la única proposición del estado inicial, p. Después, las acciones aplicables en P_0 se incluyen en *AbiertaApp*: a_1 con coste acumulado de 15 y a_2 con coste acumulado de 20. En este momento, la acción con menor coste acumulado de *AbiertaApp* es a_1. Esta acción genera la siguiente capa de acciones, A_0. El límite de coste de esta capa, es $limite\_coste_1$ (15), el coste acumulado de la acción aplicada. La aplicación de a_1 en A_0 da lugar a la siguiente capa de proposiciones, P_1, que contiene, por lo tanto, q y r. Dada la capa P_1, existe una nueva acción aplicable: a_5 (sus precondiciones son q y r. Esta acción se añade a *AbiertaApp*. La siguiente capa de proposiciones, A_1, contiene la acción de *AbiertaApp* con menor coste acumulado. En este caso es a_2, con un coste acumulado de 20. La acción a_5 también es aplicable, pero no se selecciona porque su coste acumulado es mayor (35=15+20). El proceso continua hasta que todas las metas (t y k) se encuentran en una capa de proposiciones. En este caso, la capa P_4. Una vez terminado el grafo relajado se extrae el plan relajado. En este ejemplo el plan relajado es $[a_1, a_2, a_3, a_4]$, y el valor de la heurística $15 + 20 + 10 + 2 = 47$, que es el óptimo.

3.3. Diferencias algorítmicas con otras heurísticas basadas en *RPGs*

Desde el punto de vista algorítmico, las principales diferencias con las heurísticas numéricas previas basadas en grafos relajados de planificación (las heurísticas de METRIC-FF y SAPA) y el procedimiento para expandir el grafo de SIMPLANNER son:

- Al contrario que en la heurística de METRIC-FF, en este caso el grafo de planificación relajado es sensible a los costes de las acciones.
- En relación con las heurísticas de SAPA, la principal diferencia es que en este caso la expansión del grafo de planificación termina en la primera capa en que se encuentran las metas; es decir, no es necesario construir el grafo hasta el punto fijo.
- En relación con SIMPLANNER, que también se construye en niveles incrementales de costes, la diferencia radica en la forma de calcular los costes acumulados. Por otro lado, en SIMPLANNER, el *RPG* no se utiliza para calcular una heurística numérica, sino con otros propósitos, y por lo tanto no hay proceso de extracción.

Existen también otras diferencias más sutiles. Para clarificarlas se hace a continuación una descripción de los procedimientos de METRIC-FF, SAPA y SIMPLANNER en función de los algoritmos generalizados que se han introducido en este capítulo.

3.3.1. Heurística de METRIC-FF

En la generación del grafo de planificación relajado, METRIC-FF resuelve los cuatro aspectos mencionados de la siguiente manera:

- **Niveles de proposiciones**: el primer nivel es igual al estado a evaluar, $P_0 = s$, pero en general, cada nivel de proposiciones contiene el conjunto de literales que se han conseguido hasta el momento: $P_{i+1} = P_i \cup add(a), \forall a \in A_i$, donde A_i es el nivel de acciones inmediatamente anterior.
- **Niveles de acciones**: cada nivel de acciones contiene el conjunto de acciones aplicables en la capa actual $i : A_i = \{a \in \mathcal{A} \mid pre(a) \subseteq P_i\}$. Aunque en el algoritmo descrito en (Hoffmann, 2003) se consideran todas las acciones aplicables, en realidad, y dado que las capas de proposiciones son monótonas, no es necesario considerar las acciones que se han aplicado en niveles anteriores debido a que sus efectos ya se encuentran en capas de proposiciones anteriores. Por lo tanto, aunque esto no se expresa explícitamente en el algoritmo, y al igual que en h_{level}, se puede considerar que cada acción aparece como mucho una vez en el *RPG*.
- **Condición de éxito**: la construcción del *RPG* finaliza con éxito cuando después de construir el nivel de proposiciones i, todas las metas del problema están incluidas en él: $\mathcal{G} \subseteq P_i$. En este caso, P_i, es el último nivel del *RPG*.
- **Condición de fallo**: el algoritmo termina con fallo cuando en dos niveles de proposiciones consecutivos aparecen las mismas proposiciones, $P_{i+1} = P_i$ y no aparecen todas las metas, es decir, se ha llegado al punto fijo sin conseguir las metas del problema.

A la hora de extraer el plan relajado del *RPG*, METRIC-FF instancia el algoritmo de extracción de la siguiente manera. En el paso (1) seleccionan primero las (sub)metas que aparecen por primera vez en un nivel de proposiciones posterior. Sin tener en cuenta la consideración de *efectos secundarios* esta decisión influiría sólo en el orden en el que las acciones aparecen en el plan relajado, dado que cada meta se resuelve de forma independiente; en el paso (2) selecciona la acción que aparece en el nivel de acciones más cercano al estado inicial, es decir, la que aparece en el nivel de acciones con menor índice. Cuando existen varias acciones en este nivel los empates se resuelven utilizando la heurística de la dificultad; es decir, seleccionando la acción con menor dificultad: aquélla para la que la suma de los índices del primer nivel en que aparece cada precondición es menor.

$$dificultad(a) = \sum_{q \in pre(a)} \min_{i \in [0..final]} \{i | q \in P_i\} \tag{3.2}$$

donde $final$ es el índice de la última capa del *RPG*.

3.3.2. Ejemplo: cálculo de h_{mff}

Retomando el ejemplo de la Figura 3.6, el grafo de planificación relajado que construyen tanto METRIC-FF como FF se muestra en la Figura 3.8. En este ejemplo, el primer nivel de proposiciones, P_0, contiene la única proposición del estado inicial. La primera capa de acciones, A_0, contiene las acciones aplicables en el estado inicial: a_1 y a_2. En la siguiente capa de proposiciones, P_1, se incluyen todos los efectos positivos de las acciones en A_0: q y r por a_1, y $s4$ por a_2. Las acciones que se pueden aplicar a partir de P_1, a_3, a_5 y a_6, se incluyen en el siguiente nivel de acciones, A_1, y sus efectos positivos en P_2. Dado que P_2

contiene las metas del problema (t y k), el proceso de expansión del grafo de planificación relajado termina con éxito. De este grafo de planificación se extrae un plan relajado como sigue:

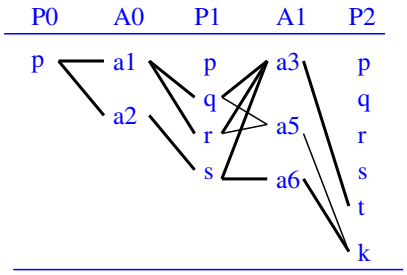

Figura 3.8: Ejemplo del RPG para h_{mff}.

1. Se elige arbitrariamente una de las dos metas, dado que ambas aparecen por primera vez en el mismo nivel de proposiciones. Supongamos que se elige k. Existen dos acciones en el grafo que consiguen k: a_5 y a_6. Ambas acciones aparecen en el mismo nivel de acciones, por lo tanto se selecciona aquélla con menor dificultad. La dificultad de a_5 es $2 = 1+1$ (suma de los índices del primer nivel de proposiciones en el que aparece cada precondición). La dificultad de a_6 es 1 (dado que la única precondición es s, situada en P_1). El plan relajado actual es por lo tanto $RP = [a_6]$ y la precondición de a_6 se incorpora como nueva (sub)meta: $metas = \{t, s\}$.

2. A continuación se selecciona la (sub)meta que aparece en el mayor nivel de proposiciones: t. La única acción que consigue t es a_3, que se incluye en el plan relajado: $RP = [a_3, a_6]$. Sus precondiciones se incluyen como nuevas (sub)metas: $metas = \{s, p, q, r\}$.

3. En este momento todas las (sub)metas pertenecen al mismo nivel de proposiciones y se selecciona una de ellas arbitrariamente. Supongamos que se selecciona s. La única acción que consigue s, a_2, se incluye en el plan relajado, que queda: $RP = [a_2, a_3, a_6]$. La única precondición de a_2 ya pertenece al conjunto de (sub)metas, así que no es necesario insertarla en él.

4. Finalmente, las metas pendientes, q y r se resuelven insertando a_1 en el plan relajado. En este momento el nuevo conjunto de (sub)metas sólo contiene p, que pertenece al estado inicial, por lo tanto, el proceso termina y el plan relajado final es $RP = [a_1, a_2, a_3, a_6]$.

El valor de la heurística se calcula como $h_{mff}(\mathcal{G}, s) = coste(a_1) + coste(a_2) + coste(a_3) + coste(a_6) = 15 + 20 + 10 + 60 = 105$ (compárese con el óptimo que era 47).

3.3.3. Heurísticas de SAPA

En relación con el algoritmo de construcción del RPG, en las heurísticas de SAPA, éste tiene las siguientes características:

- **Niveles de proposiciones**: como en el caso de h_{mff} y h_{level}, cada nivel de proposiciones contiene el conjunto de literales conseguidos hasta el momento: $P_{i+1} = P_i \cup add(a), \forall a \in A_i$. La diferencia es que, en este caso, se almacena para cada proposición el coste asociado a ella.
- **Niveles de acciones**: cada capa de acciones contiene el conjunto de acciones aplicables dadas las proposiciones de la capa inmediatamente anterior. Las acciones que se han aplicado en niveles previos se deben reconsiderar si su coste acumulado se decrementa. Esto último marca una diferencia importante respecto a h_{mff} y h_{level} puesto que en estos dos casos no es necesario reconsiderar acciones aplicadas en niveles previos.
- **Condición de éxito**: la expansión del grafo de planificación termina cuando los costes de las proposiciones se mantienen estables en dos niveles de proposiciones consecutivos (es decir, se ha llegado a un punto fijo). En este caso, al contrario que en h_{mff} y h_{level}, es necesario siempre llegar al punto fijo, dado que los costes de las proposiciones pueden disminuir a medida que se añaden más niveles al grafo. Dado que llegar al punto fijo puede ser costoso computacionalmente en algunos casos, en (Bryce and Kambhampati, 2007) se deja la opción de elegir la condición de terminación. Por ejemplo, se puede elegir que la expansión del grafo termine en el primer nivel en el que se encuentran las metas, o construir n niveles adicionales haciendo un *n-lookahead*. El caso de ∞-lookahead, es decir de llegar al punto fijo, es el que intuitivamente debería producir *mejores* estimaciones. En este caso el algoritmo termina con éxito en un nivel i cuando el P_i contiene todas las metas del problema y los costes son estables: $\mathcal{G} \subseteq P_i$ y $\forall p \in P_i, coste(p, i) = coste(p, i-1)$, donde $coste(p, i)$ representa el coste de la proposición p en el nivel P_i. En caso de que esta condición sea cierta, P_i es el último nivel del *RPG*.
- **Condición de fallo**: el algoritmo falla cuando $P_{i+1} = P_i, \mathcal{G} \not\subseteq P_i$, y $\forall p \in P_i, coste(p, i) = coste(p, i-1)$; es decir, cuando se llega al punto fijo (los costes son estables) sin que se hayan encontrado todas las metas del problema.

Respecto al algoritmo de extracción, en estas heurísticas se decide qué meta seleccionar en el paso (1) de forma arbitraria. En el paso (2) SAPA elige la acción con un coste total más pequeño de entre aquéllas que consiguen la meta g. En caso de empate, es decir, si existen varias acciones que consiguen la meta con el mínimo coste, se selecciona una de ellas arbitrariamente.

3.3.4. Ejemplo: cálculo de $h_{sapa\text{-}max}$ y $h_{sapa\text{-}add}$

Las Figuras 3.9 y 3.10 muestran el *RPG* que se construye para calcular $h_{sapa\text{-}max}$ y $h_{sapa\text{-}add}$ respectivamente, para el ejemplo de la Figura 3.6 (página 68). El valor al lado de cada proposición representa el coste de conseguir la proposición. El valor al lado de cada acción muestra el coste de la acción más el coste de conseguir sus precondiciones. En el caso de $h_{sapa\text{-}max}$, P_0 contiene la única proposición del estado inicial, p, con coste 0. La siguiente capa de acciones, A_0, contiene todas las acciones aplicables en P_0: a_1 y a_2 con sus respectivos costes: 15 y 20. Los efectos positivos de estas acciones (q y r para a_1, y s para

a_2) y sus costes se incluyen en la siguiente capa de proposiciones, P_1. En este momento las nuevas acciones aplicables en A_1 son a_3, a_5 y a_6. Sus efectos se introducen en P_2. Los costes de las acciones se calculan haciendo agregación de los costes de sus precondiciones vía un máximo. Por ejemplo, el coste de la acción a_3 es la el coste máximo de sus precondiciones, 20 (dado por s) más el coste de la acción, 10. La capa de proposiciones P_2 contiene todas las metas del problema (t y k), pero el proceso de propagación no termina aquí, sino que continúa hasta que los costes son estables. Por eso, ahora se genera la siguiente capa de acciones, A_2. En esta capa se aplica la acción a_4. La aplicación de a_4 consigue la meta k con coste 32. De esta forma, a_4 es la acción que proporciona el mínimo coste para la meta k.

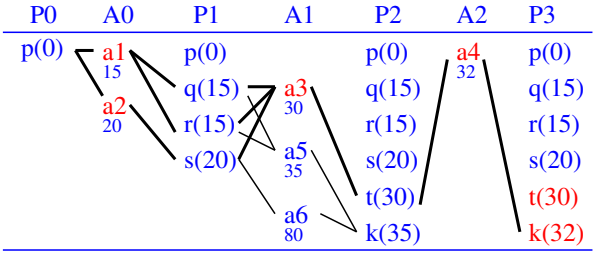

Figura 3.9: Ejemplo del RPG para $h_{sapa-max}$.

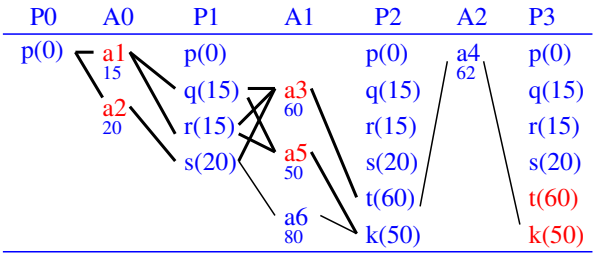

Figura 3.10: Ejemplo del RPG para $h_{sapa-add}$.

El proceso de extracción selecciona para cada (sub)meta la acción que la consigue con el mínimo coste. Así, se selecciona a_4 para k, a_3 para t, a_2 para s y a_1 para q y r. El plan relajado resultante es $[a_1, a_2, a_3, a_4]$ y el valor de la heurística en este caso $15+20+10+2 = 47$, que vuelve a ser el óptimo.

La explicación para $h_{sapa\text{-}add}$ es similar. Los niveles de proposiciones y acciones se calculan de la misma manera que para $h_{sapa\text{-}add}$, pero los costes de las acciones de calculan haciendo agregación de los costes de sus precondiciones mediante una suma. Por ejemplo, el coste de la acción a_3 en la capa A_1 es 60 porque este valor es la suma del coste de todas sus precondiciones $15(q) + 15(r) + 20(s) = 50$ más el coste de la acción (10). El proceso de extracción selecciona para cada (sub)meta la acción que genera la (sub)meta con el mínimo coste propagado. Así, se selecciona a_5 para k, a_3 para t, a_2 para s y a_1 para q y r. El plan relajado resultante en este caso es $[a_1, a_2, a_3, a_5]$, y el valor de la heurística

$15 + 20 + 10 + 20 = 65$.

3.3.5. Procedimiento de SIMPLANNER

El algoritmo descrito en (Sapena and Onaindía, 2004) para expandir el *RPG* se puede transformar fácilmente en el algoritmo de h_{level} descrito en la Figura 3.4. En realidad, la única diferencia entre lo que calculan ambos algoritmos es que el coste de las precondiciones de cada acción se calcula agregando el coste de cada precondición a través de una suma, en lugar de un máximo, es decir, utilizando $\sum_{q \in pre(a)} \text{mín}\{limite\_coste_i | q \in P_i\}$ en lugar de $limite\_coste_i$:

$$coste\_app(a) = coste(a) + \sum_{q \in pre(a)} \text{mín}\{limite\_coste_i | q \in P_i\}$$

El algoritmo original, presenta otras diferencias que se explican a continuación, pero en realidad el resultado es equivalente al que consigue el algoritmo de la Figura 3.4. Las diferencias son:

- Se mantiene una lista de *proposiciones programadas* en lugar de una lista de acciones aplicables, de manera que sólo las *proposiciones programadas* con coste mínimo se incluyen en la siguiente capa de proposiciones. El límite de coste asociado a esta capa es precisamente este coste mínimo. Como en el algoritmo de la Figura 3.4, el coste de cada proposición se calcula como el coste acumulado (*coste_app*) de las acciones que la generan con mínimo coste acumulado.
- Cada nivel de acciones, A_i, se calcula incluyendo sólo aquellas acciones para las que se cumple que todas sus precondiciones q tienen un coste menor que el límite de coste de la capa i: $coste(q) < limite\_coste_i$. Esto lleva a incluir en la capa A_i acciones cuyos efectos positivos no se encuentran en la siguiente capa de proposiciones, P_{i+1}, debido a que en realidad estas acciones no se ejecutan en la capa A_i. La transformación al algoritmo de la Figura 3.4 implica situar estas acciones en la capa en la que se ejecutan en realidad. De esta forma el algoritmo de extracción se puede mantener igual.

El mismo proceso de extracción de h_{level} (incluida la heurística de la dificultad) permite extraer del *RPG* un plan relajado, y calcular, como en los demás casos, un valor heurístico como la suma de los costes de las acciones en este plan.

3.3.6. Ejemplo: procedimiento de SIMPLANNER

La Figura 3.11 muestra el *RPG* que se expande considerando el algoritmo de SIMPLANNER, para el ejemplo de la Figura 3.6. El valor que hay asociado a cada capa proposicional es el límite de coste asociado a ese nivel, $limite\_coste_i$. La lista abierta, *AbiertaApp*, se muestra debajo de cada nivel. El límite de coste de la primera capa, $limite\_coste_0$, es 15: el coste de la acción más barata que se puede aplicar en el estado inicial (a_1). La capa de acciones A_0 sólo contiene esa acción. La acción a_2 con coste 20, también es aplicable en

Construcción	Parada	Niveles de acciones (A_i)	Niveles de proposiciones (P_{i+1})	Propagación de costes
METRIC-FF	$\mathcal{G} \subseteq P_i$	aplicables en P_i	generadas por A_i	no
h_{level}	$\mathcal{G} \subseteq P_i$	**menor coste**	generadas por A_i	**acciones 1ª capa aplicable**
SAPA	$P_i = P_{i+1}$	aplicables en P_i	generadas por A_i	acc. y prop. suma/máximo
SIMPLANNER	$\mathcal{G} \subseteq P_i$	aplicables en P_i	menor coste	proposiciones suma

Tabla 3.1: Diferencias algorítmicas en la construcción del *RPG*.

el estado inicial, y por este motivo se incluye en *AbiertaApp*. La aplicación de a_1, añade q y r en P_1. Así, $limite\_coste_1$ es el coste de a_2, dado que esta acción es la menos costosa de *AbiertaApp*. La acción a_5 también es aplicable en este nivel, pero no se extrae de *AbiertaApp* porque su coste acumulado (50=15+15+20) es mayor que el de a_2. El proceso continúa hasta que todas las metas (en este ejemplo t y k) aparecen en una capa de proposiciones. Esta capa es P_4. Una vez construido el *RPG*, y siguiendo la forma en que h_{level} extrae el plan relajado, resultaría $[a_1, a_2, a_5, a_3]$. Por lo tanto, el valor de la heurística sería $15 + 20 + 20 + 10 = 65$.

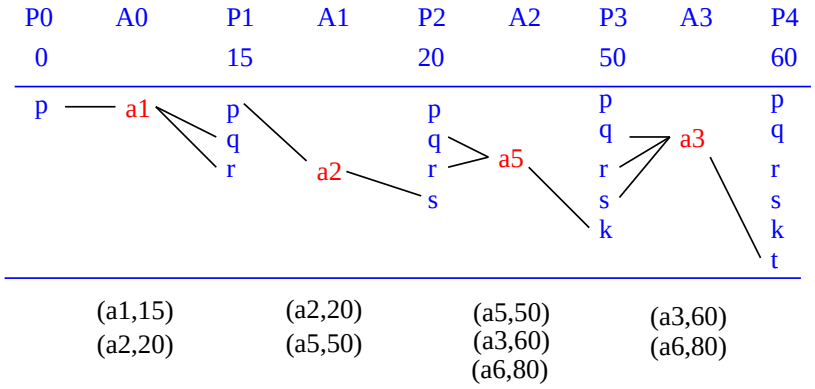

Figura 3.11: Ejemplo del *RPG* según SIMPLANNER.

3.3.7. Resumen de las diferencias entre los algoritmos

Las Tablas 3.1 y 3.2 muestran de forma resumida las diferencias entre los algoritmos que se utilizan para expandir el *RPG* y para posteriormente extraer un plan relajado, respectivamente.

Así, METRIC-FF y SAPA expanden el plan relajado de la misma forma con las siguientes diferencias: SAPA lleva a cabo un proceso de propagación de costes de acciones de proposiciones (efectos) y de proposiciones a acciones (precondiciones), en el que los costes de las

Extracción	Selección de la "mejor" acción que genera cada (sub)meta
METRIC-FF	acción en la menor capa
h_{level}	acción en la menor capa
SAPA	acción con menor coste asociado

Tabla 3.2: Diferencias algorítmicas en la extracción del plan relajado.

precondiciones se agregan de forma bien aditiva o bien mediante un máximo. Además en el caso de SAPA el proceso finaliza en el punto fijo. Por otro lado, h_{level} retrasa la inclusión de acciones en el *RPG* según su coste estimado, mientras que SIMPLANNER retrasa la inclusión de proposiciones según su coste estimado. En este último caso la propagación de costes de las precondiciones a las acciones se realiza de forma aditiva, mientras que en h_{level} se toma el máximo, que viene dado por el coste de la primera capa en que la acción es aplicable.

Respecto a la extracción, mientras que en METRIC-FF y h_{level} se selecciona la acción situada en una capa menor, en SAPA se selecciona aquélla que tiene asociado un menor coste acumulado.

Dado que el algoritmo de SIMPLANNER se puede transformar fácilmente en el de h_{level}, a partir de aquí nos referiremos a las heurísticas que se obtienen como $h_{level\text{-}max}$ (original) y $h_{level\text{-}add}$ (SIMPLANNER) respectivamente.

3.4. Comparación experimental preliminar

En esta sección se exponen algunos resultados experimentales orientados a comparar la heurística $h_{level\text{-}max}$ con otras heurísticas diseñadas para planificación basada en costes.

En relación con la comparación de heurísticas para planificación basada en costes, cabe mencionar varios aspectos a tener en cuenta:

1. Excepto la heurística *max* y las de la familia h^m, todas las demás heurísticas para planificación basada en costes no son admisibles. Cuando dos heurísticas son admisibles está claramente establecido cómo compararlas: una heurística está más informada que otra si sus valores son siempre mayores, siempre dentro de la admisibilidad. El hecho de que una heurística esté más informada que otra asegura que los algoritmos de búsqueda admisibles (aquellos que aseguran encontrar la solución óptima) encontrarán la solución en menor tiempo (es decir, explorando menos estados). En el caso de heurísticas no admisibles, esta propiedad no se cumple y no hay algoritmos que garanticen encontrar la solución óptima, a no ser que se agote la búsqueda; es decir, que se exploren todos los estados.
2. Si se comparan las heurísticas numéricas a partir de los resultados que se obtienen utilizando un algoritmo de búsqueda concreto, el algoritmo de búsqueda afecta a los resultados.
3. En el caso de planificación basada en costes, la distribución de costes es diferente para cada problema de cada dominio.

4. La solución óptima de los problemas de planificación basada en costes de una complejidad media o elevada no se conoce, porque ningún planificador existente es capaz de encontrarla.

El primer punto, es decir, el hecho de que las heurísticas sean no admisibles, impide que éstas se puedan comparar desde un punto de vista teórico dado que no existen propiedades teóricas que lleven a concluir cuál es la mejor heurística. Esto supone que la única vía posible para realizar comparativas sea la experimental. Desde el punto de vista experimental, uno se podría plantear comparar los resultados obtenidos utilizando uno o varios algoritmos de búsqueda, pero esto, a su vez, plantea algunos problemas a la hora de analizar los resultados. Primero porque como se menciona en el punto 2 el resultado obtenido depende del algoritmo utilizado (además de depender de la heurística). Segundo porque determinar qué se considera una *buena* heurística también depende del algoritmo con que ésta se utilice: por ejemplo con una búsqueda *hill-climbing* una buena heurística debe ser capaz de establecer un *ranking* correcto entre estados hermanos, mientras que con una búsqueda *mejor-primero* una buena heurística debe ser capaz de establecer un *ranking* correcto entre todos los estados. Además, como se menciona en el punto 3 la distribución de costes depende de cada problema de cada dominio, y la heurística depende de la distribución de costes. Todo esto significa que las conclusiones que se podrían establecer comparando los resultados de combinar las heurísticas con un algoritmo de búsqueda concreto, serían del tipo: *la heurística h_1 es mejor que la heurística h_2 para el problema p del dominio d, utilizando el algoritmo de búsqueda x.*

Dado que es deseable obtener conclusiones más generales que la del párrafo anterior, se puede plantear dejar de lado el algoritmo de búsqueda y comparar exclusivamente los valores de la heurística. Aunque las heurísticas son no-admisibles parece lógico que sea preferible aquella heurística que proporciona valores lo más cercanos posible al valor real óptimo. La cercanía al valor real óptimo de una heurística se puede medir, por ejemplo, calculando el error medio que comete la heurística en todos los estados del espacio de búsqueda (o en una muestra significativa de ellos). Sin embargo, desde el punto un punto de vista práctico esto es imposible de llevar a cabo, puesto que, como se afirma en el punto 4, en general no se conoce la solución óptima de los problemas (y mucho menos la solución óptima para cada estado posible o muestreado), con lo cual no se puede medir tal error.

En este apartado de experimentos se ha optado por el siguiente método para comparar la precisión de las heurísticas:

- Se ha diseñado un algoritmo de búsqueda que trata de encontrar la primera solución de cada problema con una búsqueda *Enforced Hill-Climbing* modificada para planificación basada en costes. Una vez encontrada la primera solución, el coste de la misma se utiliza como límite de coste (límite de $g(n)$) en una búsqueda mejor primero con $f(n) = W \times h(n) + g(n)$, que no finaliza en la primera solución encontrada sino que va proporcionando soluciones de mejor calidad cada vez hasta agotar el límite de tiempo considerado.
- El algoritmo del punto anterior se ha ejecutado para cada heurística considerando un límite de tiempo de 300 sg.

- Para cada problema de cada dominio, se almacena la mejor solución encontrada de las soluciones encontradas con todas las heurísticas.
- Para cada estado en el camino de la mejor solución encontrada, se contrasta el valor de la heurística en ese estado y el coste real (en general subóptimo) que supone ir desde el estado considerado al último estado de la mejor solución encontrada.

La Figura 3.4 muestra cómo se calcula para cada estado el coste de la mejor solución encontrada a partir de él. Partiendo de la mejor solución encontrada para cada problema (camino que se muestra en la figura), se contrasta la heurística del nodo: $h(n)$ con el coste de la mejor solución encontrada a partir de ese nodo: $mejor\_coste(n) = g(\mathcal{G}) - g(n)$.

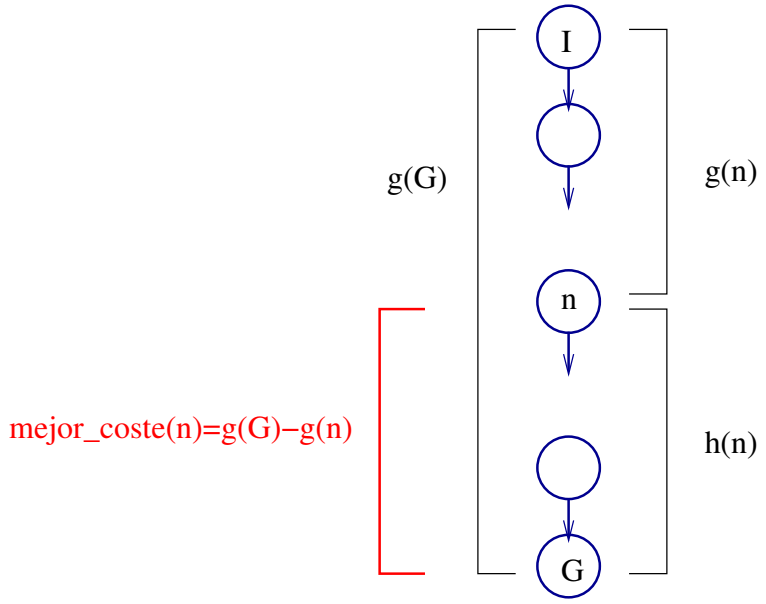

Figura 3.12: Cálculo de la relación entre h(n) y la mejor h encontrada para ese nodo.

En los experimentos se han comparado las heurísticas de h_{max}, h_{add}, la heurística de METRIC-FF, y las heurísticas $h_{level\text{-}max}$ y $h_{level\text{-}add}$. Las Figuras 3.13 y 3.14 muestran los resultados obtenidos aplicando el método descrito en cinco dominios de las Competición Internacional de Planificación del año 2003, los dominios: *Zenotravel*, *Satellite*, *Driverlog*, *Rovers* y *Depots*. En las gráficas, para cada estado, se muestra en el eje x el mejor coste encontrado ($mejor\_coste(n)$) y en el eje y la estimación heurística ($h(n)$). La diagonal indica cuál debería ser el valor de la heurística. Cuando se conoce la solución óptima, valores por encima de la diagonal son sobre-estimaciones y valores por debajo de la diagonal son sub-estimaciones del valor óptimo real. En cada dominio, la gráfica de la derecha muestra sólo los estados para los que se ha conseguido una solución óptima. En el caso de las gráficas de la izquierda, la solución es subóptima y la diagonal es una estimación, puesto que se ha dibujado utilizando un coste subóptimo (la diagonal real está por debajo de la dibujada).

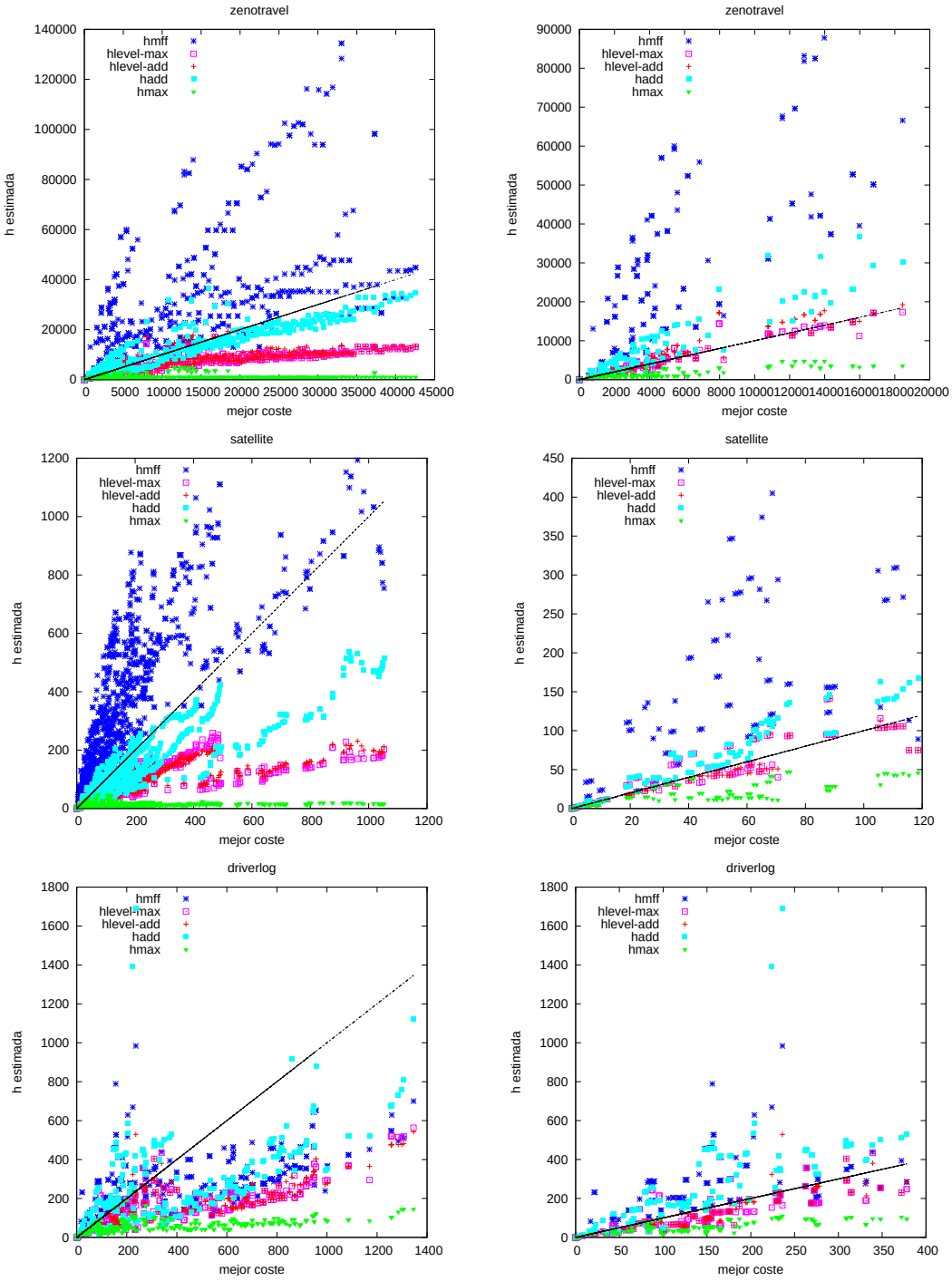

Figura 3.13: Precisión aproximada de las heurísticas (I).

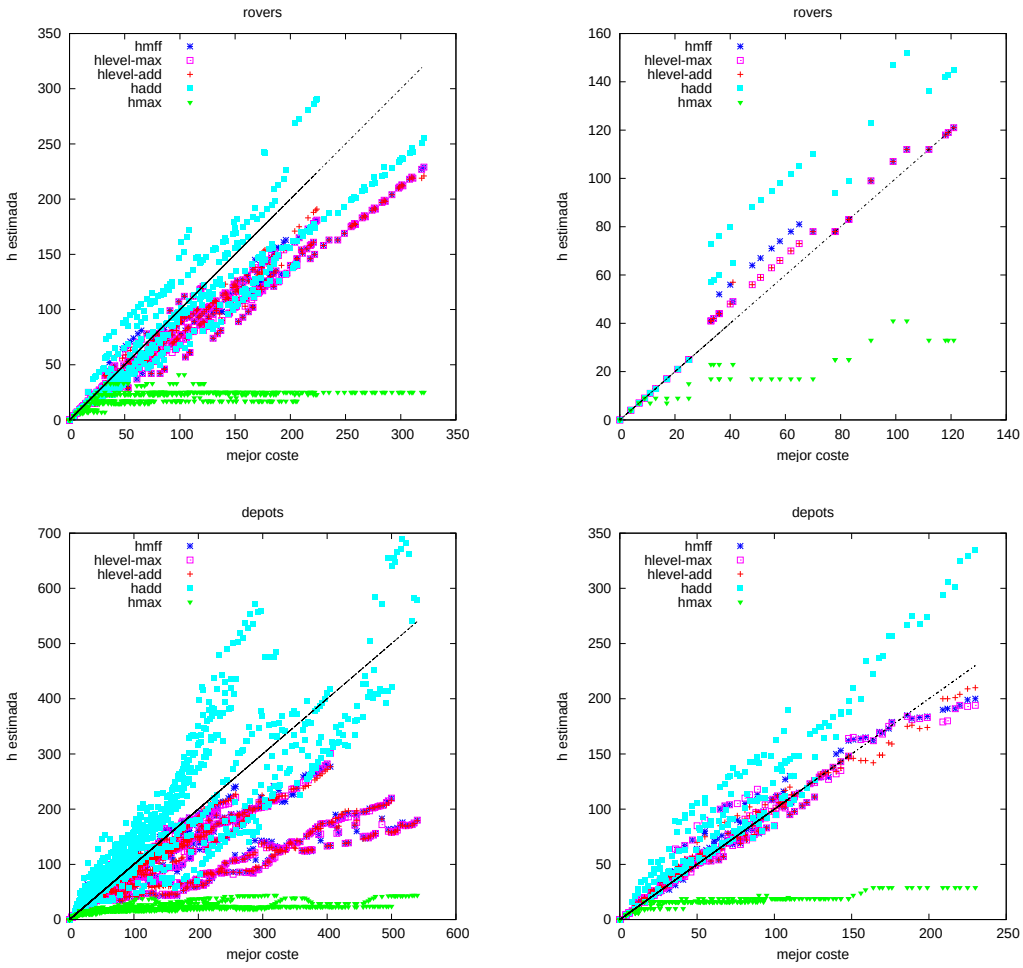

Figura 3.14: Precisión aproximada de las heurísticas (II).

A la vista de las gráficas se pueden observar características de las heurísticas que son bien conocidas como que:

- La heurística h_{max} (la única admisible en estos experimentos), subestima considerablemente en los 5 dominios, produciendo estimaciones bastante alejadas de la diagonal, lo que implica que su precisión es baja.
- La heurística h_{add} proporciona estimaciones siempre más altas que h_{max} (lo cual es normal, ya que teóricamente se puede demostrar fácilmente que $h_{add}(\mathcal{G}, s) > h_{max}(\mathcal{G}, s)$).
- Observando las gráficas obtenidas a partir de soluciones óptimas, se puede decir que en la mayoría de los casos la heurística h_{add} sobre-estima, al menos en problemas relativamente sencillos. Desde el punto de vista teórico esto no tiene por qué ser así ya que al calcular la heurística se ignoran los borrados. Sin embargo, parece que

el hecho de considerar independencia entre las (sub)metas y hacer una agregación de los costes de las precondiciones aditiva, incrementa más el valor de la heurística que si se incluyeran los borrados.

Por otro lado:

- La heurística de METRIC-FF que no tiene en cuenta los costes de las acciones produce estimaciones muy alejadas de la diagonal (por encima de la misma, y bastante por encima de la heurística h_{add}) en los dominios *Zenotravel* y *Satellite*, por lo tanto en estos dominios tiene una precisión muy baja. Estos dominios se caracterizan porque las acciones de desplazamiento (*fly* y *zoom* en *Zenotravel*, y *turn-to* en *Satellite*) no tienen más restricciones que el hecho de que el avión/satélite se encuentren en el lugar de origen. Así, se puede conseguir que el avión/satélite esté en un determinado lugar de muchas formas distintas, donde la forma de conseguirlo con menos operadores puede estar muy lejos de la forma de conseguirlo con menos coste.
- La heurística de METRIC-FF tiene un comportamiento más adecuado en dominios dónde el número de formas distintas de conseguir hechos es más limitado, como en *Driverlog*, *Rovers* y *Depots*. Esto es razonable, ya que en el extremo, si sólo hay una forma de conseguir un determinado tipo de hechos, poco importa considerar el coste. En *Driverlog*, los valores de h_{mff} se aproximan más a los de la heurística aditiva, mientras que en los otros dos dominios sus valores se aproximan más a los de $h_{level\text{-}max}$ y $h_{level\text{-}add}$.
- En los cinco dominios los valores de $h_{level\text{-}max}$ y $h_{level\text{-}add}$ son muy parecidos.

3.5. Resumen

En este capítulo se ha presentado la heurística h_{level}, haciendo una descripción procedural de la misma y utilizando para ello una descripción generalizada de los algoritmos para expandir el grafo de planificación y para extraer una solución al problema relajado del mismo. Asimismo, se han establecido las diferencias, siempre desde un punto de vista procedural, entre el algoritmo de h_{level} y otros algoritmos para calcular heurísticas a partir de grafos relajados.

Al final de este capítulo se han reportado una serie de experimentos con el objetivo de comparar la relación entre las estimaciones heurísticas en cinco dominios de planificación basada en costes. En estos dominios las heurísticas basadas en una construcción del grafo de planificación relajado en niveles incrementales de coste son aparentemente muy similares. Las heurísticas más adecuadas parecen ser éstas junto con la heurística aditiva. Tanto la heurística del máximo (en todos los dominios) como la de METRIC-FF (en algunos dominios) son poco informadas. En cualquier caso, mientras que las heurísticas del máximo y aditiva se han venido describiendo en la literatura desde un punto de vista declarativo (indicando *qué* se calcula), las basadas en grafos de planificación relajados se han descrito desde un punto de vista procedural (indicado *cómo* se calcula). Esto motiva los dos siguientes capítulos en los que se hace un esfuerzo por establecer la relación desde un punto de vista declarativo vía un análisis matemático y una definición unificada.

Capítulo 4

Análisis matemático de las heurísticas basadas en *RPGs*

En este capítulo se realiza un análisis matemático de las heurísticas basadas en grafos de planificación relajados. Este análisis, que tiene un carácter esencialmente inductivo, tiene por objetivo determinar desde un punto de vista teórico las similitudes y diferencias entre ellas. Para conseguir este objetivo, en este capítulo se caracterizan las acciones cuyos costes se tienen en cuenta en el cálculo de cada una de las heurísticas, mediante una definición declarativa y matemática.

En planificación basada en costes, el coste de cualquier plan solución π (incluyendo el plan óptimo), a partir de cualquier estado, se calcula como:

$$coste(\pi) = \sum_{a \in \mathcal{A}} N(a) \times coste(a)$$

donde $N(a) \in \mathbb{N} \cup \{0\}$ indica el número de veces que se suma el coste de la acción a en el cálculo del coste total del plan. El valor $N(a)$ de cada acción es desconocido a priori.

Las heurísticas basadas en Grafos de Planificación Relajados (*RPGs*) se basan en hacer una estimación de $N(a)$ para cada acción $a \in \mathcal{A}$, $N'(a)$, de manera que $N'(a)$ únicamente puede tomar los valores 0 ó 1. En el problema relajado no hace nunca falta aplicar una acción más de una vez, ya que ninguna otra acción borra sus efectos. Sin embargo, existen otras heurísticas, también basadas en ignorar los efectos negativos de las acciones en las que $N'(a)$ puede ser mayor que 1, como es el caso de h_{add}.

En este apartado se hace un estudio orientado a determinar cuáles son las acciones cuyo coste se multiplica por un factor $N'(a) = 1$ en el cálculo de las heurísticas basadas en *RPGs*. Las relaciones y diferencias entre estas heurísticas vienen determinadas por los factores $N'(a)$ de las acciones. Hasta este trabajo, las heurísticas basadas en *RPGs* se han venido definiendo desde un punto de vista procedural, expresando *cómo* calcular la heurística. Sin embargo una descripción declarativa que muestre concisamente *qué* es lo que

se calcula puede ser de gran utilidad para establecer las relaciones reales. Este argumento sigue la línea de H. Geffner (Geffner, 2007), que afirma que las declaraciones matématicas son definiciones declarativas claras y concisas.

El punto de partida del análisis en este capítulo son las definiciones matemáticas de las heurísticas h_{max} y h_{add}. A partir de éstas se establecen las siguientes definiciones previas, que serán de utilidad en el análisis posterior de las heurísticas.

4.1. Definiciones previas

Las definiciones de h_{max} y h_{add} permiten estimar el coste de conseguir una proposición p a partir de un estado s. Esta estimación se obtiene considerando al menos una acción a que consigue la proposición p (es decir, $p \in add(a)$). Para esta acción, el factor $N'(a)$ que multiplica su coste es mayor o igual que 1. La siguiente definición se obtiene fijando la acción con la que se debe conseguir p.

Definición 4.1 (Coste de una proposición p que se genera mediante la acción a, según h_{max} y h_{add}). El coste de conseguir una proposición p mediante la acción a ($p \in add(a)$) a partir de un estado s, se define según h_{max} y h_{add} respectivamente como:

$$h^a_{max}(p,s) = coste(a) + \max_{q \in pre(a)} \{h_{max}(q,s)\}$$

$$h^a_{add}(p,s) = coste(a) + \sum_{q \in pre(a)} h_{add}(q,s)$$

Utilizando esta definición se consiguen las definiciones generales de $h_{max}(p,s)$ y $h_{add}(p,s)$, en las que la acción que se elige para generar p es, de entre todas aquéllas que generan p (denotadas como $A(p)$), precisamente la que minimiza la estimación heurística:

$$h_{max}(p,s) = h^a_{max}(p,s), \text{ donde } a \in \arg\min_{a' \in A(p)} \left\{ coste(a') + \max_{q \in pre(a')} \{h_{max}(q,s)\} \right\} \quad (4.1)$$

y

$$h_{add}(p,s) = h^a_{add}(p,s), \text{ donde } a \in \arg\min_{a' \in A(p)} \left\{ coste(a') + \sum_{q \in pre(a')} h_{add}(q,s) \right\} \quad (4.2)$$

Definición 4.2 ($\pi_{add}(p,s)$: conjunto de acciones cuyo factor $N'(a)$ es mayor o igual que la unidad en el cálculo de $h_{add}(p,s)$ de una proposición). El conjunto de acciones cuyo factor $N'(a)$ es mayor o igual que la unidad en el cálculo de $h_{add}(p,s)$ para una proposición p y un estado s, se define como:

$$\pi_{add}(p,s) = \begin{cases} \emptyset & \text{si } p \in s \\ \{a_p\} \bigcup \{ \bigcup_{q \in pre(a_p)} \pi_{add}(q,s) \} & \text{en otro caso} \end{cases}$$

donde $a_p \in \arg\min_{a \in A(p)} h^a_{add}(p, s)$.

Definición 4.3 ($\pi_{add}(P, s)$**: conjunto de acciones cuyo factor** $N'(a)$ **es mayor o igual que la unidad en el cálculo de** $h_{add}(P, s)$ **de un conjunto de proposiciones).** El conjunto de acciones cuyo factor $N'(a)$ es mayor o igual que la unidad en el cálculo de $h_{add}(P, s)$ para un conjunto de proposiciones P y un estado s, se define como:

$$\pi_{add}(P, s) = \bigcup_{p \in P} \pi_{add}(p, s)$$

En el caso de h_{max}, los factores $N'(a)$ que multiplican el coste de cada acción toman únicamente los valores 0 ó 1. Así:

Definición 4.4 ($\pi_{max}(p, s)$**: conjunto de acciones cuyo factor** $N'(a)$ **es igual que la unidad en el cálculo de** $h_{max}(p, s)$ **de una proposición).** El conjunto de acciones cuyo factor $N'(a)$ es igual a la unidad en el cálculo de $h_{max}(p, s)$ para una proposición p y un estado s, se define como:

$$\pi_{max}(p, s) = \begin{cases} \emptyset & \text{si } p \in s \\ \{a_p\} \bigcup \pi_{max}(q, s) & \text{en otro caso} \end{cases}$$

donde $a_p \in \arg\min_{a \in A(p)} h^a_{max}(p, s)$ y $q \in \arg\max_{q' \in pre(a_p)} h_{max}(q', s)$

Definición 4.5 ($\pi_{max}(P, s)$**: conjunto de acciones cuyo factor** $N'(a)$ **es igual que la unidad en el cálculo de** $h_{max}(P, s)$ **de un conjunto de proposiciones).** El conjunto de acciones cuyo factor $N'(a)$ es igual a la unidad en el cálculo de $h_{max}(P, s)$ para un conjunto de proposiciones P y un estado s, se define como:

$$\pi_{max}(P, s) = \pi_{max}(p, s)$$

donde $p \in \arg\max_{p' \in P} h_{max}(p', s)$

De estas definiciones se puede deducir lo siguiente: todas las acciones en $\pi_{add}(P, s)$ y $\pi_{max}(P, s)$ minimizan $h^a_{add}(p, s)$ y $h^a_{max}(p, s)$ respectivamente, donde p es la (sub)meta que generan y por la que fueron seleccionadas. $\pi_{add}(P, s)$ contiene una acción a_p por cada proposición p del conjunto P. Además, contiene una acción a_q por cada submeta q que se obtiene haciendo regresión de metas a través de a_p en un proceso recursivo. Sin embargo, en el caso de $\pi_{max}(P, s)$ esto no es así. $\pi_{max}(P, s)$ contiene sólo una acción a_p que genera una proposición de P, la proposición p que maximiza $h_{max}(p, s)$. Además, $\pi_{max}(P, s)$ contiene sólo una acción para generar una de las submetas que se generan haciendo regresión de metas a través de a_p. Concretamente, la submeta q que maximiza $h_{max}(q, s)$.

El párrafo anterior explica el motivo por el que la heurística h_{max} produce subestimaciones tan elevadas: se calcula tomando solamente algunas de las acciones para generar las metas y (sub)metas del problema relajado. Es decir, las acciones cuyo coste se tiene en

cuenta para calcular h_{max} no constituyen, en general un plan relajado, sino una parte de él. Sin embargo el conjunto $\pi_{max}(P,s)$ se puede ampliar de manera que, guiándose siempre por h_{max}, contenga acciones *para generar cada proposición* del conjunto P y *para generar cada proposición* obtenida a través de la regresión de metas (es decir, de manera que las acciones que se tienen en cuenta sí constituyan un plan relajado); respetando también siempre que las acciones que se elijan sean las que minimizan el valor h^a_{max} de la proposición correspondiente.

La forma de ampliar el conjunto π_{max} se explica a continuación. Este conjunto contiene las acciones con factor $N'(a) = 1$ en el cálculo de la heurística h_{max}. Al ampliarlo, $N'(a)$ será igual a uno para más acciones y obtendremos una estimación heurística distinta y mayor o igual a h_{max}. Denominaremos la heurística resultante *heurística del máximo ampliada* y la denotaremos como h^+_{max}. La heurística h^+_{max} se calcula como la suma de los costes de las acciones en el conjunto π_{max} ampliado, que denotaremos como π^+_{max}. Este conjunto se define como sigue:

Definición 4.6 ($\pi^+_{max}(p,s)$**: conjunto** $\pi_{max}(p,s)$ **ampliado).** El conjunto de acciones cuyo factor $N'(a)$ es mayor o igual que la unidad en el cálculo de $h^+_{max}(p,s)$ para una proposición p y un estado s, se define como:

$$\pi^+_{max}(p,s) = \begin{cases} \emptyset & \text{si } p \in s \\ \{a_p\} \bigcup \{ \bigcup\limits_{q \in pre(a_p)} \pi^+_{max}(q,s)\} & \text{en otro caso} \end{cases}$$

donde $a_p \in \arg\min\limits_{a \in A(p)} h^a_{max}(p,s)$.

Es decir, el conjunto ampliado se obtiene haciendo una unión de los conjuntos $\pi^+_{max}(q,s)$ de cada precondición de la acción a_P seleccionada, donde antes se elegía exclusivamente el conjunto $\pi_{max}(q,s)$ de la precondición con mayor $h_{max}(q,s)$. Nótese, que las acciones a_p se siguen eligiendo con el mismo criterio, por lo tanto se asegura que $\pi_{max}(p,s) \subseteq \pi^+_{max}(p,s)$.

El conjunto π_{max} ampliado para un conjunto de proposiciones se obtiene aplicando la misma idea:

Definición 4.7 ($\pi^+_{max}(P,s)$**: conjunto** $\pi_{max}(P,s)$ **ampliado).** El conjunto de acciones cuyo factor $N'(a)$ es mayor o igual que la unidad en el cálculo de $h^+_{max}(P,s)$ para un conjunto de proposiciones P y un estado s, se define como:

$$\pi^+_{max}(P,s) = \bigcup_{p \in P} \pi^+_{max}(p,s)$$

Así, la heurística del máximo ampliada se calcula sumando los costes de las acciones del conjunto $\pi^+_{max}(\mathcal{G},s)$:

Definición 4.8 (**Heurística del máximo ampliada).** La heurística del máximo ampliada para conseguir una proposición p y para conseguir un conjunto de proposiciones P, a partir de un estado s, se define respectivamente como:

- El coste estimado de conseguir una proposición p a partir de un estado s es:

$$h^+_{max}(p,s) = \sum_{a \in \pi^+_{max}(p,s)} coste(a)$$

- El coste estimado de conseguir un conjunto de proposiciones P a partir de un estado s es:

$$h^+_{max}(P,s) = \sum_{a \in \pi^+_{max}(P,s)} coste(a)$$

O lo que es lo mismo, la heurística de máximo ampliada se calcula como

$$h^+_{max}(P,s) = \sum_{a \in \mathcal{A}} N'(a) \times coste(a) \tag{4.3}$$

donde

$$N'(a) = \begin{cases} 1 & \text{si } a \in \pi^+_{max}(P,s) \\ 0 & \text{en otro caso} \end{cases}$$

Como ya se ha mencionado, se garantiza la siguiente propiedad:

Propiedad 4.1. $\pi_{max}(P,s)$ es un subconjunto de $\pi^+_{max}(P,s)$: $\pi_{max}(P,s) \subseteq \pi^+_{max}(P,s)$.

Demostración. Dado que:

- $\forall p_i \in P$, se cumple que $\pi_{max}(p_i,s) \subseteq \pi^+_{max}(p_i,s)$,
- $\pi_{max}(P,s)$ es uno de los conjuntos $\pi_{max}(p_i,s)$ (concretamente el de la proposición p_j para la que se maximiza h_{max}), y
- $\pi^+_{max}(P,s)$ se calcula como la unión de todos los conjuntos $\pi^+_{max}(p,s)$,

tenemos que $\pi_{max}(P,s) = \pi_{max}(p_j,s) \subseteq \bigcup_i \pi_{max}(p_i,s) = \pi^+_{max}(P,s)$

Por lo tanto, se puede asegurar que $\pi_{max}(P,s) \subseteq \pi^+_{max}(P,s)$. □

La propiedad anterior asegura que la heurística del máximo ampliada proporciona siempre valores mayores o iguales que la heurística del máximo.

Por otro lado, se puede forzar a que la heurística aditiva se calcule con factores $N'(a)$ con valores entre 0 y 1, en lugar de permitir que estos factores sean mayor que 1. Esto se consigue sencillamente asignando un valor de 1 a todos los factores de las acciones que están contenidas en $\pi_{add}(\mathcal{G},s)$ y un factor de 0 a las acciones del dominio que no están contenidas en este conjunto. De esta forma, el valor de la heurística se vería decrementado. Denominaremos esta heurística *heurística aditiva restringida*, y la denotaremos como h^-_{add}.

Definición 4.9 (Heurística aditiva restringida). La heurística aditiva restringida para conseguir el coste de una proposición p y el coste de un conjunto de proposiciones P, a partir de un estado s, se define respectivamente como:

- Para una proposición p:

$$h^{-}_{add}(p,s) = \sum_{a \in \pi_{add}(p,s)} coste(a)$$

- Para un conjunto de proposiciones P:

$$h^{-}_{add}(P,s) = \sum_{a \in \pi_{add}(P,s)} coste(a)$$

La heurística aditiva restringida se expresa en función de los factores $N'(a)$ de la siguiente forma:

$$h^{-}_{add}(P,s) = \sum_{a \in \mathcal{A}} N'(a) \times coste(a) \tag{4.4}$$

donde

$$N'(a) = \begin{cases} 1 & \text{si } a \in \pi_{add}(P,s) \\ 0 & \text{en otro caso} \end{cases}$$

La heurística aditiva restringida, por su definición, siempre proporciona valores menores o iguales que la heurística aditiva; los factores $N'(a)$ son distintos de cero para las mismas acciones, con la única diferencia de que en la restringida son como mucho uno, mientras que en la aditiva pueden ser mayores que uno. La heurística aditiva restringida es exactamente h_a (Keyder and Geffner, 2008) (explicada en el estado de la cuestión, sección 2.4.4.5, página 53).

Se puede observar que la única diferencia entre la heurística del máximo ampliada y la heurística aditiva restringida es el criterio que se utiliza para seleccionar las acciones que tienen un factor $N'(a)$ igual a uno. Mientras que en la heurística del máximo ampliada el criterio es seleccionar aquéllas que minimizan el valor de h_{max} de la proposición correspondiente, en la aditiva restringida son aquéllas que minimizan el valor de h_{add}. Otra posibilidad razonable para seleccionar estas acciones, que da lugar a una heurística distinta, es seleccionar las acciones que minimizan el valor de la heurística que se está calculando, es decir las que minimizan la suma de los costes de las acciones del conjunto $\pi(p,s)$. Esto se consigue sustituyendo en la definición (4.6) la expresión $a_p \in \arg\min_{a \in A(p)} h^{a}_{max}(p,s)$ por $a_p \in \arg\min_{a \in A(p)} h^{+a}_{max}(p,s)$, o en la definición (4.2), la expresión $a_p \in \arg\min_{a \in A(p)} h^{a}_{add}(p,s)$ por $a_p \in \arg\min_{a \in A(p)} h^{-a}_{add}(p,s)$. La heurística resultante se ha denominado recientemente *heurística set-additive* (Keyder and Geffner, 2007) (descrita en el estado de la cuestión, sección 2.4.4.5, página 53).

4.2. Ejemplo ilustrativo

El ejemplo a continuación ilustra las definiciones previas. El dominio y problema que utilizamos se muestran en la Figura 4.1, que es una reproducción del ejemplo utilizado en la sección anterior.

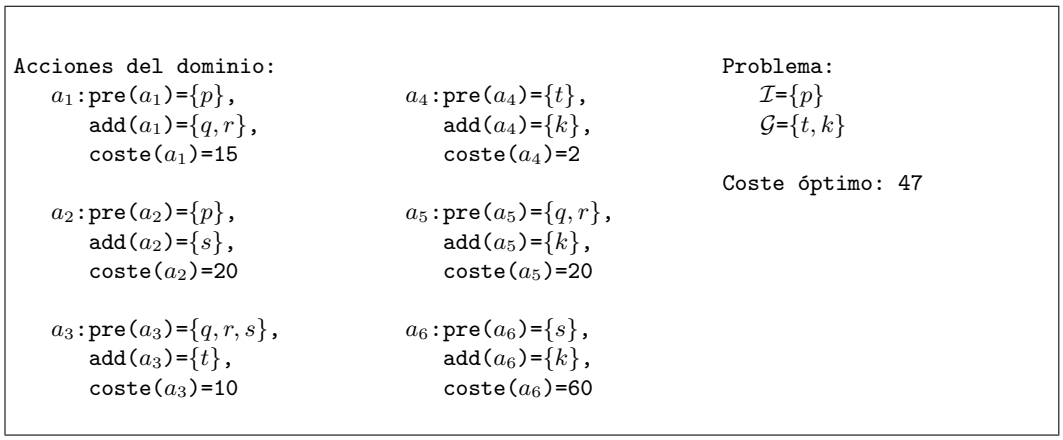

Figura 4.1: Ejemplo artificial de un problema en un dominio relajado.

El valor de h_{add} para cada proposición y el conjunto de acciones con factor $N'(a)$ mayor o igual que la unidad en la estimación de cada proposición se calculan como:

$h_{add}(q, \mathcal{I}) = coste(a_1) = 15$
$\pi_{add}(q, \mathcal{I}) = \{a_1\}$

$h_{add}(r, \mathcal{I}) = coste(a_1) = 15$
$\pi_{add}(r, \mathcal{I}) = \{a_1\}$

$h_{add}(s, \mathcal{I}) = coste(a_2) = 20$
$\pi_{add}(s, \mathcal{I}) = \{a_2\}$

$h_{add}(t, \mathcal{I}) = coste(a_3) + h_{\max}(q, \mathcal{I}) + h_{add}(r, \mathcal{I}) + h_{\max}(s, \mathcal{I}) = 10 + 15 + 15 + 20 = 60$
$\pi_{add}(t, \mathcal{I}) = \{a_1, a_2, a_3\}$

$h_{add}(k, \mathcal{I}) = \min\{coste(a_4) + h_{add}(t, \mathcal{I}), coste(a_5) + h_{add}(q, \mathcal{I}) + h_{add}(r, \mathcal{I}), coste(a_6) + h_{add}(s, \mathcal{I})\} =$
$= \min\{2 + 60, 20 + 15 + 15, 60 + 20\}$
$= \min\{62, 50, 80\} = 50$
$\pi_{add}(k, \mathcal{I}) = \{a_1, a_5\}$

El valor de h_{add} para las metas del problema se calcula como:

$h_{add}(\mathcal{G}, \mathcal{I}) = h_{add}(k, I) + h_{add}(t, I) = 50 + 60 = 110$

Así, $h_{add}(\mathcal{G}, \mathcal{I})$ se calcula como: $4 \times coste(a_1) + coste(a_2) + coste(a_3) + coste(a_5)$

Por lo tanto, el conjunto de acciones con factor $N'(a)$ mayor o igual que uno es:

$\pi_{add}(\mathcal{G}, \mathcal{I}) = \{a_1, a_2, a_3, a_5\}$

La *heurística aditiva restringida* se calcula sumando los costes de las acciones en el

conjunto anterior, por lo tanto:

$$h^{-}_{add}(\mathcal{G},\mathcal{I}) = coste(a_1) + coste(a_2) + coste(a_3) + coste(a_5) = 65$$

Respecto a h_{max}, el valor de h_{max} para cada proposición, el conjunto de acciones con factor $N'(a)$ igual a uno en la estimación de cada proposición y este mismo conjunto ampliado son:

$$h_{max}(q,\mathcal{I}) = coste(a_1) = 15$$
$$\pi_{max}(q,\mathcal{I}) = \{a_1\}$$
$$\pi^{+}_{max}(q,\mathcal{I}) = \{a_1\}$$

$$h_{max}(r,\mathcal{I}) = coste(a_1) = 15$$
$$\pi_{max}(r,\mathcal{I}) = \{a_1\}$$
$$\pi^{+}_{max}(r,\mathcal{I}) = \{a_1\}$$

$$h_{max}(s,\mathcal{I}) = coste(a_2) = 20$$
$$\pi_{max}(s,\mathcal{I}) = \{a_2\}$$
$$\pi^{+}_{max}(s,\mathcal{I}) = \{a_2\}$$

$$h_{max}(t,\mathcal{I}) = coste(a_3) + \text{máx}\{h_{\max}(q,\mathcal{I}), h_{\max}(r,\mathcal{I}), h_{\max}(s,\mathcal{I})\} = 10 + 20 = 30$$
$$\pi_{max}(t,\mathcal{I}) = \{a_2, a_3\}$$
$$\pi^{+}_{max}(t,\mathcal{I}) = \{a_1, a_2, a_3\}$$

$$\begin{aligned} h_{max}(k,\mathcal{I}) = \text{mín}\{&coste(a_4) + h_{\max}(t,\mathcal{I}),\\ &coste(a_5) + \text{máx}\{h_{max}(q,\mathcal{I}), h_{\max}(r,\mathcal{I})\},\\ &coste(a_6) + h_{max}(s,\mathcal{I})\} =\\ = \text{mín}\{&2 + h_{\max}(t,\mathcal{I}), 20 + \text{máx}\{h_{max}(q,\mathcal{I}), h_{\max}(r,\mathcal{I})\}, 60 + h_{max}(s,\mathcal{I})\} =\\ = \text{mín}\{&2 + 30, 20 + 15, 60 + 20\} = \text{mín}\{32, 35, 80\} = 32 \end{aligned}$$

$$\pi_{max}(k,\mathcal{I}) = \{a_2, a_3, a_4\}$$
$$\pi^{+}_{max}(k,\mathcal{I}) = \{a_1, a_2, a_3, a_4\}$$

El valor de h_{max} para las metas del problema se calcula como:

$$h_{max}(\mathcal{G},\mathcal{I}) = \text{máx}\{h_{\max}(k,I), h_{max}(t,I)\} = \text{máx}\{32, 30\} = 32$$

Así, $h_{max}(\mathcal{G},\mathcal{I})$ se calcula como: $coste(a_2) + coste(a_3) + coste(a_4)$

En este caso, el conjunto de acciones con factor $N'(a)$ igual a uno es:

$$\pi_{max}(\mathcal{G},\mathcal{I}) = \{a_2, a_3, a_4\}$$

Sin embargo, el conjunto ampliado de acciones con factor $N'(a)$ igual a uno es:

$$\pi^{+}_{max}(\mathcal{G},\mathcal{I}) = \{a_1, a_2, a_3, a_4\}$$

y la *heurística del máximo ampliada* tiene el valor:

$$h^{+}_{max}(\mathcal{G},\mathcal{I}) = coste(a_1) + coste(a_2) + coste(a_3) + coste(a_4) = 47$$

La diferencia entre la heurística *set-additive* y las heurísticas del máximo ampliada y aditiva restringida radica en la forma de seleccionar la acción para conseguir k. El método en la heurística *set-additive* es seleccionar la acción que da lugar a un conjunto $\pi(k,\mathcal{I})$ con menor coste. Este conjunto podría ser:

- $\{a_1, a_2, a_3, a_4\}$ con suma de costes igual a 47, o
- $\{a_1, a_5\}$ con suma de costes igual a 35, o
- $\{a_2, a_6\}$ con suma de costes igual a 80

De los que el de menor coste es el segundo. Por lo tanto, la acción que se selecciona para conseguir k es a_5. Así, $\pi_{set\text{-}additive}(\mathcal{G}, \mathcal{I}) = \{a_1, a_2, a_3, a_5\}$ que da lugar a un valor heurístico de 65. Aunque en este ejemplo las estimaciones de la heurística aditiva restringida y de la heurística *set-additive* coinciden, no tiene por qué ser así en general.

4.3. Heurística de METRIC-FF

En este apartado se establecen las definiciones y proposiciones formales necesarias para describir la heurística de METRIC-FF desde un punto de vista declarativo. El objetivo es caracterizar formalmente las acciones que pertenecen al plan relajado que calcula METRIC-FF es decir las acciones cuyo factor $N'(a)$ es igual a la unidad en el cálculo de la heurística.

El análisis que se presenta no tiene en cuenta el tratamiento de los *efectos secundarios* que se realiza en el proceso de extracción de la heurística a partir del *RPG*. Tampoco se tiene el cuenta el tratamiento de precondiciones, efectos y metas numéricas, implementado en METRIC-FF, que no cabe en el modelo de planificación con costes definido para esta tesis. Nos referiremos a esta heurística, con las simplificaciones mencionadas, como $h_{basic\text{-}mff\text{-}NSE}$.

Como se demostrará, está heurística está estrechamente relacionada con la formulación de la *heurística del máximo ampliada*, pero definida para planificación clásica (es decir, asumiendo costes unitarios). De aquí en adelante, para evitar confusiones, nos referiremos a las definiciones que consideran costes unitarios incluyendo la palabra *unit* en el nombre.

Definición 4.10. (Nivel de una proposicion en h_{mff}) El *nivel* de una proposición p en el *RPG* es el índice de la primera capa de proposiciones del *RPG*, P_i, que contiene la proposición.

$$nivel(p) = \min_{i \in [0 \ldots final]} \{i \mid p \in P_i\}$$

donde $final$ representa el índice de la última capa del *RPG*.

Definición 4.11. (Nivel de una acción en h_{mff}) El *nivel* de una acción a en el *RPG* es el índice de la capa de acciones del *RPG*, A_i, que contiene la acción:

$$nivel(a) = i, a \in A_i$$

El nivel de una acción también se puede definir como el índice de la primera capa de acciones que contiene la acción; es decir, con un mínimo, pero no es necesario, debido a que consideramos que cada acción aparece como mucho una vez en el *RPG*, como se explicó en el capítulo anterior.

De aquí en adelante, asumiremos que el nivel de las acciones que no aparecen en el *RPG* pertenece al intervalo $[final, \infty)$. Asimismo, el nivel de las proposiciones que no pertenecen al *RPG* se encuentra en el intervalo $(final, \infty)$.

Proposición 4.1. El *nivel* de una acción a es el máximo nivel de todas sus precondiciones:

$$nivel(a) = \max_{q \in pre(a)} \{nivel(q)\}$$

Demostración. Supongamos que $nivel(a) = n$. Dada la construcción del RPG, el nivel de una acción a es n cuando la acción pertenece a la capa de acciones A_n, cosa que sólo sucede cuando todas las precondiciones de a son ciertas en P_n. Para todas las proposiciones ciertas en P_n, y en particular para las precondiciones de a, se cumple por definición que su nivel es menor o igual que n: $\forall q \in pre(a), level(q) \leq n$. Las capas de proposiciones crecen monótonamente y tienen índices incrementales, por lo tanto, n es el máximo nivel de todas las precondiciones de a. Si el máximo nivel de todas las precondiciones de a fuera menor que n, la acción a se habría aplicado en una capa anterior y $nivel(a) < n$. Si por el contrario, el máximo nivel de las precondiciones de a fuera mayor que n, a no sería aplicable en la capa A_n y su nivel sería mayor que n, lo que contradice la hipótesis de que $nivel(a) = n$. □

Teorema 4.1. El nivel de una proposición p que aparece en el RPG de METRIC-FF, $nivel(p)$, es exactamente $h_{max\text{-}unit}(p, s)$[1],donde s es el estado evaluado.

$$nivel(p) = h_{max\text{-}unit}(p, s), \forall p \in P_i$$

Demostración. (por inducción sobre las precondiciones)

(a) Caso base: la proposición es cierta para proposiciones ciertas en en el estado evaluado, es decir, con nivel 0: $nivel(p) = 0 \Leftrightarrow h_{max\text{-}unit}(p, s) = 0$.
$nivel(p) = 0 \Leftrightarrow \min_{i \in [0 \ldots final]} \{i \mid p \in P_i\} = 0 \Leftrightarrow p \in P_0 \Leftrightarrow p \in s \Leftrightarrow h_{max\text{-}unit}(p, s) = 0$

(b) Caso inductivo. Demostraremos que la proposición también es cierta para todas las proposiciones p que no pertenecen al estado evaluado. Para ello, partiendo de la hipótesis de que la proposición se cumple para todas las precondiciones de las acciones que generan p, veremos que también es cierta para p.

Dado el algoritmo para construir el RPG, si el nivel de una proposición p es n (es decir, P_n es la primera capa de proposiciones en que la proposición es cierta), necesariamente de entre todas las acciones del dominio que añaden p (que denotaremos como $A(p)$), existe al menos una acción, a_p, que añade p y está situada en la capa A_{n-1}. Por lo tanto:

$$nivel(p) = nivel(a_p) + 1$$

Además, no existe ninguna otra acción que consiga p en ninguna capa de acciones anterior a A_{n-1}. Es decir, a_p es una de las acciones que consigue p en la capa de acciones con menor índice. Si este índice fuera menor que $(n-1)$, el nivel de p sería menor que n. Por el contrario, si este índice fuera mayor que $(n-1)$, no se habría conseguido p en la capa P_n y su nivel sería necesariamente mayor que n. De manera

[1] $h_{max\text{-}unit}$ se refiere a la heurística del máximo calculada asumiendo costes de acciones unitarios.

más formal:

$$\exists a_p \in A(p)[nivel(a_p) = nivel(p) - 1] \text{ y } \nexists a' \in A(p)[nivel(a') < nivel(p) - 1]$$

Así, se puede decir que la acción con menor nivel que consigue p se encuentra en el nivel A_{n-1}, y:

$$nivel(p) = nivel(a_p) + 1 = \min_{a \in A(p)} \{nivel(a)\} + 1$$

Dada la proposición (4.1), el nivel de una acción es el máximo nivel de sus precondiciones, por lo tanto, sustituyendo, queda:

$$nivel(p) = \min_{a \in A(p)} \left\{ \max_{q \in pre(a)} \{nivel(q)\} \right\} + 1$$

El mínimo de un conjunto más una constante sigue siendo el mismo si a todos los elementos se les suma la constante. Por tanto:

$$nivel(p) = \min_{a \in A(p)} \left\{ 1 + \max_{q \in pre(a)} \{nivel(q)\} \right\} \tag{4.5}$$

Aplicando la hipótesis de inducción, podemos decir que para todas las precondiciones q de las acciones que generan p se cumple que $nivel(q) = h_{max\text{-}unit}(q, s)$:

$$nivel(p) = \min_{a \in A(p)} \left\{ 1 + \max_{q \in pre(a)} \{h_{max\text{-}unit}(q, s)\} \right\}$$

Que es exactamente la definición de $h_{max\text{-}unit}(p, s)$. Por lo tanto:

$$nivel(p) = h_{max\text{-}unit}(p, s)$$

□

En la demostración anterior, se deduce que a_p debe pertenecer necesariamente al nivel $nivel(p) - 1$ para minimizar la expresión en la ecuación (4.5), y por lo tanto para minimizar $h^a_{max\text{-}unit}(p, s)$. Así, se puede establecer el siguiente corolario:

Corolario 4.1. Dada una proposición p con $nivel(p) > 0$ y una acción a_p que añade p, la acción a_p minimiza la expresión $h^a_{max\text{-}unit}(p, s)$ si y sólo si $nivel(a_p) = nivel(p) - 1$:

$$a_p \in \arg\min_{a \in A(p)} h^a_{max\text{-}unit}(p, s) \Leftrightarrow nivel(a_p) = nivel(p) - 1$$

Proposición 4.2. Si el problema relajado tiene solución, $h_{max\text{-}unit}(\mathcal{G}, s)$ es exactamente el índice la primera capa de proposiciones que contiene las metas $\mathcal{G}$ del problema, P_{final}:

$$h_{max\text{-}unit}(\mathcal{G}, s) = final$$

Demostración. Si el problema relajado tiene solución, todas las metas del problema aparecen en el *RPG*. Dada la construcción del *RPG*, para cada meta g existe una capa de proposiciones, P_i, tal que $nivel(g) = i$. Además, existe al menos una meta g_j que cumple que $nivel(g_j) = final$. En otro caso, la expansión habría terminado con un número de niveles menor. De acuerdo con el teorema (4.1), $\forall g \in \mathcal{G}, nivel(g) = h_{max\text{-}unit}(g, s)$. Los índices de las capas son incrementales, por lo tanto, el máximo índice es $final$:

$$h_{max\text{-}unit}(\mathcal{G}, s) = \max_{g \in \mathcal{G}} \{h_{max\text{-}unit}(g, s)\} = \max_{g \in \mathcal{G}} \{nivel(g)\} = final$$

□

Pasemos ahora a estudiar el proceso de extracción del plan relajado.

Teorema 4.2. Las acciones en el plan relajado de METRIC-FF, generado sin considerar efectos secundarios, son exactamente las acciones en el conjunto $\pi^+_{max\text{-}unit}(\mathcal{G}, s)$, siempre y cuando los posibles empates en la selección de acciones mediante la ecuación $a_p \in \arg\min_{a \in A(p)} h^a_{max\text{-}unit}(p, s)$ se resuelvan de la misma manera.

Demostración. Según el proceso de extracción de METRIC-FF, sin tener en consideración el tratamiento de efectos secundarios, y suponiendo que el proceso de construcción del *RPG* acaba con éxito (todas las metas se encuentran en la última capa), se puede decir que para que una acción a_p pertenezca al plan relajado se tiene que cumplir necesariamente [(a ó b) y c], donde:

(a) Caso base: (a_p añade una meta p del problema) $\exists p \in add(a_p)$ tal que $p \in \mathcal{G}$, o

(b) Caso recursivo: (a_p añade una (sub)meta p) $\exists p \in add(a_p)$ tal que $p \in pre(a')$ y $a' \in RP$ y

(c) a_p es, de entre todas las acciones que añaden p, aquélla situada en el menor nivel.

Según la demostración del teorema (4.1) la acción o acciones a_p que consiguen p en el menor nivel cumplen que $nivel(a_p) = nivel(p) - 1$. Tomando el corolario (4.1), se puede decir que a_p es una acción que minimiza $h^a_{max\text{-}unit}(p, s)$:

$$a_p \in \arg\min_{a \in A(p)} h^a_{max\text{-}unit}(p, s)$$

La selección de acciones para conseguir (sub)metas se hace de forma recursiva, de manera que una vez incluida en el plan relajado la acción que consigue una determinada (sub)meta, se aplica el mismo proceso para incluir las acciones que añaden cada una de sus precondiciones. Así, las acciones que pertenecen al plan relajado siempre minimizan $h^a_{max\text{-}unit}(p, s)$ de la (sub)meta que generan. Por lo tanto el conjunto de acciones en el plan relajado es el conjunto $\pi_{max}(\mathcal{G}, s)$ ampliado: $\pi^+_{max}(\mathcal{G}, s)$, de la definición (4.6), aunque calculado asumiendo costes unitarios en las acciones. Denominaremos a este conjunto $\pi^+_{max\text{-}unit}(\mathcal{G}, s)$. El conjunto $\pi^+_{max\text{-}unit}(\mathcal{G}, s)$ contiene las acciones a_p que consiguen cada (sub)meta del conjunto

$\mathcal{G}$ más las acciones que consiguen las precondiciones de acciones ya seleccionadas. Siempre las acciones elegidas minimizan $h^a_{max\text{-}unit}(p, s)$.

□

El teorema anterior significa que las acciones con factor $N'(a)$ igual a uno en el cálculo de la heurística son las acciones del conjunto $\pi^+_{max\text{-}unit}(\mathcal{G}, s)$. Por lo tanto:

Corolario 4.2. La heurística de METRIC-FF, sin considerar efectos secundarios, coincide con la heurística del máximo ampliada cuando ésta se calcula utilizando el conjunto $\pi^+_{max\text{-}unit}(\mathcal{G}, s)$, siempre y cuando los posibles empates que se produzcan en la selección de acciones se resuelvan de la misma manera:

$$h_{basic\text{-}mff\text{-}NSE}(\mathcal{G}, s) = \sum_{a \in \pi^+_{max\text{-}unit}(\mathcal{G},s)} coste(a)$$

Respecto a la resolución de empates, concretamente METRIC-FF selecciona la acción con menor dificultad. La heurística la dificultad (definida en el estado de la cuestión, ecuación (2.2), página 31) también se puede definir en función de $h_{max\text{-}unit}$:

Definición 4.12. (Dificultad en h_{mff}) La heurística de la *dificultad* se puede definir como:

$$\text{dificultad}(a) = \sum_{q \in pre(a)} h_{max\text{-}unit}(q, s)$$

Esta definición se puede establecer dado que según la ecuación(2.2):

$$\text{dificultad}(a) = \sum_{q \in pre(a)} \min_{i \in [0 \ldots final]} \{i \mid q \in P_i\}$$

Y,

$$\text{dificultad}(a) = \sum_{q \in pre(a)} \min_{i \in [0 \ldots final]} \{i \mid q \in P_i\} = \sum_{q \in pre(a)} nivel(q) = \sum_{q \in pre(a)} h_{max\text{-}unit}(q, s)$$

4.4. Heurísticas de SAPA

En este apartado se analizan las heurísticas de SAPA, de forma similar al análisis del apartado anterior para METRIC-FF, y con el mismo objetivo: caracterizar de forma declarativa qué acciones tienen un factor $N'(a) = 1$ en el cálculo de las heurísticas. En el análisis, únicamente se tiene en cuenta el procedimiento básico de propagación de costes, por lo tanto se ignora el tratamiento de todos los aspectos relacionados con la planificación temporal y numérica. De la misma forma se ignoran las mejoras que SAPA introduce en la heurística, como son la consideración de *mutex* estáticos y el método que ajusta la heurística utilizando información sobre recursos numéricos (estado de la cuestión, sección 2.4.4.3, página 50). Estas mejoras se pueden ignorar sin pérdida de generalidad ya que se pueden incluir en cualquiera de las heurísticas estudiadas.

A continuación se analiza la heurística de SAPA tanto con propagación de costes de las precondiciones a los efectos de las acciones a través de un máximo como con propagación de costes aditiva. De aquí en adelante nos referiremos a ellas como $h_{basic\text{-}sapa\text{-}max}$ y $h_{basic\text{-}sapa\text{-}add}$ respectivamente. Veremos que estas heurísticas están estrechamente relacionadas con la *heurística del máximo ampliada* y con la *heurística aditiva restringida* respectivamente.

Las definiciones en este apartado son relativas al *RPG* que construye SAPA (explicado en el estado de la cuestión, sección 2.4.4.3, página 50), y por lo tanto, distintas a las que realizaron respecto a METRIC-FF.

Definición 4.13. (Coste de conseguir una proposición en h_{sapa}) El **coste** de conseguir una proposición p del *RPG* es el coste mínimo asociado a la proposición en el *RPG*:

$$coste(p) = \min_{i \in [0 \ldots final]} \{coste(p, i) \mid p \in P_i\}$$

Definición 4.14. (Nivel de una proposición en h_{sapa}) El **nivel** de una proposición p en el *RPG* es el índice i de la primera capa de proposiciones que contiene la proposición con coste mínimo.

$$nivel(p) = \min_{i \in [0 \ldots final]} \{i \mid p \in P_i \text{ y } coste(p, i) = coste(p)\}$$

Definición 4.15. (Coste de aplicar una acción en h_{sapa}) El **coste** de aplicar acción a del *RPG* es el coste mínimo asociado a la acción en el *RPG*:

$$coste\_app(a) = \min_{i \in [0 \ldots final]} \{coste(a, i) \mid a \in A_i\}$$

Definición 4.16. (Nivel de una acción en h_{sapa}) El **nivel** de una acción a en el *RPG* es el índice i de la primera capa de acciones que contiene la acción con coste mínimo.

$$nivel(a) = \min_{i \in [0 \ldots final]} \{i \mid coste(a, i) = coste\_app(a)\}$$

Recuérdese que en el proceso de construcción del *RPG* de SAPA, la misma acción se puede aplicar de nuevo si el coste de conseguir sus precondiciones se decrementa, de modo que la misma acción puede aparecer en varios niveles.

En el caso de SAPA, el *RPG* se construye hasta el punto fijo. De aquí en adelante, se asumirá que el coste de las proposiciones que no aparecen en el *RPG* es infinito. De igual forma, se asumirá que el coste de aplicar acciones que no aparecen en el *RPG* es infinito.

Teorema 4.3. El coste de una proposición p del *RPG* de SAPA para evaluar el estado s es exactamente

- $h_{max}(p, s)$, para propagación de costes con el máximo:

$$coste(p) = h_{max}(p, s)$$

- $h_{add}(p, s)$, para propagación de costes aditiva:

$$coste(p) = h_{add}(p, s)$$

Demostración. (por inducción sobre las precondiciones)

(a) Caso base: la proposición se cumple para todas las proposiciones p en el estado evaluado.

Las proposiciones p en el estado evaluado pertenecen a la capa P_0 y tienen $coste(p, 0) = 0$. Por lo tanto, la capa inicial es la que minimiza $coste(p, i)$ y así, según la definición (4.13), $coste(p) = coste(p, 0) = 0$

Por otro lado, dadas las definiciones de h_{max} y h_{add}, $p \in s$ implica que:

- (prop. máximo) $h_{max}(p, s) = 0$
- (prop. aditiva) $h_{add}(p, s) = 0$

La implicación no es cierta en la dirección contraria, dado que tanto $h_{max}(p, s)$ como $h_{add}(p, s)$ pueden ser cero para proposiciones que no están en el estado evaluado, cuando el dominio contiene acciones con coste cero. Sin embargo, se puede afirmar que cuando $p \in s$:

- (prop. máximo) $coste(p) = h_{max}(p, s) = 0$
- (prop. aditiva) $coste(p) = h_{add}(p, s) = 0$

(b) Caso inductivo: la proposición se cumple para proposiciones que no están en el estado evaluado. Se demostrará que asumiendo que es cierto para todas las precondiciones q de las acciones que generan p, también es cierto para p.

Supongamos que $nivel(p) = n$. Si n es el primer nivel i en el que $coste(p, i)$ es mínimo, entonces $coste(p) = coste(p, n)$. En este caso, existe al menos una acción a_p que consigue p en la capa A_{n-1}, con $coste(a_p, n-1) = coste(p, n)$, así:

$$coste(p) = coste(a_p, n-1)$$

Además no existe ninguna otra acción que consiga p en ninguna otra capa, con un coste menor que $coste(a_p, n-1)$. Es decir, a_p es una de las acciones que consigue p con mínimo coste. Si existiera esta acción, el coste de p sería menor que $coste(p, n)$, lo que contradice la hipótesis de partida de que $nivel(p) = n$. Más formalmente:

$$\exists a_p \in A(p)[a_p \in A_{n-1} \wedge coste(a_p, n-1) = coste(p, n) = coste(p)], \text{ y}$$

$$\nexists a' \in A(p)[\exists i(a' \in A_i \wedge coste(a', i) < coste(a_p, n-1))]$$

Podría existir también alguna acción a'_p que genere p, situada en una capa A_m con $m > n-1$ para la que se cumpla que $coste(a'_p, m) = coste(p, m) = coste(p, n)$. Sin embargo esto no contradice el hecho de que deba existir necesariamente una en la capa A_{n-1}.

Por lo tanto, podemos decir que el coste mínimo de todas las acciones que consiguen p en cualquier capa, es el coste de aplicar la acción a_p que produce p en la capa $n-1$, que es a su vez el mínimo coste de a_p en cualquier capa. Por lo tanto:

$$coste(p) = coste(a_p, n-1) = coste\_app(a_p) = \min_{a \in A(p), i \in [0,\ldots,final]} \{coste(a,i)\}$$

Lo que utilizando la definición (4.15) se puede reescribir como:

$$coste(p) = \min_{a \in A(p)} \{coste\_app(a)\} \tag{4.6}$$

La demostración continúa ahora analizando por separado los dos casos de propagación de costes: propagación con el máximo y aditiva:

- (prop. máximo) Considerando la propagación de costes que se realiza en el algoritmo para el caso de propagación con el máximo, el coste de aplicar una acción a en una capa i, $coste(a,i)$ es la suma del coste de la precondición q con máximo coste en P_i más el coste propio de la acción, $coste(a)$:

  $$coste(a,i) = coste(a) + \max_{q \in pre(a), i \in [0,\ldots,final]} \{coste(q,i)\}$$

  Si el coste de una acción en la capa i es mínimo, entonces $coste\_app(a) = coste(a,i)$. Esto ocurre en todas las capas en las que el coste de la precondición con mayor coste en la capa P_i es mínimo. En particular ocurre en la capa en la que el coste de todas las precondiciones es mínimo, con lo cual:

  $$coste\_app(a) = coste(a) + \max_{q \in pre(a)} \{coste(q)\}$$

  Por lo que la ecuación (4.6) se puede reescribir como:

  $$coste(p) = \min_{a \in A(p)} \left\{ coste(a) + \max_{q \in pre(a)} \{coste(q)\} \right\}$$

  La hipótesis de inducción asume que para todas las precondiciones q de las acciones que generan p, $coste(q) = h_{max}(q,s)$. Aplicándola, tenemos:

  $$coste(p) = \min_{a \in A(p)} \left\{ coste(a) + \max_{q \in pre(a)} \{h_{max}(q,s)\} \right\} \tag{4.7}$$

  Que es exactamente la definición de $h_{max}(p,s)$. Por lo tanto,

  $$coste(p) = h_{max}(p,s)$$

- (prop. aditiva) Para el caso de propagación de costes aditiva la demostración es similar, con la única diferencia de que el máximo es una suma. Así, según el algoritmo, la actualización de los costes de aplicar las acciones en cada capa se

realiza con:

$$coste(a,i) = coste(a) + \sum_{q \in pre(a), i \in [0,\ldots,final]} \{coste(q,i)\}$$

El coste de una acción en la capa i es mínimo sólo cuando lo es la suma de los costes de sus precondiciones. Esta suma es mínima sólo cuando el coste de cada precondición es mínimo, por lo que, análogamente al caso del máximo:

$$coste\_app(a) = coste(a) + \sum_{q \in pre(a)} \{coste(q)\}$$

Reescribiendo la ecuación (4.6) y aplicando la hipótesis de inducción tenemos la definición de la heurística aditiva:

$$coste(p) = \min_{a \in A(p)} \left\{ coste(a) + \sum_{q \in pre(a)} \{h_{add}(q,s)\} \right\} = h_{add}(p,s)$$

□

En la demostración del teorema anterior, se observa que para el caso de propagación con el máximo, la acción a_p es, de entre todas las que generan p, la que tiene menor coste de aplicación. Además, esta acción minimiza la expresión en la ecuación (4.6), y por lo tanto también la de la ecuación (4.7). Minimizar esta última es lo mismo que minimizar $h^a_{max}(p,s)$. Para propagación aditiva el razonamiento es análogo. Por lo tanto, se puede establecer siguiente corolario.

Corolario 4.3. Dada un proposición p con $nivel(p) > 0$ y una acción a_p que añade p, a_p es la acción con menor coste de aplicación que genera p, $coste(p) = coste\_app(a)$, si y solo si:

- minimiza la expresión $h^a_{max}(p,s)$, para propagación de costes con el máximo:

$$a_p \in \arg\min_{a \in A(p)} h^a_{max}(p,s)$$

- minimiza la expresión $h^a_{add}(p,s)$, para propagación de costes aditiva:

$$a_p \in \arg\min_{a \in A(p)} h^a_{add}(p,s)$$

Respecto al resultado del proceso de extracción en SAPA, se puede enunciar el siguiente teorema.

Teorema 4.4. Las acciones en el plan relajado de SAPA, considerando exclusivamente el proceso básico de propagación de costes, son exactamente las acciones en el conjunto

- $\pi^{+}_{max}(\mathcal{G}, s)$, en el caso de propagación de costes con el máximo, y
- $\pi_{add}(\mathcal{G}, s)$, en el caso de propagación de costes aditiva,

siempre y cuando los empates en la selección de acciones se resuelvan de la misma forma.

Demostración. El proceso de extracción en SAPA es similar al de METRIC-FF, con la particularidad de que en lugar de elegir las acciones situadas en menor nivel, se eligen aquéllas con menor coste de aplicación. Así, para que una acción $a_p \in \mathcal{A}$ pertenezca en este caso al plan relajado se tiene que cumplir necesariamente [(a ó b) y c], donde:

(a) Caso base: (a_p añade una meta p del problema) $\exists p \in add(a_p)$ tal que $p \in \mathcal{G}$, o

(b) Caso recursivo: (a_p añade una (sub)meta p) $\exists p \in add(a_p)$ tal que $p \in pre(a')$ y $a' \in RP$ y

(c) a_p es, de entre todas las acciones que añaden p, la que tiene **menor coste asociado**.

Siguiendo el corolario (4.3), las acciones a_p que generan p minimizando su coste, son aquéllas que minimizan la expresión de $h^{a}_{max}(p, s)$ para el caso de propagación con el máximo, y $h^{a}_{add}(p, s)$ para el caso de propagación aditiva. Al igual que en el caso de METRIC-FF, la selección de acciones para conseguir (sub)metas se hace de forma recursiva, de manera que el conjunto de acciones en el plan relajado que se obtiene es exactamente:

- $\pi^{+}_{max}(\mathcal{G}, s)$, para propagación con el máximo, y
- $\pi_{add}(\mathcal{G}, s)$, para propagación aditiva.

□

A partir de este teorema y considerando las definiciones de la *heurística del máximo amplicada* y la *heurística aditiva restringida*, definiciones (4.8 y 4.9) se deduce el siguiente resultado.

Corolario 4.4. La heurística de SAPA, considerando exclusivamente el proceso básico de propagación de costes y que la resolución de empates en la selección de acciones se resuelve de la misma manera, coincide:

- con la *heurística del máximo ampliada*, en el caso de propagación de costes con el máximo:
$$h_{basic\text{-}sapa\text{-}max}(\mathcal{G}, s) = h^{+}_{max}(\mathcal{G}, s)$$
- con la *heurística aditiva restringida*, en el caso de propagación de costes aditiva:
$$h_{basic\text{-}sapa\text{-}add}(\mathcal{G}, s) = h^{-}_{add}(\mathcal{G}, s)$$

4.5. Heurísticas h_{level}

En este apartado se realiza un análisis similar a los anteriores para el caso de una expansión del *RPG* en niveles de coste. Esta expansión se utiliza en la heurística h_{level} y también en el procedimiento para construir el *RPG* de SIMPLANNER. Al analizar estos dos casos desde el punto de vista algorítmico en el capítulo anterior, se descubrió que los dos algoritmos son equivalentes, con la diferencia principal de que en el caso de h_{level} la propagación de costes se realiza a través de un máximo, mientras que en el caso de SIMPLANNER es aditiva. Esto significa que, mediante el algoritmo para construir el *RPG* de h_{level}, pero con una propagación de costes aditiva, se obtiene el mismo resultado que mediante el algoritmo de SIMPLANNER. Por este motivo, en este apartado se realiza un análisis conjunto a partir del algoritmo de h_{level}, y teniendo en cuenta los dos casos de propagación de costes.

Al igual que en casos anteriores consideraremos el proceso básico de extracción, en el que no se tienen en cuenta los efectos secundarios. Nos referiremos a las heurísticas resultado como $h_{level\text{-}max\text{-}NSE}$ y $h_{level\text{-}add\text{-}NSE}$ respectivamente.

El *RPG* de h_{level} comparte algunas características con el de METRIC-FF, como que cada acción puede aparecer a lo sumo en una única capa y que se construye hasta el primer nivel que contiene las metas. Esto hace que algunas deficiones que se relizaron en el caso de METRIC-FF sean útiles aquí. Concretamente, las definiciones de *nivel de una proposición*, definición (4.10) y *nivel de una acción*, definición (4.11). Las demás definiciones son específicas del *RPG* de h_{level}.

Definición 4.17. (Coste de una proposición en h_{level}) El **coste** de una proposición p del *RPG* es el límite de coste, $limite\_coste_i$, la capa de proposiciones P_i en que aparece por primera vez la proposición:

$$coste(p) = limite\_coste_i, \text{ donde } i = nivel(p)$$

Definición 4.18. (Coste de aplicar una acción en h_{level}) El coste de aplicar una acción a que está situada en la capa A_i del *RPG*, $nivel(a) = i$, es el límite de coste de la capa inmediatamente posterior, $limite\_coste_{i+1}$:

$$coste\_app(a) = limite\_coste_{i+1},\ a \in A_i$$

De aquí en adelante, asumiremos que el coste de proposiciones y de aplicar acciones que no pertenecen al *RPG* se encuentra en el intervalo $(limite\_coste_{final}, \infty)$.

Propiedad 4.2. Dadas dos acciones a_1 y a_2 en el *RPG* de h_{level}, el coste de aplicar a_1 es menor o igual que el coste de aplicar a_2 si y solo si el nivel de a_1 es menor o igual que el nivel de a_2:

$$coste\_app(a_1) \leq coste\_app(a_2) \Leftrightarrow nivel(a_1) \leq nivel(a_2)$$

Demostración. El coste de aplicar una acción a es por definición $limite\_coste_{nivel(a)}$. El *RPG* se construye de tal manera que el límite de coste nunca se decrementa con los

niveles. Por lo tanto, si $nivel(a_1) \leq nivel(a_2)$, necesariamente $limite\_coste_{nivel(a_1)} \leq limite\_coste_{nivel(a_2)}$ □

Proposición 4.3. El coste de aplicar una acción a del RPG es:

- el coste de la acción a más el coste de su precondición más cara, para propagación de costes con el máximo:

$$coste\_app(a) = coste(a) + \max_{q \in pre(a)} \{coste(q)\}$$

- el coste de la acción a más las suma de los costes de todas sus precondiciones, para propagación de costes aditiva:

$$coste\_app(a) = coste(a) + \sum_{q \in pre(a)} \{coste(q)\}$$

Demostración. (diferenciando los dos casos de propagación de costes)

Supongamos que la acción a tiene $nivel(a) = n$. Por la definición (4.18) el coste de aplicar una acción a con $nivel(a) = n$ es $limite\_coste_{n+1}$:

$$coste\_app(a) = limite\_coste_{n+1}$$

- (prop. máximo) Dado el algoritmo para construir el RPG en $h_{level\text{-}max}$, si el nivel de una acción a es n, el límite de coste de la siguiente capa de proposiciones se calcula como:

$$limite\_coste_{n+1} = coste(a) + limite\_coste_k \tag{4.8}$$

  donde k es el índice de la primera capa de proposiciones en que la acción es aplicable. Es decir, en la iteración k-ésima del algoritmo, la acción a se ha introducido en la lista *AbiertaApp* (ver algoritmo 3.4, sección 3.2, página 65). Sin embargo, la acción no entra en el RPG hasta el nivel $n \geq k$. Recuérdese que la inclusión de acciones en el RPG se realiza cuando tienen coste mínimo. Por lo tanto, el hecho de que una acción sea aplicable en un nivel no implica que la acción se incluya en ese nivel.

  Si la acción a es aplicable a partir del nivel de proposiciones P_k y no en un nivel anterior es porque todas sus precondiciones son ciertas en esa capa, lo que significa que aparecen por primera vez en el grafo en una capa igual o anterior a k:

$$\forall q \in pre(a), nivel(q) \leq k$$

  Además, existe al menos una precondición q' que aparece por primera vez en al capa k, $nivel(q') = k$. En otro caso la acción se habría podido aplicar en algún nivel anterior. Por definición, el coste de esa precondición es el límite de coste de la capa P_k:

$$\exists q' \in pre(a), coste(q') = limite\_coste_k$$

Los límites de coste no pueden decrementarse con las capas, por lo que:

$$limite\_coste_k = \max_{q \in pre(a)} \{coste(q)\}$$

y entonces, sustituyendo en la ecuación (4.8) queda:

$$limite\_coste_{n+1} = coste(a) + \max_{q \in pre(a)} \{coste(q)\}$$

Con lo que:

$$coste\_app(a) = coste(a) + \max_{q \in pre(a)} \{coste(q)\}$$

- (prop. aditiva) Dado el algoritmo para construir el RPG en $h_{level\text{-}add}$, si el nivel de una acción a es n, el límite de coste de la siguiente capa de proposiciones se calcula como:

$$limite\_coste_{n+1} = coste(a) + \sum_{q \in pre(a)} \text{mín}\{limite\_coste_i \mid q \in P_i\} \tag{4.9}$$

Según la definición (4.10), de entre todas las capas que contienen una proposición q, la capa con menor índice es $nivel(q)$. Además, de todas estas capas, $limite\_coste_{nivel(q)}$ es el menor límite de coste, ya que los límites de coste no se decrementan con las capas. Por lo tanto:

$$\text{mín}\{limite\_coste_i \mid q \in P_i\} = limite\_coste_{nivel(q)}$$

Por la definición (4.17), $limite\_coste_{nivel(q)}$ es exactamente $coste(q)$:

$$\text{mín}\{limite\_coste_i \mid q \in P_i\} = coste(q)$$

Sustituyendo en la ecuación (4.9):

$$limite\_coste_{n+1} = coste(a) + \sum_{q \in pre(a)} \{coste(q)\}$$

Y entonces:

$$coste\_app(a) = coste(a) + \sum_{q \in pre(a)} \{coste(q)\}$$

□

Teorema 4.5. El coste de una proposición p que se encuentra en el RPG de h_{level} para evaluar el estado s es exactamente:

- $h_{max}(p, s)$, para propagación de costes con el máximo:

$$coste(p) = h_{max}(p, s)$$

- $h_{add}(p, s)$, para propagación de costes aditiva:

$$coste(p) = h_{add}(p, s)$$

Demostración. (*por inducción sobre las precondiciones*)

(a) Caso base: la proposición es cierta para todos los hechos en el estado evaluado.

Los hechos p en el estado evaluado pertenecen a P_0, por lo tanto $nivel(p) = 0$ y $coste(p) = limite\_coste_0 = 0$.

Dadas las definiciones de h_{max} y h_{add}, para toda proposición en el estado evaluado el coste estimado por ambas heurísticas es cero. Por lo tanto, se puede decir que:

- (prop. máximo): $coste(p) = h_{max}(p, s) = 0$
- (prop. aditiva): $coste(p) = h_{max}(p, s) = 0$
  El valor de $h_{max}(p, s)$ y de $h_{add}(p, s)$ también puede ser cero para proposiciones que no están en el estado evaluado en dominios en los que hay acciones con coste cero.

(b) Caso inductivo. Se demostrará que la proposición también es cierta para hechos que no están en el estado evaluado. Para ello, se parte de la hipótesis de que es cierta para todas las precondiciones de las acciones que generan p, y se demuestra que también lo es para p.

Supongamos que $nivel(p) = n$, entonces $coste(p) = limite\_coste_n$, y necesariamente existe al menos una acción a_p que genera p con $nivel(a_p) = n - 1$. Según la definición (4.18), el coste de aplicar una acción es el límite de coste de la capa inmediatamente posterior a su nivel. Por lo tanto, $coste\_app(a_p) = limite\_coste_n$. Con lo que se puede afirmar que:

$$coste(p) = coste\_app(a_p) = limite\_coste_n$$

Por la construcción del RPG, al ser P_n la primera capa que contiene a p, no existe ninguna otra acción que consiga p en un nivel anterior a $nivel(a_p)$. O lo que es lo mismo según la propiedad (4.2), no existe ninguna otra acción con menor coste de aplicación que genere p. Así:

$$coste(p) = coste\_app(a_p) = \min_{a \in A(p)} \{coste\_app(a)\} \tag{4.10}$$

A partir de este momento la demostración diferencia los casos de propagación con el máximo y aditiva.

- (prop. máximo)
  Tomando la proposición (4.3) para el caso de propagación con el máximo, tenemos que para las acciones en el RPG:

$$coste\_app(a) = coste(a) + \max_{q \in pre(a)} \{coste(q)\}$$

Asumiendo que esta ecuación se cumple también para las acciones que no aparecen en elRPG, podemos sustituir en la ecuación (4.10), con lo que queda:

$$coste(p) = \min_{a \in A(p)} \left\{ coste(a) + \max_{q \in pre(a)} \{coste(q)\} \right\}$$

Aplicando la hipótesis de inducción, se consigue la definición de $h_{max}(p, s)$:

$$coste(p) = \min_{a \in A(p)} \left\{ coste(a) + \max_{q \in pre(a)} \{h_{max}(p, s)\} \right\}$$

Con lo que:

$$coste(p) = h_{max}(p, s)$$

- (prop. aditiva)
  Para este caso, la demostración es análoga al anterior, pero tomando la proposición (4.3) para el caso de propagación aditiva:

  $$coste\_app(a) = coste(a) + \sum_{q \in pre(a)} \{coste(q)\}$$

  Sustituyendo en la ecuación (4.10) y aplicando la hipótesis de inducción se llega a:

  $$coste(p) = \min_{a \in A(p)} \left\{ coste(a) + \sum_{q \in pre(a)} \{h_{add}(p, s)\} \right\}$$

  que es la definición de $h_{add}(p, s)$. Por lo tanto:

  $$coste(p) = h_{add}(p, s)$$

□

Ya que a_p minimiza la expresión de la heurística del máximo y la de la heurística aditiva, respectivamente a los dos casos de propagación, de la demostración anterior se deduce que el corolario (4.3) (página 99), enunciado para el caso de las heurísticas de SAPA, también es cierto para las heurísticas h_{level}.

Proposición 4.4. Si el problema relajado tiene solución, el límite de coste de la primera capa de proposiciones que contiene todas las metas $\mathcal{G}$ del problema, $limite\_coste_{final}$, es exactamente

- $h_{max}(\mathcal{G}, s)$, para propagación de costes con el máximo:

  $$limite\_coste_{final} = h_{max}(\mathcal{G}, s)$$

- el máximo valor de $h_{add}(g, s)$ de las metas $g \in \mathcal{G}$ del problema, para propagación de

costes aditiva:

$$limite\_coste_{final} = \max_{g \in \mathcal{G}} \{h_{add}(g, s)\}$$

Demostración. Asumamos que todas las metas del problema aparecen en el RPG, es decir, el algoritmo termina con éxito: $\forall g \in \mathcal{G}, g \in P_{final}$. Entonces, para cada meta del problema g existe una capa de proposiciones P_i tal que $coste(g) = limite\_coste_i$. Además existe al menos una meta cuyo nivel es el último (P_{final}). En otro caso, el RPG tendría menos niveles.

- (prop. máximo) Dado el teorema (4.5) para propagación con el máximo, $\forall g \in \mathcal{G}, coste(g) = h_{max}(g, s)$. Además los límites de coste nunca se decrementan con las capas; por lo tanto el máximo es $limite\_coste_{final}$ y:

$$h_{max}(\mathcal{G}, s) = \max_{g \in \mathcal{G}} \{h_{max}(g, s)\} = \max_{g \in \mathcal{G}} \{coste(g)\} = limite\_coste_{final}$$

- (prop. aditiva) Dado el teorema (4.5), para propagación aditiva, $\forall g \in \mathcal{G}, coste(g) = h_{add}(g, s)$. Además los límites de coste nunca se decrementan con las capas; por lo tanto el máximo es $limite\_coste_{final}$ y:

$$\max_{g \in \mathcal{G}} \{h_{add}(g, s)\} = \max_{g \in \mathcal{G}} \{coste(g)\} = limite\_coste_{final}$$

□

Teorema 4.6. Las acciones en el plan relajado de h_{level}, sin considerar efectos secundarios y siempre y cuando los empates en la selección de acciones se resuelvan de la misma manera, son exactamente las acciones en el conjunto

- $\pi^{+}_{max}(\mathcal{G}, s)$, en el caso de propagación de costes con el máximo.
- $\pi_{add}(\mathcal{G}, s)$, en el caso de propagación de costes aditiva.

Demostración. El proceso de extracción del plan relajado de las heurísticas h_{level} es igual al de METRIC-FF. Así, sin tener en consideración el tratamiento de efectos secundarios, y suponiendo que el proceso de construcción del RPG acaba con éxito, se puede decir que para que una acción $a_p \in \mathcal{A}$ pertenezca al plan relajado se tiene que cumplir necesariamente [(a ó b) y c]:

(a) Caso base: (a_p añade una meta p del problema) $\exists p \in add(a_p)$ tal que $p \in \mathcal{G}$, o

(b) Caso recursivo: (a_p añade una (sub)meta p) $\exists p \in add(a_p)$ tal que $p \in pre(a')$ y $a' \in RP$ y

(c) a_p es, de entre todas las acciones que añaden p, aquélla situada en el menor nivel.

Supongamos que el nivel en que se encuentra una (sub)meta p es n, $nivel(p) = n$. La acción con menor nivel que genera n tiene necesariamente $nivel(a_p) = n-1$. Puesto que el nivel de p es n y no menor, no existe ninguna otra acción que genere p con nivel menor a $n-1$. Tomando la propiedad (4.2), al tener menor nivel, a_p es también la acción que genera p con menor coste de aplicación. Según el corolario (4.3), esta acción cumple:

- (prop. máximo)
$$a_p \in \arg\min_{a \in A(p)} h^a_{max}(p, s)$$
- (prop. aditiva)
$$a_p \in \arg\min_{a \in A(p)} h^a_{add}(p, s)$$

La selección de acciones para conseguir (sub)metas se hace de forma recursiva, de manera que una vez incluida en el plan relajado la acción que consigue una determinada (sub)meta se aplica el mismo proceso para incluir las acciones que añaden cada una de sus precondiciones. Así, considerando el plan relajado como un conjunto de acciones, su definición coincide con:

- (prop. máximo) $\pi^+_{max}(\mathcal{G}, s)$
- (prop. aditiva) $\pi_{add}(\mathcal{G}, s)$

□

Del teorema anterior se deduce el siguiente corolario.

Corolario 4.5. La heurística de h_{level}, sin considerar efectos secundarios y considerando que los empates en la selección de acciones se resuelven de la misma manera, coincide:

- con la *heurística del máximo ampliada*, en el caso de propagación de costes con el máximo:
$$h_{level\text{-}max\text{-}NSE}(\mathcal{G}, s) = h^+_{max}(\mathcal{G}, s)$$
- con la *heurística aditiva restringida*, en el caso de propagación de costes aditiva:
$$h_{level\text{-}add\text{-}NSE}(\mathcal{G}, s) = h^-_{add}(\mathcal{G}, s)$$

La resolución de empates para la heurística h_{level} se realiza utilizando la heurística de la dificultad pero definida (en el siguiente apartado) en términos de las heurísticas del máximo ampliada y aditiva restringida.

4.6. Resolución de empates: heurística de la dificultad

La idea de la heurística de la dificultad que se utiliza en METRIC-FF para resolver parcialmente empates en la selección de acciones, se puede adaptar para que tenga en cuenta

los costes de las acciones, como se hizo en el capítulo anterior (sección 3.2, página 67). La adaptación se puede hacer de varias maneras. Siguiendo la idea de la dificultad en METRIC-FF, definimos a continuación una posible heurística de la dificultad para la heurística del máximo ampliada y para la heurística aditiva restringida, en la que en lugar de sumar los niveles de las precondiciones (como en METRIC-FF) se suman sus costes correspondientes:

Definición 4.19. (Dificultad para h^+_{max}) La heurística de la dificultad en el caso de h^+_{max} se puede definir como:

$$\text{dificultad}_{\text{h}^+_{max}}(a) = \sum_{p \in pre(a)} coste(p) = \sum_{p \in pre(a)} h_{max}(p, s)$$

Definición 4.20. (Dificultad para h^-_{add}) La heurística de la dificultad en el caso de h^-_{add} se puede definir como:

$$\text{dificultad}_{\text{h}^-_{add}}(a) = \sum_{p \in pre(a)} coste(p) = \sum_{p \in pre(a)} h_{add}(p, s)$$

La dificultad es otro concepto que se ha venido definiendo en función de características del *RPG*, que también tiene una descripción procedural. Aquí se muestra que también existe una definición declarativa para la dificultad, que la relaciona con la heurística del máximo o aditiva, según el caso.

4.7. Heurísticas que se pueden derivar del mismo *RPG*

Tras el análisis realizado en este capítulo se observa que del proceso de construcción, incremental o no, de un mismo *RPG* se pueden derivar distintas heurísticas.

Por ejemplo, de la construcción del *RPG* para calcular $h_{level\text{-}max}$ se puede obtener la heurística $h_{level\text{-}max}$ y la heurística h_{max}, que como indica la proposición (4.4) es el límite de coste de la última capa de proposiciones:

$$h_{max}(\mathcal{G}, s) = limite\_coste_{final}$$

También se podría obtener una nueva heurística que estime el coste de cada proposición con $h_{max}(p, s)$, es decir asumiendo que el coste de una acción viene dado por el coste de la precondición más cara más el coste de la acción; y sin embargo considere que el coste de conseguir las metas del problema se calcula como una suma de las estimaciones particulares. Denominaremos a esta heurística $h_{add\text{-}max}$:

$$h_{add\text{-}max}(\mathcal{G}, s) = \sum_{g \in \mathcal{G}} h_{max}(g, s)$$

que según el teorema (4.5) se puede escribir en función de características del *RPG* como:

$$h_{add\text{-}max}(\mathcal{G}, s) = \sum_{g \in \mathcal{G}} limite\_coste_{nivel(g)}$$

Los factores $N'(a)$ para la heurística $h_{add\text{-}max}(\mathcal{G}, s)$ serían mayores o iguales que uno, ya que se pueden repetir acciones, para todas las acciones del conjunto resultante de hacer la unión de los conjuntos $\pi_{max}(g, s)$ de cada meta $g \in \mathcal{G}$.

De la misma forma, del proceso de construcción del RPG para calcular $h_{level\text{-}add}$ se puede obtener la heurística $h_{level\text{-}add}$, la heurística h_{add}, que se calcula como la suma, por cada meta, del límite de coste de la capa que contiene por primera vez la meta:

$$h_{add}(\mathcal{G}, s) = \sum_{g \in \mathcal{G}} limite\_coste_{nivel(g)}$$

Y también una nueva heurística que considere que el coste de conseguir cada proposición es $h_{add}(p, s)$, y sin embargo considere que el coste de conseguir las metas del problema se calcula como el máximo de las estimaciones particulares. Denominaremos a esta heurística $h_{max\text{-}add}$:

$$h_{max\text{-}add}(\mathcal{G}, s) = \max_{g \in \mathcal{G}} \{h_{add}(g, s)\}$$

que según el teorema (4.5) es:

$$h_{max\text{-}add}(\mathcal{G}, s) = \max_{g \in \mathcal{G}} \left\{ limite\_coste_{nivel(g)} \right\}$$

que también es lo mismo, según la proposición (4.4) que el límite de coste de la última capa de proposiciones:

$$h_{max\text{-}add}(\mathcal{G}, s) = limite\_coste_{final}$$

Los factores $N'(a)$ para la heurística $h_{max\text{-}add}(\mathcal{G}, s)$ son mayores o iguales que uno para todas las acciones del conjunto resultante de seleccionar el conjunto para el que se maximiza el valor de $h_{add}(g, s)$ de entre los conjuntos $\pi_{add}(g, s)$ de cada meta $g \in \mathcal{G}$.

Ya que del mismo RPG se pueden obtener varias heurísticas, y la diferencia computacional entre obtener una y varias es mínima, quizás sería interesante explorar si el uso combinado de las mismas aporta algún beneficio sobre el uso individual. Esta tesis se ha orientado hacia el estudio de combinaciones de heurísticas de distinto tipo, numéricas, de poda y *lookahead*, pero no estudia combinaciones de heurísticas numéricas entre sí.

4.8. Conclusiones del análisis matemático

En este capítulo se han analizado desde un punto de vista matemático las heurísticas diseñadas para planificación basada en costes que se calculan utilizando grafos de planificación relajados. El objetivo era clarificar las diferencias teóricas entre ellas y su relación con las heurísticas *max* y *aditiva*. Este objetivo viene motivado por el hecho de que todas las heurísticas calculadas a partir de grafos de planificación relajados se han venido describiendo en la bibliografía desde un punto de vista procedural. Las descripciones procedurales describen formalmente cuál es el proceso para calcular el RPG, pero, en general, tienen el inconveniente de que no definen explícitamente qué están calculando. Dado que se

pueden diseñar algoritmos muy distintos para generar un mismo resultado, si las descripciones algorítmicas no se complementan con descripciones formales que definan qué calcula exactamente el algoritmo, se puede generar cierta confusión o incluso la creencia de que el resultado de ejecutar algoritmos distintos es diferente. En este capítulo se han tomado como punto de partida los algoritmos de expansión del *RPG* y de extracción de la solución al problema relajado, para generar a partir de ellos una descripción declarativa de las heurísticas. El resultado es que las heurísticas basadas en *RPGs* coinciden (es decir, son equivalentes desde un punto de vista teórico) o bien con la heurística del máximo ampliada (h^+_{max}) o bien con la heurística aditiva restringida (h^-_{add}), definidas al comienzo del capítulo. Por ejemplo, $h_{basic\text{-}sapa\text{-}max}$ es equivalente a $h_{level\text{-}max\text{-}NSE}$, ya que su definición coincide con la heurística del máximo ampliada; y $h_{basic\text{-}sapa\text{-}add}$ es equivalente a $h_{level\text{-}add\text{-}NSE}$ y también a la heurística h_a introducida recientemente (Keyder and Geffner, 2008), puesto que en los tres casos la definición coincide con la de la heurística aditiva restringida. Como resultado se puede afirmar que no es necesario el uso de *RPGs* para describir las heurísticas. De hecho se pueden implementar con algoritmos similares a los explicados en el estado de la cuestión, sección 2.3.3.1, página 25, donde, h_{level} sigue la idea de un *Dijkstra generalizado*.

Desde el punto de vista práctico, se puede considerar que heurísticas que resultan equivalentes pueden proporcionar distintos valores (aunque en general son muy similares) debido a que la equivalencia sólo es cierta si se asume que la implementación de cada una de ellas resuelve ciertos casos de empate de la misma manera. Los casos de empates se refieren a la selección de las acciones que minimizan una determinada expresión.

En resumen, todas las heurísticas para planificación basada en costes se calculan como:

$$h(s, \mathcal{G}) = \sum_{a \in \mathcal{A}} N'(a) \times coste(a)$$

donde $N'(a) \in \mathbb{N} \cup \{0\}$.

Y las heurísticas analizadas se diferencian en los siguientes aspectos:

(a) Cómo se calcula el valor final de la heurística: mientras que en las heurísticas *max*, *add*, *max-add* y *add-max* el valor final de la heurística se calcula directamente, a partir de los costes propagados para las proposiciones que son metas del problema, todas las heurísticas basadas en grafos de planificación calculan, como paso intermedio, un conjunto de acciones π (que ordenado convenientemente, respetando las restricciones causales, constituye una solución al problema relajado), y después extraen el valor de la heurística como la suma de los costes de las acciones en este conjunto. Es decir, las acciones con factor $N'(a)$ igual a uno en el cálculo de la heurística son las acciones del conjunto π.

(b) Cómo se agregan los costes de las precondiciones. En las heurísticas estudiadas esto se hace bien vía la función máximo o bien vía la función suma. Cuando el coste de las precondiciones se agrega con un máximo, π es el conjunto $\pi^+_{max}(\mathcal{G}, s)$ y cuando se agrega con una suma es $\pi_{add}(\mathcal{G}, s)$. Es decir, los conjuntos a partir de los cuales se calculan la heurística del máximo ampliada y la heurística aditiva restringida respectivamente.

(c) Si se tienen en cuenta los costes de las acciones para calcular el coste de cada proposición o, por el contrario, el proceso asume que el coste de cada acción es uno. El único caso en que se asumen costes unitarios es la heurística de METRIC-FF. Esto da lugar a un conjunto π calculado asumiendo costes unitarios, que para METRIC-FF es $\pi^{+}_{max\text{-}unit}(\mathcal{G}, s)$.

(d) Cómo se resuelven los empates a la hora de seleccionar entre varias acciones que consiguen una misma proposición. Actualmente esto se hace o bien de forma arbitraria, o bien utilizando una heurística que mide la dificultad de conseguir las precondiciones de cada acción. Aunque la forma particular de resolver estos empates afecta al cálculo de las heurísticas basadas en grafos de planificación, no afecta a los valores de las heurísticas *max*, *add*, *max-add* y *add-max*, dado que para calcular estas heurísticas se propagan directamente los costes de las acciones y no las acciones en sí.

La Tabla 4.1 muestra las definiciones matemáticas de las heurísticas estudiadas. La primera columna indica el nombre de la heurística; la segunda cómo se calcula; y la tercera los factores $N'(a)$ correspondientes. Las heurísticas en la misma fila tienen la misma definición y el hecho de que sus valores coincidan depende de la resolución de empates. La heurística $h_{basic\text{-}mff\text{-}NSE}$ es una versión de la heurística del máximo ampliada, en la que el conjunto π se calcula asumiendo costes unitarios de las acciones.

Por otro lado, todos los conjuntos π excepto $\pi_{max}(\mathcal{G}, s)$ contienen acciones suficientes para generar todas (sub)metas que se producen en el proceso de regresión de metas. Por lo tanto, si las acciones de estos conjuntos se ordenan respetando el orden causal constituyen soluciones al problema relajado.

Respecto a la relación entre los valores de las heurísticas, las propiedades teóricas garantizan que siempre se cumplen las siguientes desigualdades:

$$h_{max} \leq h_{add}$$

$$h_{max} \leq h^{+}_{max}$$

$$h^{-}_{add} \leq h_{add}$$

$$h_{max\text{-}add} \leq h_{add}$$

$$h_{max} \leq h_{add\text{-}max}$$

Cuando en un problema de un dominio, para una (sub)meta p en el estado a evaluar se cumple que:

$$\arg\min_{a \in A(p)} h_{max}(p, s) = \arg\min_{a \in A(p)} h_{add}(p, s)$$

Tanto h^{+}_{max} como h^{-}_{add} seleccionan la misma acción para conseguir la (sub)meta, siempre que los empates se resuelvan de la misma manera.

La igualdad anterior es cierta, es decir en h^{+}_{max} y h^{-}_{add} se selecciona la misma acción para conseguir la proposición p, en los siguientes casos:

Heurística	$h(\mathcal{G}, s)$	**Factor** $N'(a)$
máximo (h_{max})	$\max_{g \in \mathcal{G}} \{h_{max}(g, s)\}$	$N'(a) = 1, \forall a \in \pi_{max}(\mathcal{G}, s)$
aditiva (h_{add})	$\sum_{g \in \mathcal{G}} \{h_{add}(g, s)\}$	$N'(a) \geq 1, \forall a \in \pi_{add}(\mathcal{G}, s)$
versión **máximo ampliada** (h^+_{max} con $\pi^+_{max\text{-}unit}$) $h_{basic\text{-}mff\text{-}NSE}$	$\sum_{\forall a \in \pi^+_{max\text{-}unit}(\mathcal{G}, s)} coste(a)$	$N'(a) = 1, \forall a \in \pi^+_{max\text{-}unit}(\mathcal{G}, s)$
máximo ampliada (h^+_{max}) $h_{basic\text{-}sapa\text{-}max}$ $h_{level\text{-}max\text{-}NSE}$	$\sum_{\forall a \in \pi^+_{max}(\mathcal{G}, s)} coste(a)$	$N'(a) = 1, \forall a \in \pi^+_{max}(\mathcal{G}, s)$
aditiva restringida (h^-_{add}) $h_{basic\text{-}sapa\text{-}add}$ $h_{level\text{-}add\text{-}NSE}$ h_a (Keyder and Geffner, 2008)	$\sum_{\forall a \in \pi_{add}(\mathcal{G}, s)} coste(a)$	$N'(a) = 1, \forall a \in \pi_{add}(\mathcal{G}, s)$
add-max ($h_{add\text{-}max}$)	$\sum_{g \in \mathcal{G}} \{h_{max}(g, s)\}$	$N'(a) \geq 1, \forall a \in \bigcup_{g \in \mathcal{G}} \pi_{max}(g, s)$
max-add ($h_{max\text{-}add}$)	$\max_{g \in \mathcal{G}} \{h_{add}(g, s)\}$	$N'(a) \geq 1, \forall a \in \pi_{add}(g, s)$ donde $g \in \arg\max_{g' \in \mathcal{G}} h_{add}(g', s)$

Tabla 4.1: Definición de las heurísticas estudiadas.

1. Cuando sólo existe una acción que genera p.
2. Cuando todas las acciones que generan p tienen una única precondición.
3. Cuando no se cumplen los anteriores, pero la distribución de costes del problema implica que la acción que minimiza $h_{max}(p,s)$ es también la que minimiza $h_{add}(p,s)$.

Cuando para todas las (sub)metas se da alguno de los tres casos anteriores, h^+_{max} y h^-_{add} coinciden. Si además:

$$\arg\min_{a \in A(p)} h_{max}(p,s) = \arg\min_{a \in A(p)} h_{max\text{-}unit}(p,s)$$

también coinciden con la heurística de METRIC-FF ($h_{basic\text{-}mff\text{-}NSE}$). Esto ocurre cuando el coste de los planes está directamente relacionado con el numero de operadores. Es decir, todos los planes con coste menor tienen también un número de operadores menor.

Existen muchos puntos comunes en el proceso para calcular las heurísticas estudiadas. Todas ellas dependen de las acciones que contiene el conjunto π correspondiente. Los conjuntos π se construyen de forma similar seleccionando una acción para conseguir cada (sub)meta y seleccionando acciones para conseguir sus precondiciones en un proceso recursivo. La acción que se selecciona es la que minimiza una determinada función (y esta función es la que marca la principal diferencia entre las heurísticas). Por ejemplo los conjuntos $\pi^+_{max}(\mathcal{G},s)$ y $\pi_{add}(\mathcal{G},s)$ tienen la misma definición y la única diferencia es que en el primer caso las acciones que se seleccionan son las que minimizan h_{max} mientras que en el segundo caso son las que minimizan h_{add}.

Por otro lado, en todos los casos se puede calcular la estimación heurística a partir del conjunto π correspondiente, excepto en los casos de la heurística aditiva y las heurísticas $h_{add\text{-}max}$ y $h_{max\text{-}add}$. En estos casos, el conjunto π por sí mismo no es suficiente para determinar los factores $N'(a)$ de las acciones que se tienen en cuenta en el cálculo de la heurística. Es necesario conocer el número de veces que hay que sumar el coste de cada acción.

Los dos párrafos anteriores junto con el hecho de que actualmente no existe una definición unificada de las heurísticas estudiadas motivan el capítulo siguiente, en el cual se presenta una definición generalizada que unifica los puntos comunes y marca claramente las diferencias entre ellas. La definición unificada cubre, además de las heurísticas analizadas en este capítulo, otras heurísticas, como la familia de heurísticas h^m (Haslum and Geffner, 2000) y otras de reciente aparición como son la heurística *set-additive* (Keyder and Geffner, 2008) y la heurística h_{pmax} (Mirkis and Domshlak, 2007) .

Capítulo 5

Descripción unificada de las heurísticas

En este capítulo de propone una descripción unificada de algunas heurísticas que se basan en ignorar total o parcialmente los efectos negativos de las acciones. La idea es propagar, de las precondiciones de las acciones a sus efectos, información lo suficientemente general como para generar la mayoría de las heurísticas existentes. La descripción generalizada constituye un marco para definir y comparar las heurísticas existentes, y se puede utilizar también para generar heurísticas nuevas.

5.1. Conceptos previos

En la definición generalizada utilizaremos *multiconjuntos* (Blizard, 1989). Los multiconjuntos se diferencian de los conjuntos en que un mismo miembro puede aparecer varias veces. Los elementos de un multiconjunto se pueden representar como un par de la forma $(e, m(e))$, donde e es un miembro del multiconjunto, y $m(e)$ es un valor entero que representa su multiplicidad (es decir, el número de veces que aparece). Diremos que e es miembro o pertenece a un multiconjunto M cuando M tiene un elemento de la forma $(e, m(e))$.

Se pueden definir distintos tipos de operaciones para operar con multiconjuntos. En concreto, utilizaremos dos: la *unión aditiva* y la *unión con el máximo*.

La *unión aditiva*, $\uplus$, de dos multiconjuntos, M_1 y M_2, produce un nuevo multiconjunto que contiene todos los elementos de $M1$ y $M2$ cuyos miembros son diferentes, más un nuevo par por cada miembro que está en los dos multiconjuntos a la vez, con multiplicidad igual la suma de las multiplicidades:

$$\begin{aligned} M_1 \uplus M_2 = &\{(e, m(e)) \mid (e, m(e)) \in M_1, e \notin M_2\} \cup \\ &\{(e, m(e)) \mid (e, m(e)) \in M_2, e \notin M_1\} \cup \\ &\{(e, m_1(e) + m_2(e)) \mid (e, m_1(e)) \in M_1, \\ &\qquad (e, m_2(e)) \in M_2\} \end{aligned} \tag{5.1}$$

La *unión con el máximo*, $\underset{\max}{\cup}$, de dos multiconjuntos, se define de forma similar, pero utilizando el máximo de las multiplicidades en lugar de la suma:

$$\begin{aligned} M_1 \underset{\max}{\cup} M_2 = &\{(e, m(e)) \mid (e, m(e)) \in M_1, e \notin M_2\} \cup \\ &\{(e, m(e)) \mid (e, m(e)) \in M_2, e \notin M_1\} \cup \\ &\{(e, \text{máx}(m_1(e), m_2(e))) \mid (e, m_1(e)) \in M_1, \\ &\qquad (e, m_2(e)) \in M_2\} \end{aligned} \tag{5.2}$$

Los procedimientos básicos para calcular todas las heurísticas basadas en ignorar los efectos negativos (borrados) de las acciones utilizan información relacionada con las acciones que se necesitan para conseguir las metas del problema y las (sub)metas que se generan en un proceso hacia atrás y recursivo de regresión de metas. Esta información da lugar a las acciones cuyo coste se multiplica por un factor $N'(a)$ mayor que cero en el cálculo de la heurística, donde como se explicó en el capítulo anterior, en todos los casos la heurística se calcula como:

$$h(\mathcal{G}, s) = \sum_{a \in \mathcal{A}} N'(a) \times coste(a) \tag{5.3}$$

Los factores $N'(a)$ indican el número de veces que se suma el coste de la misma acción para generar la heurística y son mayores que cero para las acciones que se encuentran en un conjunto que, como en el capítulo anterior, denominaremos π.

Los factores $N'(a)$ dependen de las proposiciones que consigue cada acción en el proceso de regresión de metas. Por este motivo, tanto las acciones como las proposiciones que éstas consiguen son relevantes. En la definición generalizada de la siguiente sección utilizaremos multiconjuntos cuyos miembros son pares acción-proposición (a, p) (es decir, los elementos de los multiconjuntos tienen la forma $((a, p), m((a, p)))$, donde $a \in \mathcal{A}$ es una acción, $p \in \mathcal{P}$ es una proposición, y $p \in add(a)$. Estos pares se utilizarán para denotar que en el proceso de regresión de metas la acción a se utiliza para conseguir la proposición p tantas veces como indica su multiplicidad.

Por otro lado, el hecho de que unas acciones tengan un factor $N'(a)$ mayor que cero implica que otras también lo tengan: en general, aquéllas que se utilizan para conseguir sus precondiciones, aunque en algunas heurísticas no se incluyen todas. Se necesita que todas las precondiciones sean ciertas para poder aplicar una acción, al igual que se necesita que todas las metas del problema sean ciertas para considerarlo resuelto. Si contamos con n conjuntos π, con las acciones necesarias para conseguir cada una de n proposiciones, éstos se tienen que agregar de alguna manera para conseguir el conjunto de todas las acciones necesarias para conseguir todas las proposiciones a la vez. Por este motivo definiremos *funciones de agregación* que operan con multiconjuntos de pares acción-proposición.

Sea $\mathcal{A} \times \mathcal{P}$ el conjunto de pares acción-proposición en un dominio de planificación instanciado, y sea $M_{\mathcal{A}\times\mathcal{P}}$ un multiconjunto con miembros en el dominio $\mathcal{A} \times \mathcal{P}$. Las funciones de agregación son de la forma:

$$Agg : M_{\mathcal{A}\times\mathcal{P}} \times \ldots \times M_{\mathcal{A}\times\mathcal{P}} \rightarrow M_{\mathcal{A}\times\mathcal{P}}$$

Es decir, dado un número finito de multiconjuntos de pares acción-proposición, una función de agregación genera otro multiconjunto de pares acción-proposición que es el resultado de agregar sus argumentos. Cuando una función de agregación se utiliza con un único argumento el resultado es el propio argumento. Funciones clásicas de agregación son el máximo y la suma, que se utilizan en las heurísticas del máximo y aditiva respectivamente. En estos casos, la agregación se define sobre escalares en lugar de multiconjuntos.

Por otro lado, el hecho de que unas acciones pertenezcan al conjunto π y otras no, también depende del coste que supone aplicarlas (incluyendo el coste de conseguir sus precondiciones). Por lo tanto, también es necesario definir funciones que permitan generar un coste (es decir, un número) a partir de multiconjuntos de pares acción-proposición. Las denominaremos *funciones de coste*, y tienen la forma:

$$Cost : M_{\mathcal{A}\times\mathcal{P}} \rightarrow \mathbb{R}$$

Es decir, una función de coste recibe un multiconjunto de pares acción-proposición y devuelve un valor real.

5.2. Definición generalizada basada en multiconjuntos: primera aproximación

La definición declarativa generalizada que se propone en este apartado está basada en propagar multiconjuntos de pares acción-proposición. La definición obvia el proceso de extracción que se realiza en los *RPGs*. Esto se puede hacer porque la selección de acciones en el proceso de extracción se lleva a cabo escogiendo las acciones que minimizan cierta expresión. La expresión a minimizar, que es siempre la misma para la misma heurística, depende de lo que haya en el *RPG* antes de la aparición de la acción. Así, el proceso de extracción se puede eliminar si se propaga hacia delante la información necesaria para hacer la selección de acciones.

La definición que se propone está compuesta de las cinco definiciones siguientes. π representa un multiconjunto de pares acción-proposición. El subíndice, en cada caso, indica si el multiconjunto contiene la información necesaria para estimar el coste de conseguir una proposición (*prop*), el coste de conseguir un conjunto de proposiciones (*set-prop*), o el coste de aplicar una acción (*ac*) para conseguir una proposición determinada.

Definición 5.1. El multiconjunto de pares acción-proposición para estimar el coste de conseguir una proposición p a partir de un estado s, utilizando una función de agregación Agg, se define como:

$$\pi_{prop}(p; s; Agg) = \begin{cases} \emptyset & \text{if } p \in s \\ \pi_{ac}(a_p; p; s; Agg) & \text{en otro caso} \end{cases} \tag{5.4}$$

donde $\pi_{ac}(a_p; p; s; Agg)$ es el multiconjunto de pares acción-proposición para estimar el coste de conseguir p desde el estado s, pero utilizando la acción a_p para generar p, $p \in add(a_p)$. El multiconjunto π_{ac} contiene el par (a_p, p) junto con los pares acción-proposición

que permiten estimar el coste de conseguir las precondiciones de a_p partiendo del estado s. Estos pares se obtienen agregando los multiconjuntos asociados a cada precondición q ($\pi_{prop}(q; s)$), mediante la función Agg.

En general, $\pi_{ac}(a; p; s; Agg)$ es el multiconjunto que permite estimar el coste de aplicar la acción a para conseguir la proposición p, partiendo del estado s y utilizando como función de agregación Agg. Este multiconjunto por lo general se calcula como la *unión aditiva* de dos multiconjuntos: un multiconjunto que contiene únicamente el par (a, p) con multiplicidad 1, y el multiconjunto que resulta de aplicar la función Agg para agregar los multiconjuntos de cada precondición de a:

Definición 5.2. El multiconjunto de pares acción-proposición para estimar el coste de utilizar la acción a para conseguir una proposición p a partir de un estado s, utilizando la función de agregación Agg, se calcula como:

$$\pi_{ac}(a; p; s; Agg) = \{((a, p), 1)\} \uplus \text{Agg}(\{\pi_{prop}(q; s; Agg) \mid q \in pre(a)\}) \tag{5.5}$$

Ahora bien, ¿cómo se elige la acción a_p de la Definición (5.1)? Es decir ¿cómo se elige, de entre todas las posibles, la acción cuyo coste se tendrá en cuenta en la heurística para generar cada proposición? En general, se elige la acción que minimiza una determinada función de coste, como se define a continuación.

Definición 5.3. La acción a_p que genera el multiconjunto de pares acción-proposición para estimar el coste de conseguir la proposición p a partir de un estado s cumple la siguiente ecuación:

$$a_p \in \underset{a \in A(p)}{\arg\min}\ Cost_{a_p}\left(\pi_{ac}(a; p; s; Agg_{a_p})\right) \tag{5.6}$$

Lo que quiere decir que entre todas las acciones que generan p, que denotamos como $A(p)$, a_p es la que minimiza la función $Cost_{a_p}$ aplicada sobre cada uno de los multiconjuntos que permiten estimar el coste de cada acción que genera p. Usualmente a la acción a_p se la denomina *best-supporter* de la proposición p. La función de agregación que se utiliza en esta fórmula, es decir para seleccionar el *best-supporter*, es la función Agg_{a_p}.

Finalmente, el multiconjunto de pares acción-proposición para estimar el coste de un conjunto de proposiciones se define como:

Definición 5.4. El multiconjunto de pares acción-proposición para estimar el coste de conseguir un conjunto de proposiciones P partiendo del estado s y utilizando la función de agregación genérica Agg, se define como:

$$\pi_{set\text{-}prop}(P; s; Agg) = Agg(\{\pi_{prop}(p; s; Agg) \mid p \in P\}) \tag{5.7}$$

Esta es la fórmula que se utiliza para calcular el multiconjunto de pares acción-proposición que sirve para estimar el coste de conseguir todas las proposiciones que son metas del problema. Una vez calculado este multiconjunto, el valor heurístico se calcula aplicando una función de coste, $Cost_h$ sobre él:

Definición 5.5. El coste estimado de conseguir las metas $\mathcal{G}$ del problema a partir del estado s, se calcula como:

$$h(\mathcal{G}; s) = Cost_h(\pi_{set\text{-}prop}(\mathcal{G}; s; Agg_h)) \tag{5.8}$$

Nótese que la función de agregación en este caso, Agg_h, puede ser diferente de la función de agregación Agg_{a_p} que se utiliza para seleccionar las acciones a tener en cuenta.

Para instanciar estas definiciones y conseguir una heurística particular es necesario especificar cuatro parámetros funcionales, de los cuales, dos son funciones de agregación y dos son funciones de coste:

- Agg_h: Función de agregación para calcular el valor final de la heurística: $h(\mathcal{G}; s)$.
- Agg_{a_p}: Función de agregación para determinar la acción a_p que se considera (o selecciona) para generar cada proposición necesaria (según la regresión de metas).
- $Cost_{a_p}$: Función de coste a minimizar para la selección las acciones a_p.
- $Cost_h$: Función de coste para calcular la estimación heurística final.

En la siguiente sección se detallan las funciones de agregación y coste que se han venido utilizando en las distintas heurísticas.

5.3. Funciones de agregación y coste

Para conseguir la definición de la mayoría de las heurísticas basadas en ignorar los efectos negativos de las acciones, mediante una instanciación de la definición generalizada, es necesario definir tres funciones de agregación y tres funciones de coste:

5.3.1. Funciones de agregación

- Función *Agg-union+*: esta función agrega un número finito de multiconjuntos, aplicando la unión aditiva de multiconjuntos entre ellos, definida en la Ecuación (5.1):

$$\textit{Agg-union+}(\pi_1, \pi_2, \ldots, \pi_n) = \biguplus \pi_i \tag{5.9}$$

- Función *Agg-union-max*: esta función agrega un número finito de multiconjuntos, aplicando la unión con el máximo entre ellos, definida en la Ecuación (5.2):

$$\textit{Agg-union-max}(\pi_1, \pi_2, \ldots, \pi_n) = \bigcup_{\max} \pi_i \tag{5.10}$$

- Función *Agg-max*: esta función agrega un número finito de multiconjuntos, $\pi_1, \pi_2, \ldots, \pi_n$, seleccionando de entre ellos aquel multiconjunto que maximiza una determinada función de coste, que denominaremos $Cost_{Agg\text{-}max}$:

$$\textit{Agg-max}(\pi_1, \pi_2, \ldots, \pi_n) = \pi_i \tag{5.11}$$

donde $\pi_i \in \underset{\pi_j, j\in[1,\ldots,n]}{\arg\max}\ Cost_{Agg\text{-}max}(\pi_j)$

Cuando una función de agregación se instancia con *Agg-max*, es necesario instanciar también la función de coste a maximixar, $Cost_{Agg\text{-}max}$. Lo usual, como veremos después, es que esta función de coste sea la función *Cost-single*, definida en el siguiente apartado.

5.3.2. Funciones de coste

Las funciones de coste que utilizan algunas heurísticas no necesitan de toda la información que contiene el multiconjunto de pares acción-proposición al que se aplican. En algunos casos, únicamente se utiliza la información relacionada con el número de veces que se debe aplicar cada acción, sin tener en cuenta las proposiciones que ésta consigue. Esto motiva la definición previa de una función que permita compactar multiconjuntos de pares acción-proposición en multiconjuntos de acciones. Denominaremos a esta función *función de compactación* ($\mathcal{C}$).

Definición 5.6 (Función de compactación, $\mathcal{C}$)**.** La función de compactación $\mathcal{C} : M_{\mathcal{A}\times\mathcal{P}} \rightarrow M_{\mathcal{A}}$ que recibe un multiconjunto de pares acción-proposición y genera un multiconjunto de acciones a partir de él se define como:

$$\mathcal{C}(\pi)=\{(a, m(a)) \quad | \quad ((a,p), m((a,p))) \in \pi, m(a) = \sum_{p} m((a,p))\}$$

Es decir, dado un multiconjunto de pares acción-proposición π, el multiconjunto de acciones resultado de la compactación $\mathcal{C}(\pi)$ contiene un elemento por cada acción diferente en los pares (a,p) de π. La multiplicidad de cada acción a se calcula como la suma de las multiplicidades de todos los pares que contienen la acción a en el multiconjunto original.

Las funciones de coste son las siguientes:

- Función *Cost-single*: esta función calcula el coste de un multiconjunto de pares acción-proposición, π, como la suma de los costes de cada acción en el multiconjunto compactado $\mathcal{C}(\pi)$:

$$Cost\text{-}single(\pi) = \sum_{(a,m(a))\in\mathcal{C}(\pi)} cost(a) \tag{5.12}$$

- Función *Cost-multiple*: esta función calcula el coste de un multiconjunto de pares acción-proposición, π, como la suma del coste de cada acción en el conjunto compactado multiplicado por su multiplicidad:

$$Cost\text{-}multiple(\pi) = \sum_{(a,m(a))\in\mathcal{C}(\pi)} m(a) \times cost(a) \tag{5.13}$$

- Función *Cost-single-eff*: esta función calcula el coste de un multiconjunto de pares acción-proposición, como la suma del coste de cada acción a en un par (a,p), dividida

por el numero de efectos positivos de la acción ($\#add(a)$):

$$\textit{Cost-single-eff}(\pi) = \sum_{((a,p),m((a,p)))\in\pi} \frac{cost(a)}{\#add(a)} \tag{5.14}$$

La estimación heurística de un estado se calcula aplicando la función $Cost_h$ al multiconjunto $\pi_{\textit{set-prop}}(\mathcal{G}; s; Agg)$. También se pueden determinar fácilmente los factores $N'(a)$ de cada acción en el cálculo de las heurística. Estos dependen de la instanciación de la función $Cost_h$:

- Para $Cost_h = \textit{Cost-single}$

$$N'(a) = \begin{cases} 0 & \text{si } (a, m(a)) \notin \mathcal{C}(\pi_{\textit{set-prop}}(\mathcal{G}; s; Agg)) \\ 1 & \text{si } (a, m(a)) \in \mathcal{C}(\pi_{\textit{set-prop}}(\mathcal{G}; s; Agg)) \end{cases}$$

- Para $Cost_h = \textit{Cost-multiple}$

$$N'(a) = \begin{cases} 0 & \text{si } (a, m(a)) \notin \mathcal{C}(\pi_{\textit{set-prop}}(\mathcal{G}; s; Agg)) \\ m(a) & \text{si } (a, m(a)) \in \mathcal{C}(\pi_{\textit{set-prop}}(\mathcal{G}; s; Agg)) \end{cases}$$

- Para $Cost_h = \textit{Cost-single-eff}$

$$N'(a) = \begin{cases} 0 & \text{si } ((a,p), m((a,p))) \notin \pi_{\textit{set-prop}}(\mathcal{G}; s; Agg) \\ \frac{1}{\#add(a)} & \text{si } ((a,p), m((a,p))) \in \pi_{\textit{set-prop}}(\mathcal{G}; s; Agg) \end{cases}$$

5.4. Instanciaciones

Considerando los procesos básicos de cálculo de las heurísticas en cada caso, las siguientes heurísticas, cuyas definiciones matemáticas se resumen en la Tabla 4.1 (sección 4.8, página 112), son instanciaciones de la definición formulada:

- La heurística del máximo, h_{max}, para planificación clásica y para planificación basada en costes.
- La heurística aditiva, h_{add}, para planificación clásica y para planificación basada en costes.
- La heurística del máximo ampliada para planificación clásica (costes unitarios), cuya instanciación coincide con la heurística del planificador FF, si ésta se calcula sin considerar efectos secundarios en la extracción ($h_{\textit{ff-NSE}}$).
- La versión de la heurística del máximo ampliada calculada utilizando $\pi^{+}_{\textit{max-unit}}$. O lo que es lo mismo, la heurística de METRIC-FF.
- La heurística del máximo ampliada, que coincide con las heurísticas $h_{\textit{basic-sapa-max}}$ y $h_{\textit{level-max-NSE}}$.

- La heurística aditiva restringida, que coincide con las heurísticas $h_{basic\text{-}sapa\text{-}add}$, $h_{level\text{-}add\text{-}NSE}$, y h_a.

Otras heurísticas de reciente aparición que también constituyen instanciaciones son la heurística *set-additive* y la heurística h_{pmax} (ambas definidas en el estado de la cuestión, sección 2.4.4.5, página 53).

La heurística del máximo y la heurística aditiva, se calculan en su definición original propagando escalares. Sin embargo, las heurística del máximo ampliada (y sus derivadas), la heurística aditiva restringida y la heurística *set-additive* se calculan propagando conjuntos de acciones. En el capítulo 4 se ha mostrado que las heurísticas basadas en *RPGs*, se pueden calcular propagando conjuntos de acciones. Para generalizar estas heurísticas bastaría con propagar conjuntos de acciones, que es lo más general: dado un conjunto de acciones se puede obtener el escalar correspondiente si éste se calcula en función del coste de las acciones del mismo.

La heurística $h_{\max}$ tiene una definición sustancialmente distinta a las anteriores, ya que propaga vectores un elemento por proposición. Esta heurística es la que hace necesario incluir las proposiciones en los elementos a propagar.

Por lo explicado en los párrafos anteriores, los multiconjuntos de pares acción-proposición contienen información suficientemente general como para generar todas las heurísticas incluídas en la Tabla 5.1. Las funciones de agregación y coste correspondientes a cada caso se obtienen aplicando las transformaciones adecuadas para generar la información relevante a cada heurística. Por ejemplo, para la heurística $h_{\max}$:

- En las acciones, la agregación de precondiciones, es decir la información resultante de la operación que realiza $h_{\max}$ con los vectores de las precondiciones, se puede obtener haciendo la unión de los multiconjuntos de pares acción-proposición de las precondiciones. En el caso de $h_{\max}$ la multiplicidad es irrelevante, por lo que es válida tanto la únión con el máximo como la unión aditiva. En realidad para esta heurística, bastaría con propagar conjuntos de pares acción-proposición en lugar de multiconjuntos. El contenido de los vectores, que es numérico y está relacionado con con los costes de las acciones, se puede obtener a partir de las acciones que se propagan en los multiconjuntos. La Figura 5.1 muestra la equivalencia entre la propagación de vectores y la propagación de multiconjuntos. En esta figura se asume que hay una acción a_3 con precondiciones p_0 y p_1, un único efecto p_2, y que esta acción es el *best supporter* de p_2. El vector asociado a a_3 (que no aparece por motivos de claridad) es el mismo que el de p_2 excepto la tercera componente que es cero. Esto es así porque $h_{\max}$ no incluye la información de la acción en el vector de la propia acción, sino en el de sus efectos. En la figura, la expresión $c(a_i)$ indica el coste de la acción a_i.
- Para cada efecto, en $h_{\max}$, el *best supporter* es la acción para la cual la suma de todas sus componentes más el coste de la acción dividido entre el número de efectos positivos de la misma es mínimo. Lo cual es equivalente a elegir la acción cuyo multiconjunto minimiza función *Cost-single-eff*. Obsérvese que para el ejemplo de la Figura: 5.1:

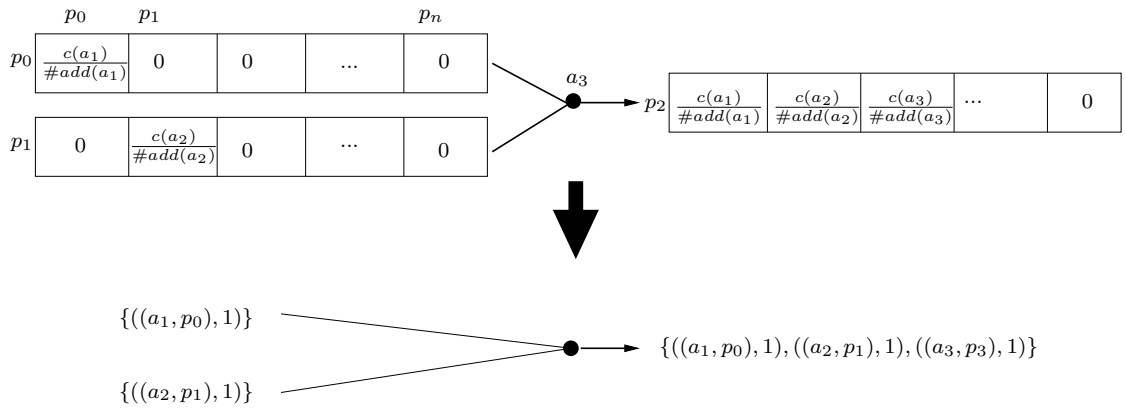

Figura 5.1: Multiconjuntos para la agregación de precondiciones en h_{pmax}.

$$\textit{Cost-single-eff}(\{((a_1, p_0), 1), ((a_2, p_1), 1), ((a_3, p_2), 1))\}) = \frac{c(a_1)}{\#add(a_1)} + \frac{c(a_2)}{\#add(a_2)} + \frac{c(a_3)}{\#add(a_3)}$$

Las instanciaciones concretas para todas las heurísticas se muestran en la Tabla 5.1. Las funciones en cuyo nombre aparece la palabra *unit* asumen costes de acciones unitarios. En todos los casos en que se aplica la función de agregación *Agg-max*, $Cost_{\textit{Agg-max}} = \textit{Cost-single}$. Cuando la función de coste que se aplica a los multiconjuntos no es una función de coste múltiple (es decir, que utilice la multiplicidad) en las uniones entre multiconjuntos, utilizar la unión aditiva es equivalente a utilizar la unión con el máximo.

La Tabla 5.1 es coherente con las conclusiones del capítulo anterior. Mientras en el capítulo anterior se demostraron las equivalencias con un análisis inductivo (de lo particular a lo general) en este caso se demuestran con un análisis deductivo (de lo general a lo particular) ya que como se observa en la tabla, las heurísticas equivalentes (las que están en la misma fila) tienen la misma instanciación. Además se pueden identificar claramente las siguientes relaciones:

- La diferencia entre las heurísticas aditiva, *set-additive*, aditiva restringida y h_{pmax} viene dada por la instanciación de las funciones de coste. De hecho, la única diferencia entre la heurística *set-additive* y la heurística aditiva restringida es que en la primera la función de coste para seleccionar los *best-supporters* es *Cost-single*, mientras que en la segunda es *Cost-multiple*. Sin embargo, h_{pmax} utiliza *Cost-single-eff*.
- Las heurísticas aditiva y aditiva restringida, siempre seleccionan los mismos *best-supporters* dado que utilizan la misma función de agregación y la misma función de coste para realizar la selección. Además, dado que las funciones para calcular Agg_h también coinciden, el multiconjunto asociado con las metas del problema $\pi_{\textit{set-prop}}(\mathcal{G}; s; Agg_h)$ siempre es el mismo para ambas heurísticas. La diferencia radica sólo en la función de coste que se utiliza para calcular el valor final de la heurística de este multiconjunto,

Heurística	Agg_h	Agg_{a_p}	$Cost_{a_p}$	$Cost_h$
máximo unit ($h_{max\text{-}unit}$)	*Agg-max*	*Agg-max*	*Cost-single-unit*	*Cost-single-unit*
máximo (h_{max})	*Agg-max*	*Agg-max*	*Cost-single*	*Cost-single*
aditiva unit ($h_{add\text{-}unit}$)	*Agg-union+*	*Agg-union+*	*Cost-multiple-unit*	*Cost-multiple-unit*
aditiva (h_{add})	*Agg-union+*	*Agg-union+*	*Cost-multiple*	*Cost-multiple*
máximo ampliada unit ($h^+_{max\text{-}unit}$) $h_{ff\text{-}NSE}$	*Agg-union+* o *Agg-union-max*	*Agg-max*	*Cost-single-unit*	*Cost-single-unit*
versión **máximo ampliada** (h^+_{max} con $\pi^+_{max\text{-}unit}$) $h_{basic\text{-}mff\text{-}NSE}$	*Agg-union+* o *Agg-union-max*	*Agg-max*	*Cost-single-unit*	*Cost-single*
máximo ampliada (h^+_{max}) $h_{basic\text{-}sapa\text{-}max}$ $h_{level\text{-}max\text{-}NSE}$	*Agg-union+* o *Agg-union-max*	*Agg-max*	*Cost-single*	*Cost-single*
aditiva restringida (h^-_{add}) $h_{basic\text{-}sapa\text{-}add}$ $h_{level\text{-}add\text{-}NSE}$ h_a	*Agg-union+* o *Agg-union-max*	*Agg-union+*	*Cost-multiple*	*Cost-single*
set-additive	*Agg-union+* o *Agg-union-max*	*Agg-union+* o *Agg-union-max*	*Cost-single*	*Cost-single*
h_{pmax}	*Agg-union+* o *Agg-union-max*	*Agg-union+* o *Agg-union-max*	*Cost-single-eff*	*Cost-single-eff*

Tabla 5.1: Algunas instanciaciones de los parámetros funcionales en la definición unificada.

es decir, en la función $Cost_h$. Lo mismo ocurre con las heurísticas del máximo con costes unitarios, la heurística de FF y la heurística de METRIC-FF. La única diferencia entre la heurística de METRIC-FF y la heurística del máximo ampliada es la función de coste que se utiliza para seleccionar los *best-supporters*: en METRIC-FF es la función *Cost-single-unit*, mientras que en la heurística del máximo ampliada es *Cost-single*.

- Todas las heurísticas, menos la heurística del máximo recolectan un conjunto de acciones en $\pi_{set\text{-}prop}(\mathcal{G}; s; Agg_h)$ que, si se ordena respetando el orden causal, constituye una solución al problema relajado. Esto es porque cuando la función de agregación Agg_h garantiza que en el resultado de la agregación no se pierde ninguna acción de los conjuntos que se agregan (esto ocurre en el caso de que $Agg_h = Agg\text{-}union+$ o $Agg_h = Agg\text{-}union\text{-}max$, pero no en el caso de $Agg_h = Agg\text{-}max$), ya que se van recolectando todas las acciones necesarias para conseguir todas las (sub)metas.

5.5. Ejemplo ilustrativo

En esta sección se muestra un ejemplo de las diferentes instanciaciones. Supongamos el problema relajado de la Figura 5.2. Las proposiciones que están conectadas a la izquierda de cada acción son sus precondiciones, mientras que las que están conectadas a la derecha son sus efectos positivos. El valor que aparece debajo de cada acción representa su coste. El estado inicial $\mathcal{I}$ sólo cuenta con la proposición p. Las metas del problema son t y u.

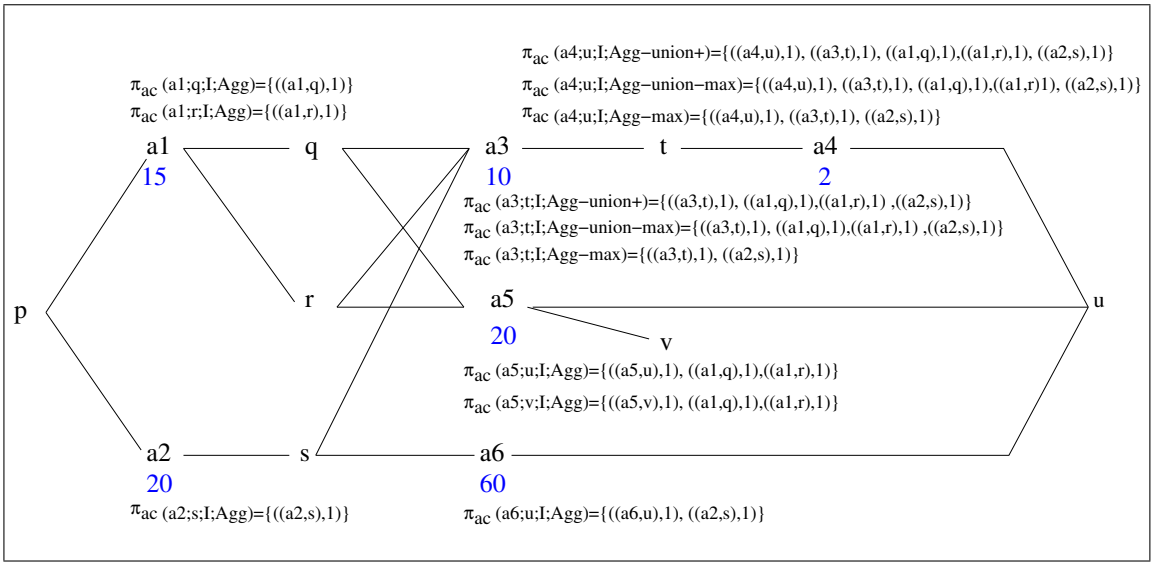

Figura 5.2: Ejemplo de las instanciaciones sobre un problema relajado.

Inicialmente, el multiconjunto $\pi_{prop}(p; \mathcal{I}; Agg)$ es el conjunto vacío, ya que p pertenece al estado $\mathcal{I}$. Los multiconjuntos para las acciones a_1 (para generar q y r) y a_2 (para generar s) son los mismos para las tres funciones de agregación que se han definido (*Agg-union+*, *Agg-union-max*, y *Agg-max*), dado que ambas acciones tienen como única precondición p. Así:

- $\pi_{ac}(a_1; q; \mathcal{I}; Agg) = \{((a_1, q), 1)\}$
- $\pi_{ac}(a_1; r; \mathcal{I}; Agg) = \{((a_1, r), 1)\}$
- $\pi_{ac}(a_2; s; \mathcal{I}; Agg) = \{((a_2, s), 1)\}$

Los multiconjuntos para las proposiciones q y r vienen dados por la acción a_1, ya que es la única acción que genera estas proposiciones. Por el mismo motivo, el multiconjunto para la proposición s viene dado por a_2. De esta manera:

- $\pi_{prop}(q; \mathcal{I}; Agg) = \{((a_1, q), 1)\}$
- $\pi_{prop}(r; \mathcal{I}; Agg) = \{((a_1, r), 1)\}$
- $\pi_{prop}(s; \mathcal{I}; Agg) = \{((a_2, s), 1)\}$

A continuación, los multiconjuntos para la acción a_5 (para conseguir u) y para la acción a_6 (también para conseguir u) son también los mismos para las tres funciones de agregación, porque a_5 tiene dos precondiciones que genera la misma acción y a_6 sólo tiene una precondición. Por lo tanto:

- $\pi_{ac}(a_5; u; \mathcal{I}; Agg) = \{((a_5, u), 1), ((a_1, q), 1), ((a_1, r), 1)\}$
- $\pi_{ac}(a_6; u; \mathcal{I}; Agg) = \{((a_6, u), 1), ((a_2, s), 1)\}$.

El multiconjunto para la acción a_3 (para conseguir t) depende de la función de agregación que se haya seleccionado:

- Para *Agg-union+* y *Agg-union-max* los multiconjuntos de las precondiciones (q, r y s) se combinan mediante la *unión aditiva* y la *unión con el máximo* respectivamente. En este caso el resultado coincide:

$$\pi_{ac}(a_3; t; \mathcal{I}; \textit{Agg-union<+/max>}) = \{((a_3, t), 1), ((a_1, q), 1), ((a_1, r), 1), ((a_2, s)1)\}$$

- Para *Agg-max* (con $Cost_{Agg\text{-}max} = Cost\text{-}single$), los multiconjuntos de las precondiciones de agregan tomando el multiconjunto que maximiza la función *Cost-single*. En este caso, como a_2 es más cara que a_1, el multiconjunto que maximiza esa función es $\{((a_2, s), 1)\}$, por lo tanto:

$$\pi_{ac}(a_3; t; \mathcal{I}; \textit{Agg-max}) = \{((a_3, t), 1), ((a_2, s), 1)\}$$

Dado que la única acción que genera t es a_3, el multiconjunto para t es el de a_3.

Para a_4:

- $\pi_{ac}(a_4; u; \mathcal{I}; \textit{Agg-union<+/max>}) = \{((a_4, u), 1), ((a_3, t), 1), ((a_1, q), 1), ((a_1, r), 1), ((a_2, s), 1)\}$
- $\pi_{ac}(a_4; u; \mathcal{I}; \textit{Agg-max}) = \{((a_4, u), 1), ((a_3, t), 1), ((a_2, s), 1)\}$

La proposición u se puede generar utilizando las acciones a_4, a_5 ó a_6. El hecho de que el multiconjunto asociado con u sea el de una u otra acción depende de las funciones implicadas en la selección de la acción: $Cost_{a_p}$ y Agg_{a_p}. El valor de $Cost_{a_p}$ para cada acción se muestra en la Tabla 5.2, para las tres posibles funciones Agg_{a_p} (en esta tabla *Agg-union* se refiere a ambas, la *unión aditiva* y la *unión con el máximo*, dado que para los dos casos el multiconjunto es el mismo). El valor en negrita es el mínimo (y determina la acción que se selecciona). Por ejemplo, las heurísticas que utilizan $Cost_{a_p}$ = *Cost-single* y Agg_{a_p} = *Agg-union+*, seleccionan la acción a_5 para conseguir u, porque el mínimo en este caso es 35, en la segunda fila.

Multiconjunto	*Cost-single*	*Cost-single -unit*	*Cost-multiple*	*Cost-multiple -unit*	*Cost-single -eff*
$\pi_{ac}(a_4;\mathcal{I};Agg<union/max>)$	47/**32**	4/3	62/**32**	5/3	47/
$\pi_{ac}(a_5;\mathcal{I};Agg<union/max>)$	**35**/35	**2/2**	**50**/50	3/3	**25/**
$\pi_{ac}(a_6;\mathcal{I};Agg<union/max>)$	80/80	**2/2**	80/80	**2/2**	80/

Tabla 5.2: Valores de las distintas funciones de coste para los multiconjuntos de las acciones que generan u.

La Tabla 5.3 muestra para cada una de las heurísticas:

- La acción que se elige para generar u, es decir, el *best-supporter* de u (a_u en la tabla se corresponde con a_p de la Definición (5.3)).
- Los multiconjuntos π_{prop} para cada meta del problema u y t.
- El multiconjunto para el conjunto de todas metas del problema: $\pi_{set\text{-}prop}(\mathcal{G};\mathcal{I};Agg)$, y
- El valor final de la estimación heurística.

En esta tabla, todos los multiconjuntos se muestran en su forma compactada, por razones de claridad. Los únicos que no se muestran compactados son los correspondientes a h_{pmax}, porque esta es la única heurística cuyas funciones de coste utilizan directamente los multiconjuntos de pares acción-proposición.

Por ejemplo, la heurística del *máximo ampliada*, utiliza la función de agregación *Agg-max* con la función de coste *Cost-single* para seleccionar acciones. En concreto éstas son las funciones que se utilizan para seleccionar a_u. Por lo tanto, la acción que se selecciona es a_4 (ver Tabla 5.2). Después, para calcular $\pi_{prop}(u,\mathcal{I},Agg)$ y $\pi_{prop}(t,\mathcal{I},Agg)$, utiliza *Agg=Agg-union+*. Así, el multiconjunto de u es el multiconjunto de a_4 con *Agg-union+* (ver Figura 5.2). De la misma forma, el multiconjunto para t es el multiconjunto de a_3 con *Agg-union+*. Después estos dos multiconjuntos (el de u y el de t) se agregan según la definición de $\pi_{set\text{-}prop}(\mathcal{G};\mathcal{I};Agg)$, donde Agg = *Agg-union+*. Finalmente, la estimación

Heurística	a_u	$\pi_{prop}(u, \mathcal{I}, Agg)$	$\pi_{prop}(t, \mathcal{I}, Agg)$	$\pi_{set\text{-}prop}(\mathcal{G}, \mathcal{I}, Agg)$	$h(\mathcal{G}, \mathcal{I})$
máximo unit ($h_{max\text{-}unit}$)	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_2,1)\}$	$\{(a_5,1),(a_1,2)\}$	2
	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_2,1)\}$	$\{(a_3,1),(a_2,1)\}$	2
	a_6	$\{(a_6,1),(a_2,1)\}$	$\{(a_3,1),(a_2,1)\}$	$\{(a_6,1),(a_2,1)\}$	2
	a_6	$\{(a_6,1),(a_2,1)\}$	$\{(a_3,1),(a_2,1)\}$	$\{(a_3,1),(a_2,1)\}$	2
máximo (h_{max})	a_4	$\{(a_4,1),(a_3,1),(a_2,1)\}$	$\{(a_3,1),(a_2,1)\}$	$\{(a_4,1),(a_3,1),\}(a_2,1)\}$	32
aditiva unit ($h_{add\text{-}unit}$)	a_6	$\{(a_6,1),(a_2,1)\}$	$\{(a_3,1),(a_1,2),(a_2,1)\}$	$\{(a_6,1),(a_2,2),(a_3,1),(a_1,2)\}$	6
aditiva (h_{add})	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_1,2),(a_2,1)\}$	$\{(a_5,1),(a_1,4),(a_3,1),(a_2,1)\}$	110
máximo ampliada unit ($h^+_{max\text{-}unit}$)	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_1,2),$q$(a_2,1)\}$	$\{(a_5,1),(a_1,4),(a_3,1),(a_2,1)\}$	4
$h_{ff\text{-}NSE}$	a_6	$\{(a_6,1),(a_2,1)\}$	$\{(a_3,1),(a_1,2),(a_2,1)\}$	$\{(a_6,1),(a_2,2),(a_3,1),(a_1,2)\}$	4
versión **máximo ampliada** (h^+_{max} con $\pi^+_{max\text{-}unit}$)	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_1,2),(a_2,1)\}$	$\{(a_5,1),(a_1,4),(a_3,1),(a_2,1)\}$	65
$h_{basic\text{-}mff\text{-}NSE}$	a_6	$\{(a_6,1),(a_2,1)\}$	$\{(a_3,1),(a_1,2),(a_2,1)\}$	$\{(a_6,1),(a_2,2),(a_3,1),(a_1,2)\}$	105
máximo ampliada (h^+_{max}) $h_{basic\text{-}sapa\text{-}max}$ $h_{level\text{-}max\text{-}NSE}$	a_4	$\{(a_4,1),(a_3,1),(a_1,2),(a_2,1))\}$	$\{(a_3,1),(a_1,2),(a_2,1)\}$	$\{(a_4,1),(a_3,2),(a_1,4),(a_2,2)\}$	47
aditiva restringida $h_{basic\text{-}sapa\text{-}add}$ $h_{level\text{-}add\text{-}NSE}$ h_a	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_1,2),,(a_2,1)\}$	$\{(a_5,1),(a_1,4),(a_3,1),(a_2,1)\}$	65
set-additive	a_5	$\{(a_5,1),(a_1,2)\}$	$\{(a_3,1),(a_1,2),,(a_2,1)\}$	$\{(a_5,1),(a_1,4),(a_3,1),(a_2,1)\}$	65
h_{pmax}	a_5	$\{((a_5,u),1),((a_1,q),1)),((a_1,r),1)\}$	$\{((a_3,t),1),((a_1,q),1)),((a_1,r),1),((a_2,s),1)\}$	$\{((a_5,u),1),((a_1,q),1),((a_1,r),1),((a_3,t),1),((a_2,s),1)\}$	55

Tabla 5.3: Valores heurísticos para las distintas instanciaciones.

heurística se calcula aplicando la función $Cost_h = \textit{Cost-single}$ al resultado de la agregación. La Tabla 5.3 muestra todas las posibilidades para los casos en los que se producen empates. Para el ejemplo presentado, las heurísticas más precisas son las que se encuentran en la fila de la heurística del *máximo ampliada*, que proporcionan el coste óptimo de conseguir las metas (47).

5.6. Definición generalizada: segunda aproximación

La generalización conseguida hasta este punto no es capaz de generar la familia de heurísticas h^m (Haslum and Geffner, 2000). Sin embargo, se puede generalizar aún un poco más de manera que las heurísticas de esta familia también constituyan una instanciación.

Para conseguir esto, se debe formular la definición permitiendo conjuntos de proposiciones de tamaño máximo m dónde anteriormente había exclusivamente una única proposición. Así tendríamos, que el multiconjunto $\pi_{prop}(p; s; Agg)$, definido para la proposición p, ahora estaría definido para un conjunto de proposiciones P de tamaño máximo m. Esto conlleva unificar las definiciones de π_{prop} y de $\pi_{set\text{-}prop}$, ya que ahora ambos multiconjuntos se refieren a un conjunto de proposiciones. Esto se puede hacer sin pérdida de generalidad puesto que una proposición es en realidad un conjunto de proposiciones de tamaño uno.

En la definición resultante de $\pi_{set\text{-}prop}$ habrá que distinguir entre conjuntos de proposiciones que tienen tamaño menor o igual que m y conjuntos de proposiciones que tienen tamaño mayor que m. En este último caso, el conjunto de proposiciones se particiona en todos los posibles subconjuntos de tamaño m. Después, los multiconjuntos asociados se agregan mediante la función de agregación Agg. Así:

Definición 5.7. El multiconjunto de pares acción-proposición para estimar el coste de conseguir un conjunto de proposiciones P a partir de un estado s, utilizando la función de agregación Agg, se define como:

$$\pi_{set\text{-}prop}(P; s; Agg) = \begin{cases} \emptyset & \text{si } P \in s \\ \pi_{ac}(a_P; P; s; Agg) & \text{si } P \notin s, |P| \leq m \\ Agg(\{\pi_{set\text{-}prop}(P_m; s; Agg)\}) & \text{si } P \notin s, |P| > m \end{cases} \tag{5.15}$$

donde P_m representa un subconjunto de P de tamaño m: $P_m \subseteq P$ y $|P_m| = m$. La función Agg sirve para agregar todos los $\pi_{set\text{-}prop}$ de todos los subconjuntos de tamaño m.

Por otro lado, también en la definición de $\pi_{ac}(a; p; s; Agg)$, en el parámetro relativo a la proposición p, ahora tendremos un conjunto de proposiciones P de tamaño máximo m. Además, la acción a añadía la proposición p, pero ahora se puede dar el caso de que esa acción añada varias proposiciones del conjunto P. Las proposiciones que añade son exactamente $add(a) \cap P$. Por otro lado en $\pi_{ac}(a; p; s; Agg)$ la función Agg se utilizaba para agregar los multiconjuntos de las precondiciones de p. Sin embargo ahora no es necesario realizar ninguna agregación en este punto, sino que basta con calcular el multiconjunto asociado al conjunto de proposiciones que se obtiene haciendo regresión de átomos sobre el conjunto P a través de la acción a: $Reg(P, a) = P \setminus add(a) \cup pre(a)$. Asumiremos que cuando P tiene una única proposición, $Reg(P, a) = pre(a)$. Así:

Definición 5.8. El multiconjunto de pares acción-proposición para estimar el coste de utilizar la acción a para conseguir al menos una proposición del conjunto P, a partir de un estado s, utilizando la función de agregación Agg, se calcula como:

$$\pi_{ac}(a; P; s; Agg) = \left\{ \bigcup_{p \in add(a) \cap P} \{((a, p), 1)\} \right\} \biguplus \pi_{set\text{-}prop}(Reg(P, a); s; Agg) \tag{5.16}$$

Respecto a la acción a_P que se selecciona para calcular $\pi_{set\text{-}prop}$ es, al igual que antes, la que minimiza una determinada función de coste. Sin embargo, mientras que antes la acción se seleccionaba de entre todas las que añadían p, $A(p)$, ahora se selecciona de entre las que añaden alguna proposición de P sin borrar ninguna otra. Denominaremos a este conjunto

de acciones $A(P)$. Así:

Definición 5.9. La acción a_P que genera el multiconjunto de pares acción-proposición para estimar el coste de conseguir un conjunto de proposiciones P de tamaño menor o igual que m, a partir de un estado s, cumple la siguiente ecuación:

$$a_p \in \arg\min_{a \in A(P)} Cost_{a_p}\left(\pi_{ac}(a; P; s; Agg_{a_p})\right) \tag{5.17}$$

donde, $A(P) = \{a \in \mathcal{A} \mid add(a) \cap P \neq \emptyset \quad \text{y} \quad del(a) \cap P = \emptyset\}$

Por último,

Definición 5.10. El coste estimado de conseguir las metas del problema $\mathcal{G}$ a partir de un estado s es:

$$h(\mathcal{G}, s) = Cost_h(\pi_{set\text{-}prop}(\mathcal{G}; s; Agg_h)) \tag{5.18}$$

Todas las heurísticas de la sección anterior siguen siendo instanciaciones de esta nueva definición unificada. Para todas ellas $m = 1$. La familia de heurísticas h^m se corresponde con la instanciación que muestra la Tabla 5.4, que para $m = 1$ coincide con la heurística del máximo.

Heurística	Agg_h	Agg_{a_p}	$Cost_{a_p}$	$Cost_h$
familia h^m	*Agg-max*	*Agg-max*	*Cost-single*	*Cost-single*

Tabla 5.4: Instanciación para h^m.

5.7. Propiedades de las funciones de agregación y coste

Las funciones de agregación y coste definidas cumplen las propiedades que se expresan a continuación.

Propiedad 5.1. La operación de agregación mediante una unión con el máximo de n multiconjuntos contiene las mismas acciones, con multiplicidad menor o igual que la operación de agregación con unión aditiva de los mismos multiconjuntos.

$\forall a, a \in \textit{Agg-union+}(\pi_1, \pi_2, \ldots, \pi_n) \Leftrightarrow a \in \textit{Agg-union-max}(\pi_1, \pi_2, \ldots, \pi_n)$, y

$m_{\textit{Agg-union-max}}(a) \leq m_{\textit{Agg-union+}}(a)$

Esta propiedad implica que se cumplen las siguientes ecuaciones relacionadas con las funciones de coste:

- *Cost-single*(*Agg-union-max*$(\pi_1, \pi_2, \ldots, \pi_n)$) = *Cost-single*(*Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$)
- *Cost-single-eff*(*Agg-union-max*$(\pi_1, \pi_2, \ldots, \pi_n)$) = *Cost-single-eff*(*Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$)

- *Cost-multiple*(*Agg-union-max*$(\pi_1, \pi_2, \ldots, \pi_n)) \leq$ *Cost-multiple*(*Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$)

Propiedad 5.2. La operación de agregación con el máximo de n multiconjuntos contiene un subconjunto de las acciones que se encuentran en el resultado de la agregación con unión con el máximo de los mismos multiconjuntos, sin restricciones sobre la multiplicidad:

$\forall a, a \in$ *Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n) \Rightarrow a \in$ *Agg-union-max*$(\pi_1, \pi_2, \ldots, \pi_n)$

Con lo cual se cumple que:

- *Cost-single*(*Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n)) \leq$ *Cost-single*(*Agg-union-max*$(\pi_1, \pi_2, \ldots, \pi_n)$)
- *Cost-single-eff*(*Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n)) \leq$ *Cost-single-eff*(*Agg-union-max*$(\pi_1, \pi_2, \ldots, \pi_n)$)

Propiedad 5.3. La operación de agregación con el máximo de n multiconjuntos contiene un subconjunto de las acciones con multiplicidad menor o igual que la operación de agregación con unión aditiva de los mismos multiconjuntos.

$\forall a, a \in$ *Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n) \Rightarrow a \in$ *Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$ y

$\forall a, a \in$ *Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n), m_{Agg\text{-}max}(a) \leq m_{Agg\text{-}union+}(a)$

Con lo cual se cumple que:

- *Cost-single*(*Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n)) \leq$ *Cost-single*(*Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$)
- *Cost-single-eff*(*Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n)) \leq$ *Cost-single-eff*(*Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$)
- *Cost-multiple*(*Agg-max*$(\pi_1, \pi_2, \ldots, \pi_n)) \leq$ *Cost-multiple*(*Agg-union+*$(\pi_1, \pi_2, \ldots, \pi_n)$)

Propiedad 5.4. Dado un multiconjunto π, *Cost-single*$(\pi) \leq$ *Cost-multiple*(π)

Propiedad 5.5. Dado un multiconjunto π, *Cost-single-eff*$(\pi) \leq$ *Cost-multiple*(π)

La propiedad (5.4) garantiza que la heurística aditiva restringida siempre proporciona valores menores o iguales que la heurística aditiva. De la misma forma, la propiedad (5.3) garantiza que la heurística del máximo ampliada siempre proporcione valores mayores o iguales que la heurística del máximo.

Análogamente,

Propiedad 5.6. Si *Cost-single*(*Agg-max*$(\pi_{A1} \ldots \pi_{An})) \leq$ *Cost-single*(*Agg-max*$(\pi_{B1} \ldots \pi_{Bm})$) entonces:

- *Cost-single*(*Agg-max*$(\pi_{A1} \ldots \pi_{An})) \leq$ *Cost-single*(*Agg-union+*$(\pi_{B1} \ldots \pi_{Bm})$)
- *Cost-single*(*Agg-max*$(\pi_{A1} \ldots \pi_{An})) \leq$ *Cost-single*(*Agg-union-max*$(\pi_{B1} \ldots \pi_{Bm})$)

5.8. Algunos trabajos relacionados

En la literatura existen algunos esfuerzos previos orientados a generalizar las heurísticas que se utilizan en planificación. Por ejemplo el trabajo de J. Rintanen (Rintanen, 2006)

presenta una definición unificada de las heurísticas para planificación clásica. El autor generaliza la heurística del máximo, la heurística aditiva y la heurística de FF (ignorando también los efectos secundarios) para operadores que tienen efectos condicionales y precondiciones disyuntivas. La generalización utiliza lógica proposicional. El trabajo difiere el presentado en este capítulo en varios aspectos. En la generalización propuesta se utilizan ecuaciones matemáticas recursivas, en la línea de las definiciones originales de las heurísticas del máximo, aditiva y *set-additive*. Por otro lado, también se incluyen las heurísticas para planificación basada en costes. El tratamiento de efectos condicionales y precondiciones disyuntivas en nuestro caso es el clásico: compilar el problema en un problema de planificación clásica. En el trabajo de J. Rintanen se argumenta que esta compilación puede incrementar exponencialmente el tamaño del problema de planificación, de manera que la generalización está orientada a solventar este problema. Una conclusión importante del trabajo de J. Rintanen es la relación entre la heurística de FF y la heurística del máximo, y que la heurística de FF se puede definir sin utilizar el concepto de *grafo de planificación relajado*. Ésta es también una de las conclusiones que se obtienen de la generalización presentada en este capítulo.

Otro trabajo relacionado es el de V. Mirkis y C. Domshlak (Mirkis and Domshlak, 2007), en el cual se propone una definición generalizada para hacer propagación de costes en grafos de planificación. En este caso, los grafos de planificación se asimilan a *WAODAGs*[1]. El proceso de propagación de costes se realiza sobre los nodos y arcos de estos grafos. Al igual que en la definición de este capítulo, la definición de V. Mirkis y C. Domshlak utiliza un modelo matemático descrito en un lenguaje algebraico. Sin embargo el resultado es una definición más general que la que aquí se propone, dado que permite propagar elementos en cualquier dominio (por ejemplo, escalares, vectores, conjuntos, etc.). Esto significa su definición seguramente cubre un rango mayor de heurísticas. Sin embargo, es importante obtener un grado de generalidad adecuado al objetivo que se persigue, y aunque su definición puede ser muy útil para generar nuevas heurísticas, lo es menos para detectar las relaciones entre ellas por ser demasiado general. En el extremo, la misma heurística se puede calcular propagando elementos en dominios distintos. En el trabajo de esta tesis, en la que uno de sus objetivos es establecer las relaciones que existen entre las heurísticas, se ha unificado el dominio de los elementos que se propagan a multiconjuntos de pares acción-proposición. Estos multiconjuntos tienen la suficiente información como para poder generar a partir de ellos la mayoría de las heurísticas y la comparación se realiza a través de funciones generales que operan sobre ellos.

Otro trabajo reciente que también establece relaciones entre heurísticas existentes es el de H. Geffner y M. Helmert (Geffner, 2007; Helmert and Geffner, 2008). En este caso se relaciona la heurística aditiva con la heurística del grafo causal. Esta última es una de las pocas heurísticas que tiene en cuenta interacciones negativas entre (sub)metas, aunque también las heurísticas de la familia h^m y las basadas en PDB's tienen en cuenta algunas de estas interacciones. El resultado es una heurística que unifica ambas denominada heurística *context-enhanced additive*, cuya definición es básicamente la de la heurística aditiva, traducida al lenguaje SAS$^+$ (explicado en el estado de la cuestión, sección 2.3.2.4, página 21) y ampliada para considerar información de contexto. La información de contexto se obtiene

[1] Weighted And-Or Graphs.

evaluando el coste de las precondiciones en estados distintos al evaluado.

5.9. Resumen

En este capítulo se ha introducido una definición declarativa general para heurísticas en planificación basada en costes. En esta definición se utilizan pares acción-proposición que se propagan a través de las acciones (de las precondiciones a los efectos) utilizando funciones de agregación y coste. Muchas de las heurísticas existentes son instanciaciones de la definición generalizada, incluyendo aquéllas que se han venido definiendo en función de grafos de planificación relajados.

La definición generalizada se ha instanciado para obtener las distintas heurísticas particulares. La instanciación determina las funciones que las heurísticas comparten y las diferencias entre ellas. Cabe destacar que la definición unificada permite comparar las heurísticas desde un punto de vista teórico. Desde el punto de vista computacional, propagar multiconjuntos de pares acción-proposición no es la forma más eficiente de calcular heurísticas, puesto que algunas heurísticas sólo necesitan propagar escalares.

Desde el punto de vista experimental, puede ser importante fijar la política que se utiliza para resolver empates en la selección de los *best-supporters* ya que en otro caso, si estos se resuelven de manera arbitraria, los experimentos estarán influenciados por esta arbitrariedad.

Finalmente, la generalización propuesta se puede utilizar para generar heurísticas nuevas, bien combinando de otra forma las funciones de agregación y coste definidas, o bien definiendo funciones nuevas.

Capítulo 6

Heurísticas no numéricas

En este capítulo se estudian dos tipos de heurísticas no numéricas, como son las heurísticas de poda/ordenación y las heurísticas *lookahead* desde el punto de vista de su combinación con las heurísticas que expanden el RPG en niveles incrementales de coste: las heurísticas h_{level}.

6.1. Heurísticas de poda/ordenación

En este apartado se analizan dos tipos de heurísticas que permiten podar y/o ordenar los sucesores de un nodo en función de características de la acción que lo genera. La primera de ellas consiste en el uso de acciones *helpful* y la segunda, en ordenar las acciones según su coste.

6.1.1. Acciones *helpful*

La heurística de poda/ordenación más conocida y que se ha aplicado con más éxito en planificación clásica es la heurística de las *helpful actions* (Hoffmann and Nebel, 2001). Esta heurística, en su definición original, está expresada en función de conceptos que dependen de la construcción del $RPGs$ y de la extracción de una solución del mismo. Así, considerando una expansión clásica, dado un estado s, las *helpful actions* ($H(s)$) se definen como aquellas acciones del dominio que consiguen alguna de las (sub)metas que se producen en la extracción de una solución en el RPG en el nivel uno del RPG:

$$H(s) = \{a \in A \mid \mathit{eff}^+(a) \cap G_1 \neq \emptyset\}$$

donde G_1 son las (sub)metas que se encuentran en P_1.

Esta definición es equivalente a seleccionar *las acciones aplicables en s que generan al menos una proposición que es precondición de alguna otra acción del plan relajado (no aplicable en el estado evaluado)*. Sin embargo, para los algoritmos que construyen el grafo de planificación en niveles incrementales de coste no se produce esta equivalencia. Es necesario adaptar la idea de las *helpful actions* a este tipo de algoritmos. Para ello existen varias

posibilidades, que se explican a través del siguiente ejemplo.

La Figura 6.1 muestra un ejemplo de un problema en el dominio del satélite. En este caso hay un satélite apuntando a la dirección del `Fenómeno 1`, y el objetivo es tomar una imagen del `Fenómeno 3`. Para cumplir el objetivo el satélite debe encender un instrumento con la acción *(switch-on instrumento satélite)*; después debe calibrar el instrumento mediante la acción *(calibrate instrumento satélite)*. La calibración de un instrumento sólo se puede realizar cuando el satélite apunta hacia una determinada dirección y si el instrumento está encendido. Supongamos que en este ejemplo la dirección de calibración es la de la `Estación 1`. A continuación, el satélite debe girar para tomar la imagen. Todos los giros del satélite se realizan mediante la acción *(turn-to satélite dir-destino dir-origen)*. El satélite puede girar desde la dirección a la que apunta a cualquier otra. En la figura, los valores de los arcos indican el coste del giro. Las demás acciones tienen coste cero.

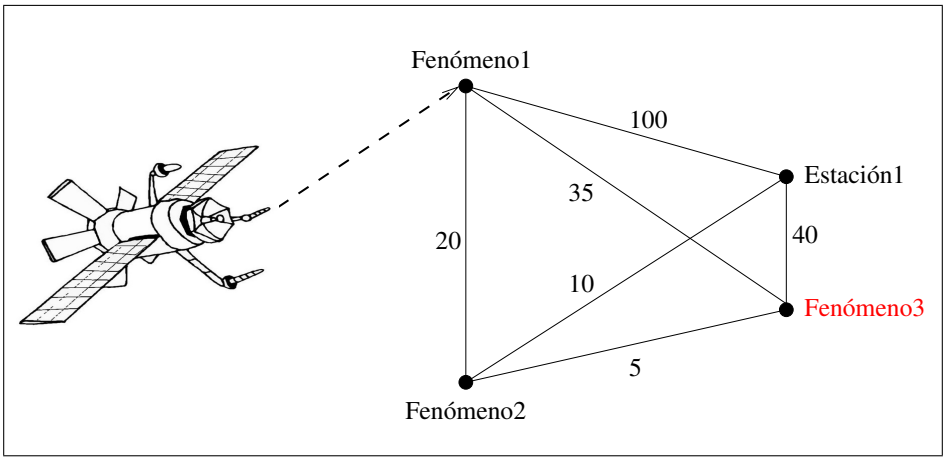

Figura 6.1: Ejemplo de un problema en el dominio del satélite.

La Figura 6.2 muestra la parte del *RPG* implicada en la extracción de la solución. En este figura, las (sub)metas están representadas con puntos. Las (sub)metas de nivel 1 (G_1) se han marcado con un círculo. Las *helpful actions* son las acciones del dominio aplicables que consiguen estas (sub)metas. En este caso todas estas acciones se encuentran en el grafo y son las marcadas en rojo.

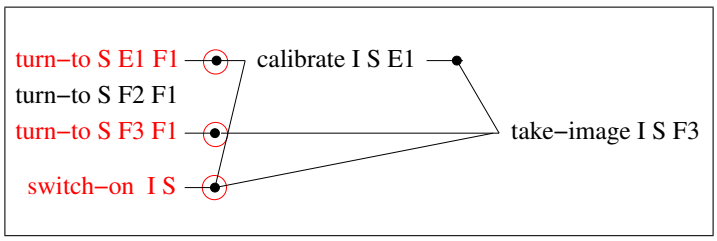

Figura 6.2: Esquema de proceso de extracción del *RPG* para costes unitarios (clásico).

Según la heurísticas h_{level} para planificación basada en costes, el *RPG* está mucho más expandido, ya que se construye en niveles incrementales de coste. La parte del *RPG* implicada en la extracción para la heurística $h_{level\text{-}max}$ se muestra en la Figura 6.3. Esto implica que la única (sub)meta de nivel 1 es tener el instrumento encendido, con lo cual siguiendo la definición original, la acción *(switch-on I S)* sería la única acción *helpful*.

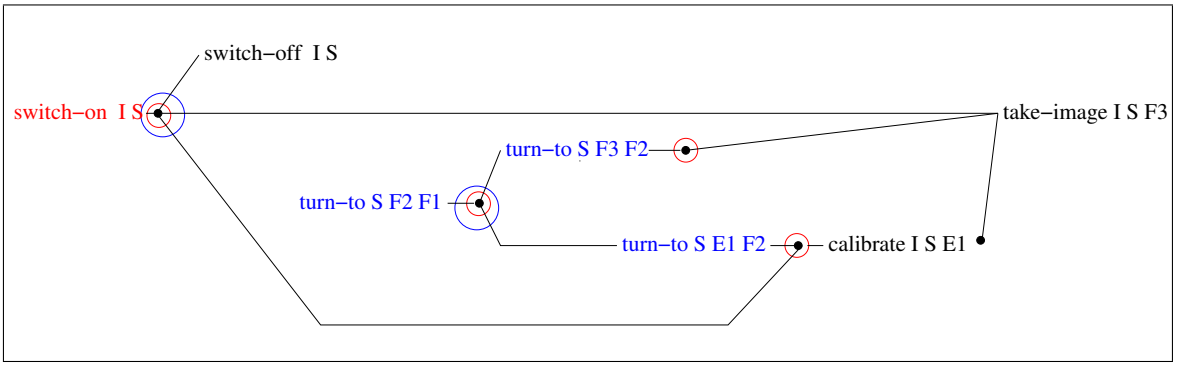

Figura 6.3: Esquema de proceso de extracción del *RPG* para $h_{level\text{-}max}$.

Aunque considerar la definición original es una opción, el número de acciones *helpful* puede disminuir considerablemente, de manera que se limitan demasiado los sucesores de cada nodo. Si esto se combina con un algoritmo de búsqueda local sobre los sucesores *helpful* (es decir, que poda en cada nodo todos aquellos sucesores que no lo son), en algunos dominios se puede llegar a perder la solución. Sin embargo, existen al menos dos posibilidades más, que denominaremos HA1 y HA2:

HA1) Considerar la definición equivalente en planificación clásica, es decir, seleccionar como *helpful actions* las acciones aplicables en el estado evaluado, que generan al menos una proposición ((sub)meta) que es precondición de alguna otra acción del plan relajado no aplicable en el estado evaluado. En el ejemplo esto supondría elegir las acciones capaces de generar los hechos marcados con un círculo pequeño. De esta forma se seleccionarían las acciones *(switch-on I S)*, *(turn-to F2 F1)*, *(turn-to E1 F1)*, porque genera el hecho *(pointing S E1)*, que en el grafo se consigue con la acción *(turn-to E1 F2)*, y *(turn-to F3 F1)*, porque genera el hecho *(pointing S F3)*, que en el grafo se consigue con *(turn-to F3 F2)*. Con esta opción, en el ejemplo se consideran las mismas acciones que en planificación clásica.

HA2) Considerar como *helpful actions* las acciones aplicables en el estado evaluado, que generan al menos una (sub)meta generada por una acción **aplicable** seleccionada en el proceso de extracción. En el ejemplo, esto supondría escoger las acciones aplicables que consiguen los hechos marcados con un círculo grande: *(switch-on I S)* y *(turn-to F2 F1)*. Obsérvese que en el caso 1) anterior no es necesario que la (sub)meta esté generada por una acción aplicable.

Para implementar la opción HA1 en el algoritmo de expansión del *RPG* de h_{level} (Figura 3.4, página 65) basta con apuntar, a medida que se construye el *RPG* aquellos hechos que

sería posible conseguir a nivel uno. Estos hechos son todos aquéllos que añaden las acciones que entran en la lista *AbiertaApp* en la primera iteración ($i = 0$). Después, las (sub)metas que se pueden alcanzar a nivel uno son las que dan lugar a las *helpful actions*. Las *helpful actions* son las acciones aplicables del dominio que generan estas (sub)metas.

Para implementar la opción HA2 basta con marcar, a medida que se construye el *RPG*, las acciones aplicables a nivel cero. Después, las (sub)metas para las cuales se selecciona en el proceso de extracción una acción aplicable a nivel cero son las que dan lugar a las *helpful actions*. Igual que antes, las *helpful actions* son las acciones aplicables del dominio que generan estas (sub)metas.

La opción HA2 es más restrictiva que la opción HA1, es decir, da lugar a menos acciones *helpful*. Sin embargo, la opción HA1 puede incluir acciones de coste alto, como es el caso de la acción *(turn-to E1 F1)* de coste 100 en el ejemplo anterior. Esto puede influir negativamente en la calidad del primer plan encontrado.

El algoritmo de la Figura 6.4 muestra lo que habría que incluir en el algoritmo de extracción en cada caso para recopilar las (sub)metas de cada capa que generan las acciones *helpful*. El algoritmo supone que se conocen las acciones que son aplicables en el nivel cero y las proposiciones que se pueden conseguir a nivel 1. Las acciones aplicables en el nivel cero son todas aquéllas que se incluyen en la lista *AbiertaApp* en la primera iteración ($i = 0$), aparezcan o no en la primera capa de acciones, A_0, del *RPG* (algunas acciones aplicables pueden no aparecer en A_0 por motivo de su coste). Las proposiciones que se pueden conseguir en nivel 1 son todas las proposiciones que generan las acciones que entran en *AbiertaApp* en la primera iteración, aparezcan o no en la segunda capa de proposiciones, P_1, del *RPG*.

6.1.2. Ordenación de las acciones considerando su coste

El coste de las acciones que generan los sucesores de un nodo se puede considerar para realizar ordenaciones heurísticas de los mismos. En los esquemas que se basan en algoritmos de búsqueda local (tipo *Hill-Climbing)*, normalmente no se considera este coste. Por el contrario los esquemas de búsqueda global (tipo *Best-First*), que utilizan una función de evaluación $f(n) = g(n) + h(n)$, sí lo consideran puesto que está incluído en $g(n)$.

Normalmente en los algoritmos de búsqueda local (sin memoria), se cuenta con un único nodo en cada iteración (el nodo raíz actual). Por lo tanto, la causa de que distintos sucesores tengan un valor distinto para $g(n)$ es coste de la acción que los genera. Así, este coste permite discriminar entre ellos. A la hora de considerar el coste de las acciones en algoritmos de búsqueda local, pueden tener sentido varias opciones:

- Ordenar los sucesores del nodo raíz según el coste de la acción que los genera. En este caso no se requiere evaluación numérica ($h(n)$) de los mismos. Este criterio tiene sentido por ejemplo en *EHC*, ya que con este algoritmo no es necesario evaluar todos los sucesores.
- Ordenar los sucesores según el valor $h(n) + coste(a)$, donde a es la acción que genera el nodo n.

```
function extraccion_RP(G,RPG)
for i := 1, ..., final_layer do
    G_i := {g ∈ G | level(g) = i}
    submetas_helpful_i := ∅
end for
for i := final_layer, ..., 1 do
    forall g ∈ G_i do
        seleccionar una acción a con g ∈ add(a) y level(a) = i − 1,
        /* opción HA1 */
        if nivel_alcanzable(g) = 1 then
            submetas_helpful_i = submetas_helpful_i ∪ {g}
        endif
        /* opción HA2 */
        if nivel_aplicable(a) = 0 then
            submetas_helpful_i = submetas_helpful_i ∪ {g}
        endif
        forall f ∈ pre(a) do
            G_nivel(f) ⋃ = {f}
        end for
    end for
end for
```

Figura 6.4: Cálculo de las (sub)metas que generan las acciones *helpful* en el proceso de extracción.

La aplicación práctica de ambos criterios se evaluará en la siguiente parte de la tesis.

6.2. Heurísticas *lookahead*: plan relajado

Las heurísticas que generan un plan relajado sensible a costes utilizan un plan relajado que se puede utilizar para generar un estado *lookahead* que no suponga empeorar demasiado la calidad de los planes. A cambio, el algoritmo de búsqueda que lo utilice será más escalable.

Por otro lado, tanto en planificación clásica como en planificación basada en costes, la obtención de estados *lookahead* a partir de planes relajados presenta problemas comunes.

Dado un plan relajado se generará un estado *lookahead* distinto dependiendo de la estrategia que se utilice para generarlo. Aunque el plan sea secuencial, en realidad el orden entre algunas de sus acciones es parcial. Todas las ordenaciones secuenciales para las que se cumplan las restricciones de causalidad por las que las acciones fueron seleccionadas (es decir, siempre que las precondiciones de cada acción se consigan antes de la aparición de la acción) dan lugar a un plan relajado distinto que posiblemente generará un estado *lookahead* también diferente.

Además, como efecto directo de ignorar los borrados, las acciones de un plan relajado presentan conflictos, por lo que en general el plan relajado no es aplicable. Los conflictos se producen porque el proceso de expansión incluye acciones antes de que se puedan aplicar realmente. Posteriormente, el proceso de extracción considera que las (sub)metas (derivadas

o no de las precondiciones de una misma acción) se consiguen de forma independiente. Esto da lugar a la aparición de acciones *no útiles* en el plan relajado. Una acción no es útil cuando su aplicación no ayuda a alcanzar los objetivos por los que fue seleccionada. En relación con la calidad de los planes, la aplicación de acciones no útiles es contraproducente. También puede serlo desde el punto de vista de la alcanzabilidad de la solución cuando no se puede deshacer y produce un camino sin salida o *dead-end*. El hecho de que haya acciones no útiles implica que en el plan relajado falten las acciones que realmente serían útiles.

En resumen, los planes relajados:se pueden ordenar de distintas maneras, presentan conflictos, presentan algunas acciones no útiles, y carecen de algunas acciones útiles.

En planificación clásica son múltiples las heurísticas que se utilizan para generar el estado *lookahead* (Vidal, 2004). En este apartado se analiza su aplicación en planificación basada en costes. Las que aparentemente son más adecuadas son:

(a) Generar primero los planes relajados ignorando las acciones que borran metas del problema.

(b) Aplicar tantas acciones como sea posible del plan relajado, que tiene por objetivo conseguir estados lo más profundos posible.

(c) Tener en cuenta a la hora de seleccionar y ordenar acciones el nivel del *RPG* en que se requiere el resultado de la acción, en lugar del primer nivel en que la acción aparece.

La primera evita que entre las acciones del plan relajado se incluyan acciones que borran las metas del problema, en caso de que esto sea posible. Si no es posible, el plan relajado se construye normalmente. Si el plan relajado contiene una acción que borra una meta del problema, probablemente carecerá de la acción que posteriormente la añade. Evitar la acción que borra la meta supone buscar un camino alternativo que en caso de encontrarlo puede evitar esa carencia. Una implicación de esta heurística es que la construcción del *RPG* inhibiendo acciones puede incrementar el valor numérico de la heurística, a menos que para cada estado se realice la evaluación dos veces.

La tercera deriva en dos heurísticas: (1) una que trata de evitar la inclusión en el plan relajado de acciones que pueden ser no útiles; y (2) otra que trata de ordenar el plan relajado de manera que las acciones con más posibilidades de ser no útiles se retrasen al máximo.

Para conseguir (1), lo que propone Vidal es que el nivel de cada (sub)meta en el proceso de extracción sea el nivel de la acción que requiere esa (sub)meta, en lugar del primer nivel del *RPG* en que aparece. Esto implica que el rango de acciones para seleccionar la (sub)meta sea mayor. Sin embargo, el hecho de que el rango de acciones sea mayor no implica que se seleccione la acción adecuada, ya que la selección de acciones es avara y no se ha definido ningún criterio que permita elegir la adecuada. Otra posibilidad es realizar una extracción con *backtracking*, pero igualmente sería necesario definir el criterio que determina cúal es la mejor acción. Por lo tanto las justificación de esta heurística desde el punto de vista intuitivo es dudosa. Por otro lado tampoco existen resultados experimentales que la contrasten. A continuación se expone un ejemplo ilustrativo sobre el ejemplo de la Figura 6.1.

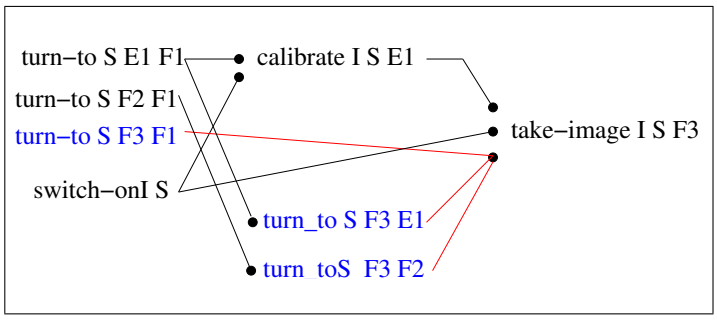

Figura 6.5: Esquema de proceso de extracción para la heurística (1) (planificación clásica).

La Figura 6.5 muestra un esquema del proceso de extracción al que daría lugar el retraso de las (sub)metas al nivel en que se requieren. En esta figura las (sub)metas, representadas con puntos, se encuentran en el mismo nivel de las acciones que las requieren. Concretamente, la (sub)meta *(pointing S F3)* requerida por la acción *take-image* (representada por el punto situado más abajo a la izquierda de la acción) antes (ver Figura 6.2) consideraba una única acción para generarla: la acción *(turn-to F3 F1)*. Obviamente esta acción es inútil porque no tiene sentido girar el satélite a la dirección en que debe tomar la imagen antes de haber calibrado el instrumento. Al emplear la heurística (1), se consideran además las acciones *(turn-to F3 E1)* y *(turn-to F3 F1)*. La acción correcta es la última, dado que el satélite después de calibrar el instrumento se encuentra apuntando a *F1*. El criterio para seleccionar esta acción requiere por tanto utilizar información relacionada con los borrados de las acciones. Si esta información no se utiliza, aunque está claro que la probabilidad de seleccionar la acción correcta aumentaría si se hiciera una selección aleatoria, no se puede garantizar que la acción seleccionada sea más útil que en caso de no usar la heurística.

Este problema se acentúa cuando la construcción del *RPG* se hace en niveles incrementales de coste. En el ejemplo, si se para en el primer nivel en que se encuentran las metas, las acciones alternativas no aparecen en el *RPG* (debido a su coste). Para garantizar que aparecen habría que construir el grafo hasta el punto fijo, con el correspondiente incremento de tiempo de cálculo de la heurística. En este caso, ocurriría lo mismo que en planificación clásica: habría que definir el criterio adecuado para seleccionar la acción correcta. Por las razones explicadas, la heurística (1) no se considerará en este trabajo.

Por el contrario, sí que parece más claro utilizar la idea (2) para ordenar las acciones del plan relajado según los niveles en que sus efectos se requieren. En el ejemplo para planificación clásica, esto supondría situar la acción no útil (*turn-to F3 F1*) después de las acciones que consiguen las precondiciones de *calibrate*, es decir de *(switch-on I S)* y *(turn-to E1 F1)*, que son acciones útiles. En el ejemplo con costes (ver Figura 6.3) ocurriría lo mismo, situando la acción no útil (*turn-to F3 F2*) después de las útiles, que son las que consiguen las precondiciones de *calibrate*. La forma concreta en que esta idea se ha llevado a la práctica se describe en el siguiente apartado.

Estos fenómenos, tanto para la heurística (1) como para la heurística (2) son comunes a muchos dominios de planificación ya que basta con que haya acciones que tienen varias

precondiciones y que existan diversas formas de conseguir algunas de ellas. El efecto de ignorar los borrados implica que se pierde la información sobre el orden en que se deben conseguir las precondiciones de las acciones, o en general el orden en que se deben conseguir las (sub)metas.

Por último, otra de las heurísticas propuestas en planificación clásica implica la reparación del plan relajado cuando no se pueden aplicar más acciones. Reparar el plan relajado puede ser útil para evitar evaluaciones en el algoritmo de búsqueda. Sin embargo, el propio algoritmo se encarga de repararlo. Por otro lado, en planificación con costes, cuando los planes de menor coste no son los planes de menor número de operadores, pueden existir secuencias de acciones que generan una (sub)meta con menor coste que una única acción, por lo que para no penalizar demasiado la calidad de los planes obtenidos mediante estados *lookahead*, la reparación debería realizarse con una secuencia de acciones en lugar de con una única acción, lo cual no es sencillo de llevar a la práctica.

6.2.1. Ordenación del plan relajado

Como se ha mencionado en el apartado anterior, una forma de retrasar las posibles acciones no útiles es ordenar las acciones según el nivel del *RPG* en que se requieren sus efectos, que denominaremos *nivel_requerido*. Esto es sencillo calcularlo durante el proceso de extracción:

- Inicialización: el nivel en que se requiere una meta del problema es el primer nivel del *RPG* en que aparece la meta. Sin embargo el nivel en que se requiere un hecho no meta se inicializa a infinito, al igual que el nivel en que se requieren cada una de las acciones del problema:

$$nivel\_requerido(g) = nivel(g), \forall g \in \mathcal{G}$$

$$nivel\_requerido(p) = \infty, \forall p \notin \mathcal{G}$$

$$nivel\_requerido(a) = \infty, \forall a \notin \mathcal{A}$$

- Cada vez que se selecciona una acción a, el nivel en que se requiere la misma se actualiza al mínimo entre el valor que tenga (inicialmente infinito) y el nivel en que se requiere el efecto (o (sub)meta) p por el que fue seleccionada.

$$nivel\_requerido(a) = \text{mín}\{nivel\_requerido(p), nivel\_requerido(a)\}$$

  .

- Cada vez que se introduce una (sub)meta p, precondición de una acción seleccionada a, el nivel en que se requiere p se actualiza al mínimo entre el nivel de la acción menos uno y el *nivel_requerido* que tenga p:

$$nivel\_requerido(p) = \text{mín}\{nivel(a) - 1, nivel\_requerido(p)\}$$

De esta forma, al finalizar el proceso de extracción, todas las acciones del plan relajado tienen asociado un valor (*nivel_requerido*) que permite ordenarlas parcialmente. El orden entre dos acciones con el mismo nivel requerido será el orden que tengan las acciones en el *RP*, es decir, estará ordenada antes aquélla que aparezca en un nivel anterior del *RPG*. El orden parcial así definido, se puede utilizar para generar el estado *lookahead*. Sin embargo, en planificación basada en costes, cuando los planes con menor coste no son los que tienen menor número de operadores, el nivel en que se requieren las acciones no consigue siempre el mismo efecto que en planificación clásica, debido a que aparecen caminos con más de una acción para conseguir (sub)metas que se podrían conseguir con una única acción. Supongamos el ejemplo anterior en el dominio satélite, pero con coste de la acción *(turn-to S E1 F1)* igual a 30 en lugar de 100, como muestra la Figura 6.6. En este caso las acciones *turn-to* que llevan al satélite a apuntar hacia la meta son más baratas que las acción que lo llevan a la dirección en que debe calibrarse. La Figura 6.7 muestra el extracto del *RPG* para $h_{level\text{-}max}$ en este caso.

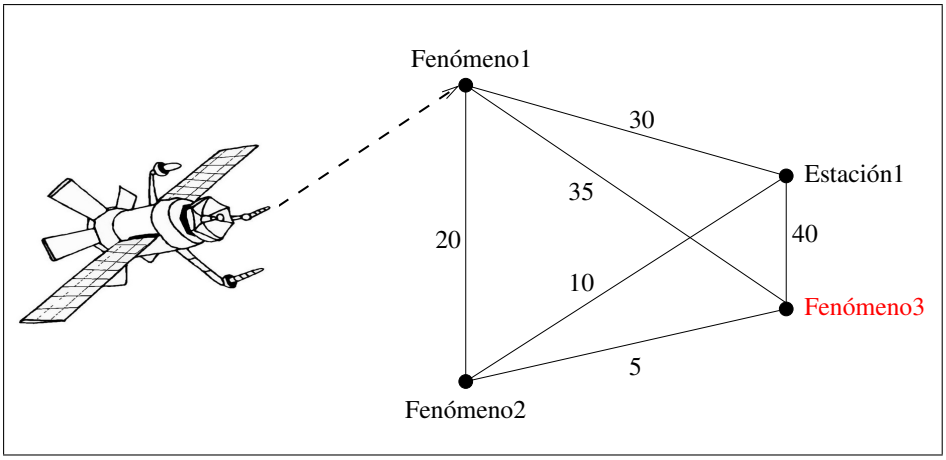

Figura 6.6: Ejemplo de un problema en el dominio del satélite.

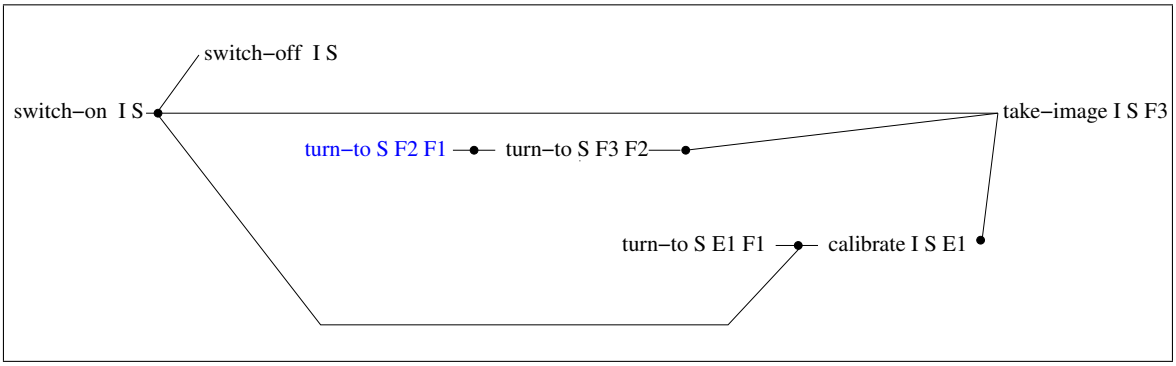

Figura 6.7: Esquema de proceso de extracción del *RPG* para $h_{level\text{-}max}$.

La acción que se debe elegir en este caso para conseguir el estado *lookahead* es *(turn-to S E1 F1)*, ya que es necesario calibrar el instrumento antes de tomar la imagen. Dar más

prioridad a las acciones cuyo *nivel_requerido* es menor, supone elegir antes *(turn-to S E1 F1)* que *(turn-to S F3 F2)*, lo cual es correcto. Sin embargo, para conseguir que el satélite apunte al fenómeno 3 se utiliza el camino más barato: *(turn-to S F2 F1)*, *(turn-to S F3 F2)*, y la acción *(turn-to S F2 F1)* es la que tiene un *nivel_requerido* menor, con lo que es la elegida para generar el estado *lookahead*. El problema es que en la extracción aparecen (sub)metas, y por lo tanto acciones, que se deben sólo a que el proceso está orientado a minimizar el coste y que no son necesarias para resolver el problema. Este problema se puede producir en todos los dominios en los que hay acciones con varias precondiciones que presentan interacciones entre ellas y se pueden conseguir mediante distintos caminos con distinto coste.

Una posible solución consiste en propagar hacia atrás el nivel requerido, de los efectos de las acciones hacia las precondiciones, en el caso de (sub)metas que se consiguen a través de un camino en el *RPG*, pero que dado el dominio se pueden conseguir en un solo paso. De esta forma el comportamiento será similar a planificación clásica. Con este objetivo se define el *nivel_requerido_propagado*, que trata de capturar el nivel requerido de las (sub)metas necesarias.

- Inicialización: el *nivel_requerido_propagado* de cada meta del problema $g \in \mathcal{G}$, es su *nivel_requerido*:

$$nivel\_requerido\_propagado(g) = nivel\_requerido(g), \forall g \in \mathcal{G}$$

- Cada vez que se selecciona una acción a se determina si la selección se debe a que el efecto que genera es necesario, o por el contrario su elección se debe a que genera un camino de menor coste. Para ello se analiza si el nivel en que es alcanzable la (sub)meta p por la que se selecciona la acción es igual al nivel en que es aplicable la acción más 1. Esto significa que sin tener en cuenta costes la acción a podría ser la seleccionada para conseguir la (sub)meta. Además, si el nivel requerido propagado de la (sub)meta coincide con su nivel requerido, se puede afirmar que las demás acciones seleccionadas en el camino en que se encuentra la (sub)meta se podrían seleccionar en planificación sin costes. Por lo que generan hechos necesarios. En este caso, la acción se elige porque genera una (sub)meta necesaria y su nivel requerido propagado es igual su nivel requerido. Lo explicado en este párrafo se puede expresar con el siguiente pseudocódigo:

  if $nivel\_alcanzable(p) = nivel\_aplicable(a) + 1$ y
  $nivel\_requerido\_prop(p) = nivel\_requerido(p)$ **then**
  $nivel\_requerido\_prop(a) = nivel\_requerido(a)$

  En otro caso, se puede decir que la acción se elige porque genera un camino de menor coste, y por lo tanto la (sub)meta que genera no es necesaria para resolver el problema. Así, el nivel requerido propagado de la acción se actualiza al mínimo entre el valor que tenga (inicialmente infinito) y el nivel requerido propagado del efecto (o (sub)meta) p por el que fue seleccionada. Es decir, se realiza la propagación. Además la acción se marca como acción no necesaria:

else

$nivel\_requerido\_prop(a) = \text{mín}\{nivel\_requerido\_prop(p), nivel\_requerido\_prop(a)\}$

$no\_necesaria(a) = \texttt{TRUE}$

- Cada vez que se introduce una (sub)meta p, precondición de una acción seleccionada a, si la acción está marcada como no necesaria, se realiza la propagación del nivel requerido propagado de a. En otro caso, el nivel requerido propagado de p es su nivel requerido:

  if $no\_necesaria(a)$ **then**

  $nivel\_requerido\_prop(p) = \text{mín}\{nivel\_requerido\_prop(p), nivel\_requerido\_prop(a)\}$

  else

  $nivel\_requerido\_prop(p) = nivel\_requerido(p)$

El efecto de este método en el ejemplo se muestra en la Figura 6.8. Como se puede observar, el nivel requerido propagado para la acción *(turn-to S F2 F1)* es 6, mayor que el nivel requerido propagado para *(turn-to S E1 F1)* que es 5. Así, esta última acción, que en este caso es la correcta, sería la prioritaria junto con la acción *(switch-on I S)*.

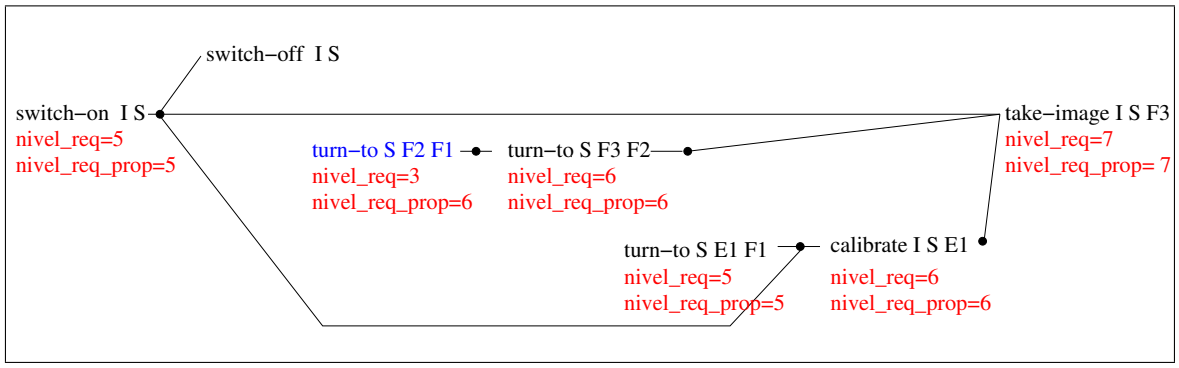

Figura 6.8: Niveles requeridos y requeridos propagados en el proceso de extracción del *RPG* para $h_{level\text{-}max}$.

En cualquier caso, el procedimiento es heurístico. Consigue el efecto que se conseguiría en planificación sin costes, pero al igual que en este caso, no garantiza siempre que se elijan siempre primero las acciones que llevan a conseguir las precondiciones que se deben conseguir antes, debido a las interacciones negativas. Para esto habría que incluir en el procedimiento información sobre este tipo de interacciones, relacionada con el orden en que se deben conseguir las precondiciones de cada acción.

En el siguiente apartado se detalla el algoritmo que permite conseguir un estado *lookahead* a partir del estado actual utilizando el nivel requerido propagado.

6.2.2. Algoritmo para generar el estado *lookahead*

La Figura 6.9 muestra el pseudocódigo del algoritmo para generar el estado *lookahead* a partir de un estado y su plan relajado, RP. Las variables *min_orden* y *max_orden* representan el mínimo y máximo nivel requerido propagado de las acciones del plan relajado. Inicialmente ninguna acción del plan relajado está ejecutada y se comienza intentando ejecutar acciones con orden mínimo (este es el orden actual inicial) en el estado actual. Esto se hace mientras el orden actual sea menor o igual que el máximo. En cada iteración se recorre el plan relajado. Cada vez que se encuentra una acción aplicable no previamente ejecutada, cuyo nivel requerido propagado sea el orden actual, ésta se ejecuta, se actualiza el estado y se vuelve a inicializar el orden actual al mínimo. Si después de recorrer el plan relajado no se ha ejecutado ninguna acción, el orden actual se incrementa en uno y se vuelve a repetir el proceso.

Cuando el estado que se consigue mediante la aplicación de una acción es un estado repetido con mayor valor de $g(n)$ que el ya existente, el algoritmo sale con fallo.

```
function obtener_estado_lookahead(estado, RP)
let min_orden = minimo_nivel_requerido_propagado(RP)
let max_orden = maximo_nivel_requerido_propagado(RP)
let ejecutada[a] = FALSE, ∀a ∈ RP
let orden_actual = min_orden
let estado_actual = estado
while orden_actual <= max_orden do
    nueva_accion_aplicada = FALSE
    forall a ∈ RP do
        if not ejecutada[a] ∧ aplicable(a, estado_actual)∧
          nivel_requerido_propagado(a) = orden_actual then
            nuevo_estado = aplicar(a, estado_actual)
            if repetido_con_mayor_g(nuevo_estado)
                return fail
            else
                ejecutada[a] = TRUE
                nueva_accion_aplicada = TRUE
                estado_actual = nuevo_estado
                orden_actual = min_orden
    if not nueva_accion_aplicada then
        orden_actual = orden_actual + 1
if estado_actual ≠ estado
    return estado_actual
else
    return fail
```

Figura 6.9: Algoritmo para la generación del estado *lookahead*.

6.2.3. Problemas en la aplicación de la heurística *lookahead*

El uso de los estados *lookahead* en un algoritmo de búsqueda puede mejorar notablemente la escalabilidad del algoritmo. Sin embargo, es más extraño desde el punto de vista intuitivo que sirva para encontrar soluciones de coste bajo. Aún partiendo de un plan relajado sensitivo a los costes de las acciones, existen varias causas que dan lugar a un incremento del coste de los planes encontrados mediante la generación consecutiva de estados *lookahead*:

- La aplicación de acciones *no útiles*, es decir, que no sirven para conseguir lo objetivos marcados, debido bien a que el plan relajado no contiene las acciones *útiles* porque se construye ignorando las interacciones negativas; o bien porque aún conteniéndolas el orden seleccionado no es el correcto. En el ejemplo anterior la acción *(turn-to F2 F1)* es una acción no útil. Si se ejecuta, antes que *(turn-to E1 F1)* incrementará el coste del plan conseguido porque o bien será necesario deshacerla, o bien generará un camino más costoso.

- La instanciación de los recursos, donde se entiende por recurso cualquier objeto, no directamente relacionado con el objetivo, que se utilice para conseguirlo. Por ejemplo si en el dominio *Zenotravel* una meta es que una persona se encuentre en un determinado lugar, es necesario utilizar un avión. El avión se puede considerar como un recurso, ya que no está directamente relacionado con el objetivo pero éste no se puede conseguir sin utilizarlo. Esta definición de recurso depende del tipo de objetivos perseguido en cada caso. Otros ejemplos de recursos pueden ser los instrumentos en el dominio del *Satellite*, los conductores y camiones en el *Driverlog* cuando el objetivo es que ciertos objetos estén en ciertos lugares, etc. Si en el estado evaluado un recurso está libre, éste se considerara libre en todo el plan relajado. Así, en el plan relajado siempre se elegirá el recurso más barato dada la situación del estado evaluado, sin tener en cuenta que el estado del recurso puede haber cambiado.

- La continuidad se pierde cuando se generan estados *lookahead* consecutivos, lo cual puede llevar a que acciones útiles pasen a ser no útiles. Esto también está relacionado con el tratamiento de los recursos. Por ejemplo, supongamos que en el dominio *Satellite* un mismo satélite debe tomar dos imágenes con instrumentos distintos, I_1 e I_2. Sin embargo un único instrumento puede estar encendido en un momento dado. En el primer estado *lookahead* el primer instrumento I_1 está encendido y calibrado (supongamos que es más barato calibrar I_1 que I_2). A continuación se genera un nuevo estado *lookahead* partiendo de éste. Supongamos que es más barato calibrar el segundo instrumento que dirigirse a tomar la imagen con el instrumento I_1. En este caso, en el siguiente estado *lookahead* se habrá apagado I_1 y calibrado I_2. Una vez apagado un instrumento hay que volver a calibrarlo. Por lo tanto, las acciones en principio útiles, que llevaron a calibrar I_1 no sirven más que para incrementar el coste del plan que se encuentra siguiendo este camino.

A pesar de esto, actualmente no conocemos una forma distinta de la experimental para determinar el compromiso real entre la escalabilidad y la calidad en cada caso. En apartados posteriores se plantea que el algoritmo de búsqueda continúe buscando nuevas soluciones

una vez encontrada la primera. En este marco, puede ser más útil encontrar rápidamente soluciones de coste medio que tardar mucho en encontrar una solución de coste bajo.

6.3. Resumen

En este capítulo se ha analizado la aplicación de dos heurísticas no numéricas: las *helpful actions* y la generación de estados *lookahead* en planificación basada en costes.

Caben al menos dos opciones para la generación de acciones *helpful* en planificación basada en costes, que se estudiarán experimentalmente en la siguiente parte de la tesis.

Respecto a la generación de estados *lookahead* se ha planteado un procedimiento que toma algunas heurísticas de planificación clásica. Otras ideas, como el dar más prioridad a las acciones cuyos efectos de requieren antes se han adaptado para que no den lugar a comportamientos erráticos. Al final del capítulo se detalla un posible algoritmo para generar estados *lookahead* teniendo en cuenta el análisis previo y se detallan algunos de los problemas que el uso de estados *lookahead* puede generar en planificación basada en costes.

En cualquier caso, al generar tanto las acciones *helpful* como los estados *lookahead* utilizando planes relajados que se construyen teniendo en cuenta los costes de las acciones, ambas heurísticas tienen cierta información en este aspecto y pueden ser útiles en planificación basada en costes. En la siguiente parte de la tesis se estudian distintos algoritmos de búsqueda, algunos de los cuales incorporan ambas heurísticas y se valora experimentalmente su utilidad.

Parte III

Algoritmos de búsqueda y resultados experimentales

Capítulo 7

Marco experimental

Esta parte de la tesis está dedicada a estudiar el comportamiento de distintos algoritmos de búsqueda combinados con distintas heurísticas, tanto numéricas como no numéricas, en planificación basada en costes. En cada capítulo se estudia un algoritmo de búsqueda y posibles variaciones sobre el mismo. Cada capítulo incluye experimentos relacionados con el algoritmo que se estudia y experimentos que lo contrastan con algoritmos estudiados en capítulos anteriores. La idea es ir estableciendo de forma incremental las combinaciones que son más adecuadas al mismo tiempo que se van descartando opciones. Esta estructura está motivada por el hecho de que combinar todos los algoritmos de búsqueda y sus variaciones con todas las posibles heurísticas generaría un gran número de experimentos que es inviable llevar a cabo en un tiempo razonable. El marco experimental, que será común a todos los experimentos de los capítulos posteriores se define en este primer capítulo. El marco experimental describe: el planificador parametrizado que permite comparar los algoritmos; los métodos de evaluación que permitirán interpretar los resultados experimentales; los dominios de planificación a utilizar; y finalmente, los planificadores existentes que se utilizarán para establecer si los mejores resultados obtenidos son competitivos.

7.1. El planificador CBP (*Cost-Based Planner*)

Tradicionalmente, en las Competiciones Internacionales de Planificación (IPCs) se contrasta el comportamiento de distintos planificadores utilizando dominios y problemas comunes. Los planificadores que compiten pueden emplear distintas técnicas de planificación, por lo que los resultados finales de la competición aportan luz sobre las técnicas más competitivas. Sin embargo, cada planificador suele incluir una combinación de mecanismos, por lo general heurísticos, orientados a dirigir el proceso de planificación. Esto hace complicado determinar qué heurística o heurísticas, dentro de la misma técnica, es más adecuada en general o a un dominio particular. Esto ocurre porque se compara el planificador como un todo sin aislar cada uno de los mecanismos que le permiten tomar determinadas decisiones.

Uno de los objetivos de esta tesis doctoral es, sin embargo, estudiar el comportamiento

que se obtiene haciendo variaciones de las heurísticas sobre la misma técnica de planificación: búsqueda heurística progresiva. Para que esto sea posible se ha construido un planificador parametrizado que permite seleccionar distintos algoritmos de búsqueda y distintas heurísticas, denominado CBP (*Cost-Based Planner*). Este planificador, que forma parte de la herramienta PLTOOL (Fernández et al., 2007), proporciona un marco común a todos los experimentos. En esta sección se describen algunas características importantes del mismo.

CBP se ha implementado sobre el código del planificador METRIC-FF, e incluye todos los algoritmos de búsqueda que se describen en los capítulos siguientes. En relación con las heurísticas numéricas, se ha implementado la construcción de *RPG* en niveles incrementales de costes, siguiendo la misma filosofía que la implementación de METRIC-FF para planificación clásica. Así, se cuenta con las siguientes heurísticas:

- Heurísticas derivadas de la propagación de costes con el máximo: $h_{level\text{-}max}$, h_{max}, y $h_{add\text{-}max}$.
- Heurísticas derivadas de la propagación de costes aditiva: $h_{level\text{-}add}$, h_{add}, y $h_{max\text{-}add}$.
- Heurística de METRIC-FF original.

Puesto que para calcular h_{max}, $h_{add\text{-}max}$, h_{add} y $h_{add\text{-}max}$ no es necesario realizar el proceso de extracción, estas heurísticas no proporcionan acciones *helpful*. Sin embargo, se da la opción de realizar la extracción para generarlas. En este caso, las acciones *helpful* para h_{max} y $h_{add\text{-}max}$ coinciden con las de $h_{level\text{-}max}$, y las acciones *helpful* para h_{add} y $h_{max\text{-}add}$ coinciden con las de $h_{level\text{-}add}$.

Por otro lado, en los tres casos, del mismo proceso mediante el que se calculan las heurísticas mencionadas se extrae una estimación del número de operadores que faltan por aplicar para conseguir un estado meta, coherente con la estimación de costes. En el caso de $h_{level\text{-}max}$ y $h_{level\text{-}add}$ es el número de acciones del plan relajado extraído. En el caso de h_{max}, es el nivel en el *RPG* (de $h_{level\text{-}max}$) de la meta con máximo nivel, y en el caso de h_{add} es la suma de los niveles en el *RPG* (de $h_{level\text{-}add}$) de cada meta [1]. En general, nos referiremos a esta estimación del número de operadores como h_{ops}.

En todas las heurísticas, los empates se solucionan utilizando la medida de la *dificultad* correspondiente, definida en la sección 4.6, página 107.

7.2. Métodos de evaluación

Las distintas IPCs que se han venido celebrando proporcionan facilidades para evaluar el rendimiento de un planificador y compararlo con el de otros planificadores. Esto es así por varios motivos:

1. A raíz de estas competiciones existe un lenguaje común, PDDL (cuya última versión a día de hoy es PDDL3.0). Esto permite hacer comparaciones entre distintos

[1] Otra posibilidad para la estimación del número de operadores en estos casos, cuando además se realiza la extracción para generar acciones *helpful*, es tomar el número de acciones del plan relajado.

planificadores de una manera estándar.

2. Existe una gran cantidad de dominios y problemas en esos dominios definidos utilizando PDDL, a los que se puede acceder vía web.

3. En las competiciones se han establecido criterios para evaluar los planificadores.

Para la parte experimental de la tesis se utilizarán algunos dominios propuestos en las IPCs. Puesto que el ámbito de la tesis es planificación basada en costes, es necesario que en estos dominios el coste de las acciones no sea uniforme (al menos de algunas acciones) y que haya una métrica de calidad definida en los problemas, siguiendo la sintaxis de PDDL2.1. Otras características que permite modelar PDDL2.1, como las precondiciones y metas numéricas, las acciones durativas, las preferencias, etc. se deben eliminar, dado que no caben en el modelo de planificación basada en costes planteado.

Los experimentos, por lo general, consistirán en proporcionar problemas a las configuraciones a comparar para obtener uno o varios planes solución, y posteriormente medir un conjunto de variables. Estos experimentos se pueden dividir en dos partes:

- Experimentos que involucran exclusivamente al planificador parametrizado CBP. En estos experimentos se variarán principalmente los parámetros relativos a la(s) heurística(s) y al algoritmo de búsqueda que se utiliza. En cualquier caso todos estos experimentos se realizarán en el marco de búsqueda heurística.

- Experimentos que permitan comparar con otros planificadores existentes. En este caso, el número de variables a comparar será menor cuando el planificador con que se compara no utiliza la misma técnica de planificación.

Todos los experimentos se han realizado una misma máquina Linux, con un procesador Intel Core2 Quad a 2.66GHz y una memoria máxima para el proceso de planificación de 2Gb.

Las variables independientes de los experimentos, que se fijan a priori, y son comunes a todos ellos son:

- Los dominios de planificación y los conjuntos de problemas en cada dominio, que por lo general son de complejidad incremental. Estos dominios se describen en la siguiente sección.

- La métrica de calidad de los problemas en el mismo dominio.

- El tiempo límite para encontrar la solución, que será de 1800 segundos.

En los experimentos que conciernen al planificador CBP, se fijarán tanto las heurísticas que se utilizan como el algoritmo de búsqueda concreto.

Las variables dependientes, que permitirán medir los resultados obtenidos, serán principalmente:

- El número de problemas resueltos.
- El número de estados o nodos evaluados por el algoritmo de búsqueda.
- El tiempo de CPU que el planificador emplea en encontrar la solución (o soluciones) a los problemas planteados.
- El coste de la solución o soluciones encontradas.
- La relación entre las dos variables anteriores: tiempo de CPU y coste. Esta variable es de especial importancia para los objetivos de la tesis que se plantea.

En los experimentos que impliquen comparar varios planificadores (o bien el mismo con distintos parámetros), se calcularán los porcentajes de mejora de uno sobre otro en relación con las variables dependientes mencionadas.

7.3. Dominios

En esta sección se describen los dominios elegidos para la experimentación. Los cuatro primeros dominios se han tomado de la tercera IPC. Los tres siguientes, son dominios diseñados para planificación basada en costes por E. keyder y H. Geffner (Keyder and Geffner, 2008); y el último dominio se ha tomado de la quinta IPC. En los dominios tomados de IPCs se han eliminado todas las ocurrencias de variables numéricas tanto en las precondiciones de las acciones como en las metas del problema; y por uniformidad, se ha fijado la misma métrica en todos los problemas.

Los dominios sobre los que se han planteado los experimentos no tienen estados sin salida. Las heurísticas que se han estudiado para planificación basada en costes detectan los mismos estados sin salida que las mismas heurísticas para planificación sin costes. Por lo tanto, en este sentido no aportan nada nuevo. Es posible que los valores numéricos de la heurística puedan llevar a un algoritmo de búsqueda concreto a descartar o elegir un camino sin salida no detectado por la heurística, aunque este hecho se puede considerar más bien casual. Por otro lado, en planificación clásica con búsqueda heurística los algoritmos más robustos respecto a los estados sin salida son los algoritmos más completos, que no toman decisiones irrevocables y no realizan podas. Lo mismo ocurre en planificación basada en costes. Por lo tanto tampoco tiene demasiado interés considerarlos desde el punto de vista de los algoritmos.

A continuación se explican los dominios y algunas de sus características más relevantes:

- *Zenotravel*: en el que hay aviones que vuelan de una ciudad a otra y pueden transportar personas. Existen dos tipos de acciones para desplazar los aviones: *fly* y *zoom*. Esta última consume más combustible (aunque que esto sea así no depende del dominio, sino de la tasa de consumo que se expresa en cada problema). El coste de estas acciones es el combustible gastado que se calcula como la distancia recorrida por la tasa de consumo del avión que se utiliza. Existen también acciones para embarcar y desembarcar personas: *board* y *debark*. Estas acciones no tienen coste. La métrica

fijada es el combustible gastado más el número de operadores. En este dominio no hay efectos secundarios ya que cada acción tiene sólo un efecto positivo.

- *Satellite*: en el que hay satélites que deben tomar imágenes de distintos fenómenos en el espacio, utilizando los instrumentos apropiados. Existen acciones para girar satélites (*turn-to*), para encender y apagar instrumentos (*switch-on* y *switch-off*), para calibrar instrumentos (*calibrate*) y para tomar imágenes (*take-image*). Cada satélite sólo puede tener encendido un instrumento en un momento dado y además únicamente se pueden tomar imágenes con instrumentos calibrados. Las acciones *turn-to* consumen una cantidad combustible que se calcula directamente a partir del tiempo que se tarda en girar de la posición inicial a la final (este tiempo es el mismo para todos los satélites). La métrica fijada es el combustible más el número de operadores. En este dominio, no hay efectos secundarios ya que cada acción tiene sólo un efecto positivo, y no hay estados sin salida ya que todas las acciones se pueden deshacer o bien producen hechos que no borra ninguna otra acción (como la acción *take-image*).[2].

- *Driverlog*: en el que hay camiones en los que se pueden cargar objetos, y que necesitan un conductor para desplazarse de entre ciudades conectadas (bidireccionalmente) por carreteras. Los conductores también pueden desplazarse a pie por caminos (también bidireccionales). Existen acciones para desplazar un camión con conductor (*drive*), para caminar (*walk*), para cargar y descargar objetos (*load*, *unload*) y para embarcar y desembarcar del camión un único conductor (*board*, *disembark*). Las acciones *walk* y *drive* consumen un tiempo (distinto entre ellas) que depende de la distancia recorrida. La métrica fijada es el tiempo total más el número de operadores. En este dominio:

  - Podrían darse efectos secundarios, ya que la acción *disembark* consigue tanto que el conductor esté en el lugar en que desembarca, como que el camión quede vacío (sin conductor).
  - Hay que realizar *path planning*.

- *Depots*: que es una combinación del mundo de los bloques con el dominio *logistics*. Los objetos pueden ser transportados de un lugar a otro utilizando camiones y una vez en un lugar pueden ponerse unos encima de otros. Existen acciones para desplazar camiones *drive*, para levantar y depositar objetos mediante grúas (*lift* y *drop*), y para cargar y descargar objetos en los camiones (*load* y *unload*). En este dominio el coste de las acciones es fijo para todas las acciones del mismo tipo, siendo: para *drive* 10, para *lift* 5, para *drop 4*, para *load* 3, y para *unload* 2. La métrica definida en los problemas es minimizar el coste total del plan. En este dominio:

  - Podrían darse efectos secundarios, ya que las acciones *lift*, *drop* ,*load* y *unload* generan varios hechos, que incluyen la definición de la situación de la grúa y la definición de la situación de los objetos.
  - Presenta muchas interacciones entre metas, es decir, las metas y (sub)metas están muy interrelacionadas, de manera que conseguir unas puede tanto facilitar como dificultar la consecución de otras.

[2] J. Hoffmann(Hoffmann, 2005) clasifica este tipo de espacios de estados como inofensivos o *harmless*.

- *Assignment*: es una simplificación del Sokoban. En este dominio únicamente hay una acción que permite mover una bola de su localización a una localización meta libre. Una vez que una posición meta se ha ocupado, no se puede liberar. El objetivo es que todas las bolas se encuentren en una posición meta. En los problemas hay tantas localizaciones meta libres como bolas, por lo tanto todos los planes posibles tienen la misma longitud (incluyen una acción por cada bola, para situar la bola en una posición meta). El coste de la única acción del dominio es la distancia entre la localización de la bola y la meta. La métrica a minimizar es la distancia total. En este dominio:

  - No hay efectos secundarios.
  - Todos los planes tienen la misma longitud, que es exactamente el número de bolas en el problema.
  - Las acciones *helpful* son todas las acciones aplicables, tanto usando la opción HA1 como la opción HA2, descritas en la sección 6.1.1, página 137. Es decir, todas aquéllas que sitúan cada bola no situada en una localización meta en cada una de las localizaciones meta libres.
  - No hay efectos secundarios, ya que aunque la acción tiene dos efectos positivos, uno para representar que la bola está en una posición meta (*(ball-at-goal ?ball)*) y otro que relaciona la bola con su posición actual (*(at-ball ?ball ?position)*, únicamente este último se encuentra en las precondiciones. Por lo tanto, es el único que puede ser una submeta en el proceso de extracción del RP.
  - $h_{level\text{-}max}$, $h_{level\text{-}add}$ y h_{add} coinciden en este dominio ya que aunque la acción (*put-ball-in-goal-loc ?from ?to*) tiene dos precondiciones (*(at-ball ?b ?from)* y *(free-goal-loc ?to)*), los planes relajados contienen una acción que situa cada bola que no se encuentra en una posición meta, en una posición meta libre en el estado evaluado y no tienen secuencialidad. Por este motivo el coste de las precondiciones de las acciones que aparecen en los planes relajados es siempre cero y las heurísticas mencionadas coinciden.[3] Las tres heurísticas proporcionan como valor la suma de distancias resultante de situar cada bola (de forma independiente) en la posición libre cuya distancia a la posición actual de la bola es menor. Por otro lado h_{mff}, tiene en cuenta el coste de situar cada bola en una posición libre cualquiera, que se elige arbitrariamente (de hecho, se elige la misma posición destino para todas las bolas).
  - No hay estados sin salida, ya que en los problemas se asegura que haya tantas posiciones meta como bolas.

- *MST* (*Minimum Spanning Tree*). En este dominio sólo existe una acción, (*connect*), que permite conectar un nodo aún no conectado con un nodo ya conectado. Esta acción supone el coste del arco elegido para realizar la conexión. Inicialmente, en todos los problemas hay un nodo conectado y el objetivo es conectar los demás nodos. No hay restricciones sobre las conexiones, cada nodo se puede conectar con cualquier otro. La métrica fijada es la suma de los costes de los arcos seleccionados. En este dominio:

[3] Por la forma de calcular la estimación en número de operadores (h_{ops}) en el caso de h_{add}, ésta difiere de $h_{level\text{-}max}$ y $h_{level\text{-}add}$, por lo que si el algoritmo de búsqueda utiliza h_{ops}, los resultados pueden diferir.

- La acción *connect* no tiene efectos negativos. Por lo tanto, el dominio relajado coincide con el dominio original. Desde el punto de vista de planificación es un dominio sencillo en el sentido de que es muy fácil encontrar una solución. El problema es encontrar la solución óptima.
- No hay efectos secundarios
- Todos los planes tienen la misma longitud, que es exactamente el número de nodos del problema menos uno.
- Con el esquema HA1 (sección 6.1.1, página 137) las acciones *helpful* son todas las acciones aplicables, ya que siempre se puede conseguir conectar cualquiera de los nodos restantes en un sólo paso (es decir, con una sola acción) (y la submeta que genera las acciones *helpful* es siempre *(connected ?node)*). En el caso de HA2 (sección 6.1.1, página 137) las acciones *helpful* quedan restringidas a conectar aquellos nodos que en el plan relajado se conectan en un solo paso.
- $h_{level\text{-}max}$ y $h_{level\text{-}add}$ coinciden en este dominio, ya que la única acción del dominio sólo tiene una precondición dinámica (hay otra precondición que se refiere a un hecho estático). Estas heurísticas no son perfectas por considerar que las acciones se pueden aplicar en paralelo y que las metas son independientes. La Figura 7.1 muestra un ejemplo en el que inicialmente está conectado el nodo A. La forma más barata de conectar B supone un coste de 20, y la forma más barata de conectar C supone un coste de 25. Así las heurísticas mencionadas proporcionan un valor de 45. Sin embargo el coste óptimo es 30 ya que el hecho de conectar primero B implica que exista una forma más barata de tener C conectado (con coste 10).
- No hay estados sin salida ya que aunque no se puedan desconectar nodos, siempre se pueden conectar los nodos restantes.

- *Delivery*: que es una simplificación del dominio TPP, en el que hay camiones que se pueden desplazar entre distintos lugares mediante la acción *drive*. El coste de esta acción es el combustible que se consume, que viene determinado por la distancia recorrida y el coste de la unidad de combustible. Además se pueden comprar productos y cargarlos en el camión mediante la acción *buy-good*. El coste de esta acción es el precio del producto en el lugar en que se haya comprado (el mismo producto puede tener distintos precios en distintos lugares). El objetivo es que ciertos productos estén en ciertos lugares, por lo que hay una acción que permite descargar los productos de los camiones: *unload-good*. Esta acción tiene siempre el mismo coste. La métrica fijada es minimizar el coste total. En este dominio no hay efectos secundarios.
- *Pipesworld*: en el que distintos productos derivados del petróleo deben propagarse por una red de *pipelines*. Éste es un domino complejo en el que los planificadores clásicos resuelven pocos problemas.

7.4. Otros planificadores

En los últimos experimentos se contrastarán los mejores resultados obtenidos con otros planificadores con el fin de determinar si las aproximaciones que se proponen en esta tesis proporcionan resultados competitivos. Con este propósito se realizarán comparaciones con:

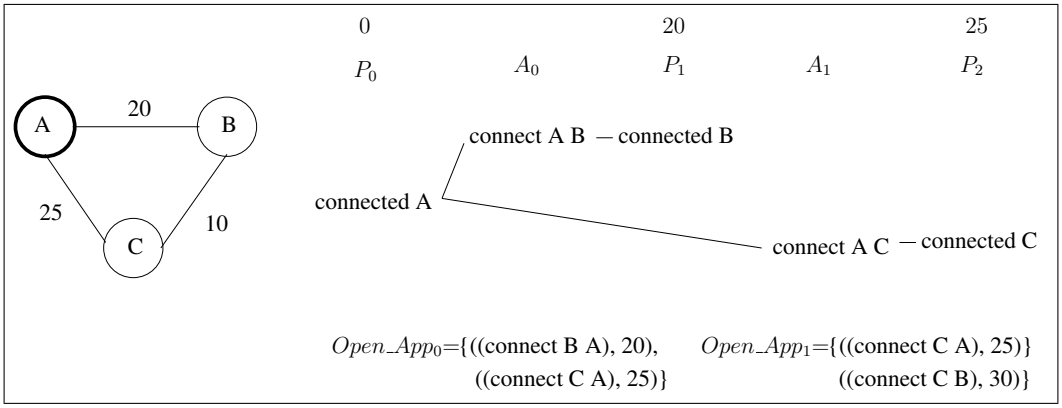

Figura 7.1: Ejemplo de *RPG* en el dominio *MST*.

- METRIC-FF (Hoffmann, 2003)
- LPG (Gerevini et al., 2004)
- LAMA (Richter and Westphal, 2008)

METRIC-FF, hoy por hoy, no es un planificador muy competitivo a la hora de encontrar planes de bajo coste; sin embargo es la base de CBP, y en éste se utilizan prácticamente las mismas estructuras de datos y la misma filosofía en la implementación de los algoritmos de búsqueda, por lo tanto resulta interesante contrastar la mejora que se obtiene.

LPG es un planificador que utiliza técnicas de planificación distintas a las empleadas en esta tesis. Sin embargo, es un planificador robusto y que proporciona soluciones razonables en términos de coste. Además se ha utilizado ya en algunas ocasiones para hacer comparativas en este sentido. LPG es un planificador estocástico, y por lo tanto distintas ejecuciones pueden dar lugar a distintas soluciones. Al contrario, los procedimientos que se plantean en la tesis son sistemáticos. Para realizar las comparaciones, con planificadores sistemáticos, los autores de LPG suelen ejecutar el planificador 5 veces para cada problema en cada dominio (cada una de ellas con el límite de tiempo con que se ejecutan los sistemáticos) y toman los resultados correspondientes a la mediana de las 5 ejecuciones en la variable que quieren medir. En nuestro caso, se adoptará también esta estrategia, tomando la solución que proporciona la mediana en términos de coste de 5 ejecuciones, cada una de ellas con un tiempo límite de 1800 segundos.

LAMA se puede considerar hoy en día el mejor planificador para planificación basada en costes. Fue el ganador de la última IPC, celebrada en 2008. Es un planificador que realiza búsqueda heurística en el espacio de estados inducido por el propio problema de planificación, al igual que las aproximaciones que se plantean en esta tesis. Utiliza también heurísticas numéricas y no numéricas en un contexto de búsqueda *mejor primero* avara. Las heurísticas numéricas resultan de combinar la heurística de FF con una heurística que se extrae del análisis de *landmarks* (estado de la cuestión, sección 2.4.4.5). La comparación con LAMA no se puede realizar fácilmente en todos los dominios que se han planteado. El

motivo es que en LAMA la expresividad del lenguaje PDDL está restringida. Concretamente, no admite operaciones aritméticas en los efectos numéricos de las acciones. Así, el coste de cada acción debe expresarse directamente como un valor o mediante el uso de una única variable numérica estática. En cuatro de los dominios propuestos (*Zenotravel*, *Satellite*, *Delivery* y *Pipesworld*), existen operaciones aritméticas para calcular el coste de las acciones. Para ejecutar LAMA en estos dominios habría que procesar estas operaciones aritméticas en todos los problemas, lo cual es un trabajo pesado. Por este motivo sólo se realizará la comparación en los dominios restantes.

Capítulo 8

Esquemas derivados de algoritmos de búsqueda local

El éxito de un proceso de planificación depende tanto de las heurísticas utilizadas como de los algoritmos de búsqueda que se aplican. En esta parte de la tesis se analizan distintos algoritmos de búsqueda a utilizar en planificación basada en costes. En la mayoría de los casos los algoritmos son incompletos y no admisibles, es decir, no se garantiza encontrar una solución y aún menos la óptima. El objetivo es establecer algoritmos y heurísticas que permiten obtener una relación razonable calidad/tiempo. En este primer capítulo se introducen dos algoritmos derivados de esquemas de búsqueda local y algunas variantes de los mismos. Asimismo, se presentan los resultados empíricos derivados de los experimentos realizados con estos algoritmos.

8.1. Cost Enforced Hill-Climbing (CEHC)

El algoritmo *Cost Enforced Hill-Climbing* (CEHC) es una extensión para planificación basada en costes del algoritmo *Enforced Hill-Climbing* (EHC). EHC, como se describió en el estado de la cuestión (sección 2.3.4.1, página 35), es uno de los algoritmos de búsqueda que ha tenido más éxito en planificación clásica. Es un algoritmo de búsqueda local, que consiste en una búsqueda en amplitud a partir del estado actual hasta encontrar un sucesor directo o indirecto que mejore su evaluación heurística. Una vez encontrado, este sucesor pasa a ser el estado actual. El proceso es irrevocable y por lo tanto las decisiones tomadas no se revisan. Sin embargo, se mantiene una tabla *hash* con los estados generados durante cada búsqueda en amplitud, para evitar repetir evaluaciones y bucles.

En el algoritmo EHC para planificación sin costes, se aplican dos tipos de heurísticas de distinta naturaleza:

- Una heurística numérica, concretamente la heurística de FF, donde los empates se resuelven mediante la heurística de la dificultad y después de forma arbitraria, y
- Una heurística de poda, concretamente la heurística de las acciones *helpful*.

En este apartado se estudia la aplicación de *EHC* en planificación basada en costes. Para ello se plantea:

1) Utilizar una heurística que estime costes, lo cual parece adecuado puesto que se trata de encontrar planes de mínimo coste y
2) Ordenar las acciones que generan los sucesores de cada nodo en orden creciente de costes. Así, el proceso de búsqueda explorará antes sucesores que se obtienen con acciones menos costosas. Es una decisión local, con lo cual no muy informada, pero tiene dos ventajas: (1) no implica evaluar el sucesor; (2) al menos asegura elegir la acción menos costosa en caso de que en el dominio se de el caso de que varias acciones con distinto coste generan los mismos hechos.

Las heurísticas que estiman costes proporcionan un valor más pequeño cuanto menor es el coste estimado de alcanzar un estado meta. El valor mínimo que puede tomar la heurística es cero. Necesariamente, los estados meta tienen una evaluación heurística de cero. Sin embargo, estados que no son meta también pueden dar lugar a una evaluación heurística de cero. Esto ocurre cuando en el dominio existen acciones sin coste y todas las acciones que la heurística estima que se deben aplicar para alcanzar el estado meta son acciones sin coste. Así, caben dos opciones para determinar si un estado es meta:

- Comprobar que todas las proposiciones meta se encuentran en el estado, o
- Utilizar también una heurística que estime número de acciones y comprobar que esta heurística proporciona un valor de cero en el estado evaluado.

En CEHC se ha optado por la segunda opción porque el valor de las heurísticas se estima a partir de un conjunto de acciones, con lo cual, cuando se conoce este conjunto de acciones, calcular una estimación del número de acciones supone una carga computacional mínima (realizar una suma). Además, la estimación de número de acciones se puede utilizar también para resolver empates en la heurística de costes de la siguiente manera: cuando la estimación de coste de un nodo y uno de sus sucesores coincide, se elegirá el sucesor si su estimación de número de acciones es menor que la del padre. La idea intuitiva es que ante planes con el mismo coste, son por lo general preferibles aquéllos que tienen menos operadores.

Por otro lado, en los estados se representan tanto las proposiciones como las variables numéricas implicadas en la métrica. Así, estados con las mismas proposiciones y distintos valores para las variables numéricas son estados distintos, pero tienen la misma evaluación heurística. Para evitar evaluar el mismo estado varias veces durante la búsqueda en amplitud se realiza una poda de estados repetidos (manteniendo los ya explorados en una tabla *hash*) en la que se considera que un estado es igual a otro si contiene los mismos hechos independientemente de los valores de las variables numéricas.

Salvo estas modificaciones el algoritmo *Cost Enforced Hill-Climbing (CEHC)* es similar a EHC. La Figura 8.1 muestra su pseudocódigo. Inicialmente, la lista ABIERTA, en la que se irán almacenando los nodos a expandir, está vacía, y el estado raíz de la búsqueda local

es el estado inicial, que también es el estado actual. En el algoritmo, h_{cost} representa la estimación heurística en términos de coste y h_{ops} la estimación heurística correspondiente a h_{cost} en términos de número de operadores. Cuando la estimación heurística relativa a número de operadores es cero, es porque el estado contiene todas las metas, y por lo tanto es un estado solución. Mientras esto no ocurra, cada vez que el estado actual tenga mejor valoración heurística que el estado raíz (el coste restante estimado desde estado actual es menor que desde el estado raíz, o bien es igual pero el número de operadores restante estimado desde el estado actual es menor que desde el estado raíz), la búsqueda se reinicia tomando el estado actual como nuevo estado raíz y reinicializando la lista ABIERTA al conjunto vacío.

```
function CEHC(I,G)
let plan = ∅;
let ABIERTA = ∅;
let nodo_raiz=I;
let nodo_actual=I;
let usar_helpful_actions=TRUE;
evaluar(I)
while h_ops(nodo_actual) ≠ 0 do
    if h_coste(nodo_actual) < h_coste(nodo_raiz)
      or ((h_coste(nodo_actual) = h_coste(nodo_raiz)
            and (h_ops(nodo_actual) < h_ops(nodo_raiz)) then

        plan = plan + camino(nodo_raiz, nodo_actual)
        nodo_raiz = nodo_actual
        ABIERTA = ∅
        usar_helpful_actions= TRUE
    else
        if usar_helpful_actions then
            sucesores= sucesores_helpful(nodo_actual)
        else
            sucesores= sucesores(nodo_actual)
        ABIERTA = ABIERTA + ordenar(sucesores)
        if ABIERTA do
            nodo_actual = pop(ABIERTA)
            evaluar(nodo_actual)
        else if usar_helpful_actions then
            usar_helpful_actions = FALSE
        else return fail
return plan
```

Figura 8.1: Búsqueda Cost Enforced Hill-Climbing.

Cuando el nodo actual no mejora la evaluación heurística del nodo raíz, se añaden sus sucesores debidos a acciones *helpful* al final de la ABIERTA, y a continuación se toma como nodo actual el siguiente de esta lista. Los sucesores se ordenan según el coste de la acción

que los genera.

Al igual que en EHC, en caso de que para un determinado nodo raíz, la lista ABIERTA se quede vacía, se desactiva la poda por acciones *helpful* y se vuelve a realizar el mismo proceso desde el nodo raíz actual. Una vez encontrado el sucesor por el que continuar, la poda se vuelve a activar. Cuando esto no ocurre (es decir, no hay un sucesor con mejor evaluación heurística que el nodo raíz) el algoritmo falla.

Cabe destacar que las acciones *helpful* se extraen del mismo proceso que se utiliza para generar la evaluación heurística numérica de cada nodo. Cuando este proceso difiere (es decir, para heurísticas numéricas distintas), las acciones *helpful* también pueden ser diferentes.

CEHC es un algoritmo de búsqueda incompleto, que no tiene ningún mecanismo para continuar la búsqueda si encuentra un estado sin salida, por lo que si en un momento determinado se elige un estado estado sin salida el algoritmo fallará.

8.1.1. Variación a CEHC

Una variación posible a CEHC es seleccionar el sucesor s para el que se minimiza $coste(a) + h(s)$, donde $coste(a)$ representa el coste de la acción que lo genera, siempre que para este sucesor se cumpla que su evaluación heurística es menor que la del nodo raíz actual. Sin embargo, esta aproximación supone evaluar todos los sucesores de cada estado y por lo tanto es más costosa, sobre todo si esto se hace en todos los niveles de la búsqueda en amplitud, dado que el número de nodos crece exponencialmente. Una opción menos costosa, sería utilizar este criterio sólo en el primer nivel. Si ningún nodo en el primer nivel tiene mejor heurística que el padre, se cambia la estrategia, a la original de CEHC: ordenando los sucesores según el coste de la acción que los genera, y seleccionando aquél que mejore la evaluación heurística del nodo raíz.

8.1.2. Resultados experimentales (I)

En esta sección se describen los experimentos realizados con el algoritmo CEHC. El primer experimento trata de determinar el comportamiento del algoritmo con las distintas definiciones de acciones *helpful*. El segundo, el comportamiento de CEHC cuando se utilizan distintas heurísticas numéricas. El tercero está orientado a evaluar el impacto de utilizar las mismas acciones *helpful* con varias heurísticas numéricas. El cuarto, a evaluar el impacto de utilizar distintas distribuciones de coste para el mismo problema desde el punto de vista de los resultados que se producen con distintas heurísticas, y el último a contrastar el comportamiento de CEHC con la variación que se explicó en el apartado anterior.

8.1.2.1. Experimento 1: acciones *helpful* en CEHC

El objetivo de este experimento es determinar qué opción para calcular las acciones *helpful* es más adecuada en el contexto de una búsqueda CEHC. Para ello se ha ejecutado CEHC con la misma heurística numérica y las dos opciones de cálculo de acciones *helpful*. En concreto, con las siguientes configuraciones:

- HA1: Heurística numérica $h_{level\text{-}max}$ y cálculo de las acciones *helpful* según la opción HA1.
- HA2: Heurística numérica $h_{level\text{-}max}$ y cálculo de las acciones *helpful* según la opción HA2.

La descripción de las opciones HA1 y HA2 se encuentra en la página 137,de la sección 6.1.1.

Las variables dependientes de este experimento son:

- El coste del plan obtenido en cada caso,
- El número de problemas resueltos,
- El número de nodo evaluados, y
- El tiempo que se tarda en conseguir un plan.

Las Figuras 8.2 y 8.3 muestran los resultados. En cada dominio se muestra: en la gráfica de la izquierda el coste de los problemas que indica el eje X; en la gráfica central el número de estados evaluados; y en gráfica de la derecha el tiempo empleado por el algoritmo en encontrar la solución (en estas dos últimas gráficas el eje Y está en escala logarítmica). En las tres gráficas los puntos del eje X sin valor corresponden a problemas no resueltos y dan lugar a discontinuidades. Los resultados en azul corresponden a la primera configuración, HA1, y los resultados en rojo a la segunda, HA2.

La Tabla 8.1 muestra un resumen de los resultados respecto a calidad y número de problemas resueltos. En esta tabla, las columnas indican: el número de problemas en cada dominio, el número de problemas resueltos en que HA2 obtiene igual, menor y mayor calidad que HA1, el número de problemas resueltos utilizando HA1 y el número de problemas resueltos utilizando HA2.

	nprobs	calidad (HA2 vs. HA1)			resueltos HA1	resueltos HA2
		igual	peor	mejor		
zenotravel	20	20	0	0	20	20
satellite	20	0	0	15	15	17
driverlog	20	11	1	2	15	15
depots	22	20	0	2	21	21
assignment	40	40	0	0	39	40
mst	20	15	0	1	16	20
delivery	32	6	1	24	31	32
pipesworld	41	11	1	1	13	15
TOTAL	215	72,4 %	1,8 %	26,5 %	79,1 %	83,7 %

Tabla 8.1: Resumen de la comparación entre distintas *helpful actions* con $h_{level\text{-}max}$.

A la vista de los resultados, se puede decir que la diferencia entre utilizar los esquemas HA1 y HA2 para calcular las acciones *helpful* no es en general muy significativa ya que en

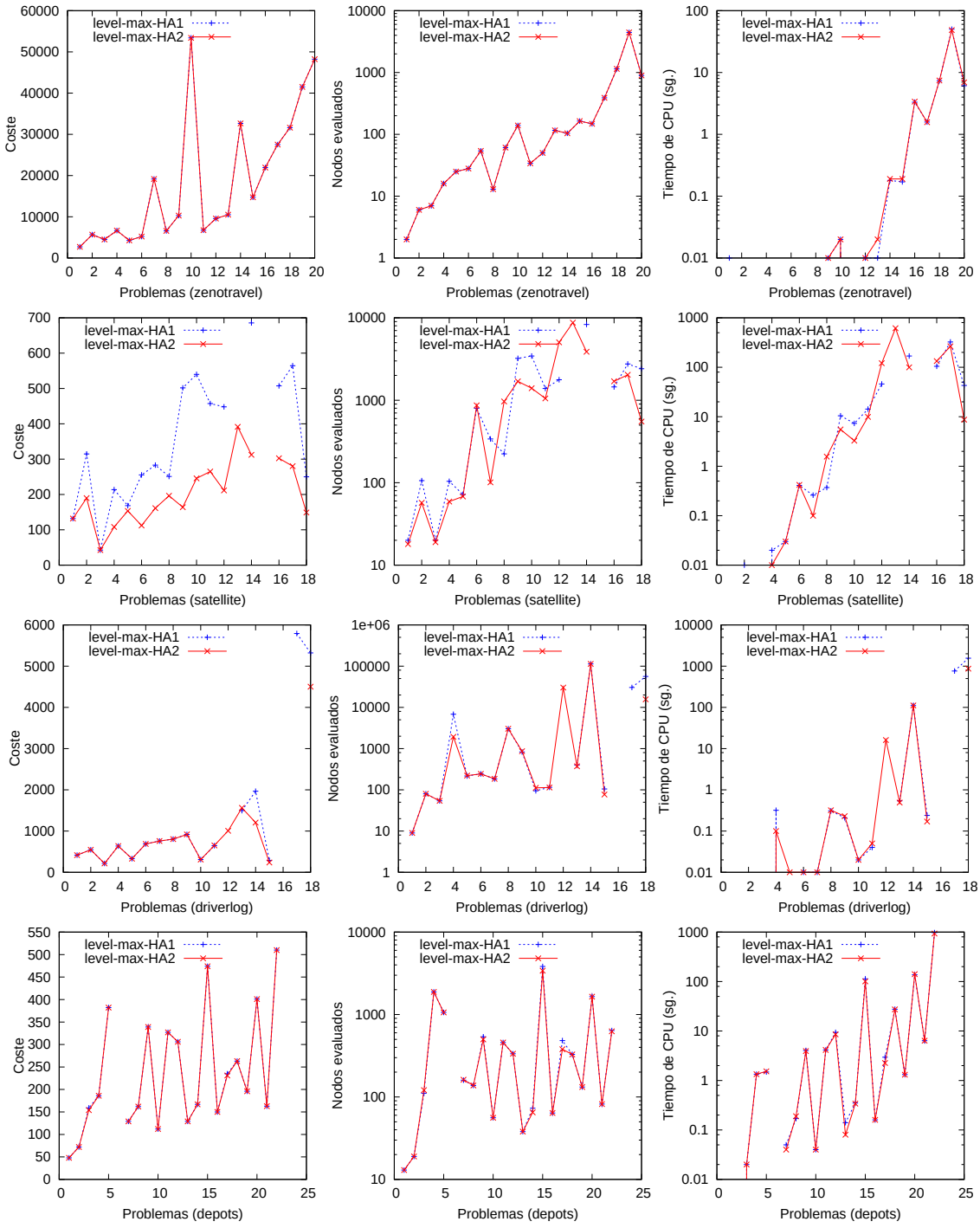

Figura 8.2: Resultados de la comparación entre distintas acciones *helpful* con $h_{level\text{-}max}$ (I).

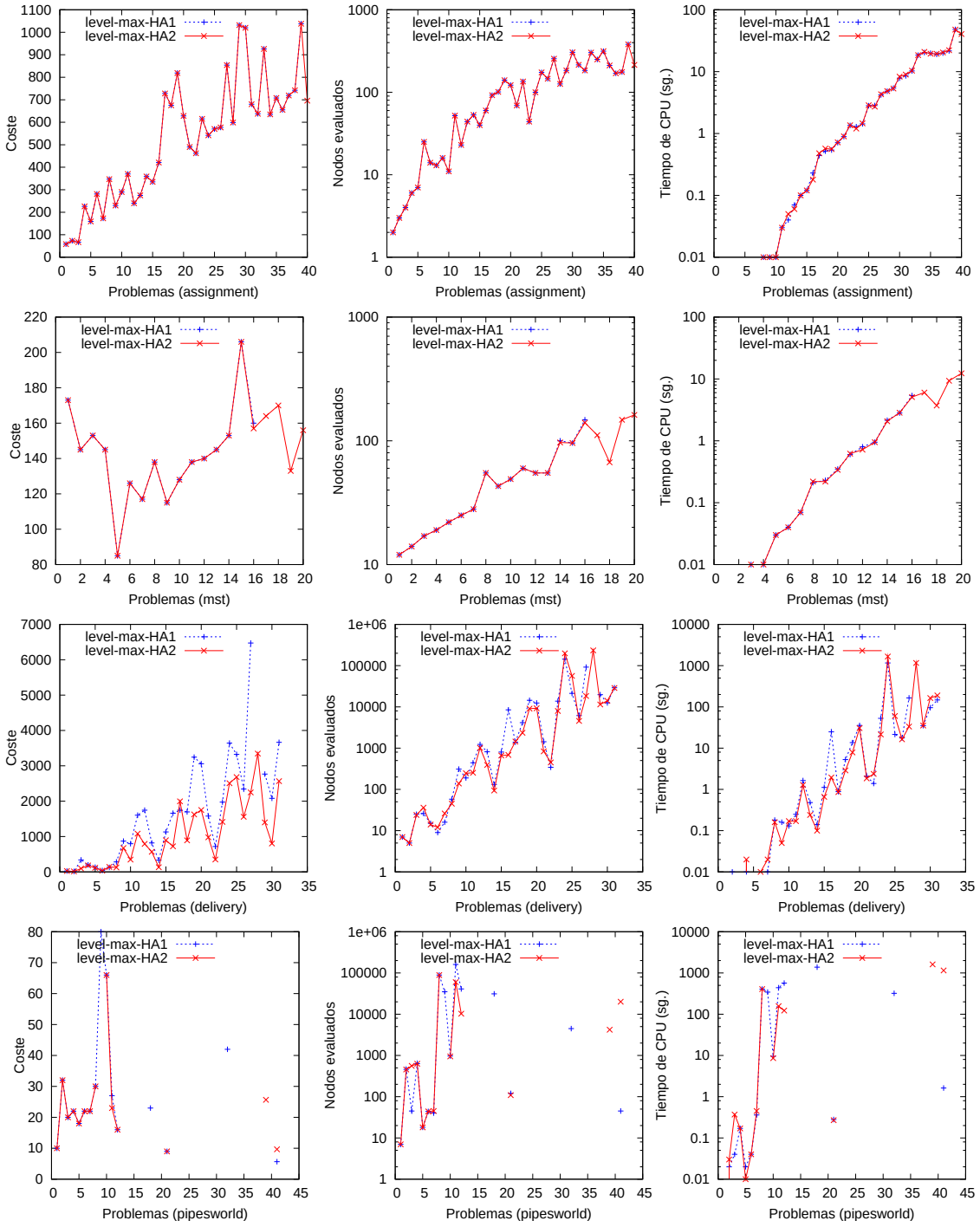

Figura 8.3: Resultados de la comparación entre distintas acciones *helpful* con $h_{level\text{-}max}$ (II).

un $72,4\,\%$ de los casos la solución encontrada es la misma. Sin embargo en dos dominios (*Satellite* y *Delivery*) se encuentran planes con una calidad considerablemente mejor utilizando HA2, sin que esto suponga una gran diferencia en términos de número de estados evaluados y tiempo. En relación con el número de problemas resueltos, el porcentaje de problemas que se resuelven con HA2 es un poco mayor que el porcentaje de problemas que se resuelven con HA1.

El mismo experimento se ha repetido utilizando la heurística $h_{level\text{-}add}$, obteniendo resultados muy similares. La Figura 8.4 muestra los resultados en los dominios en los que existen diferencias más significativas entre el uso de HA1 y HA2.

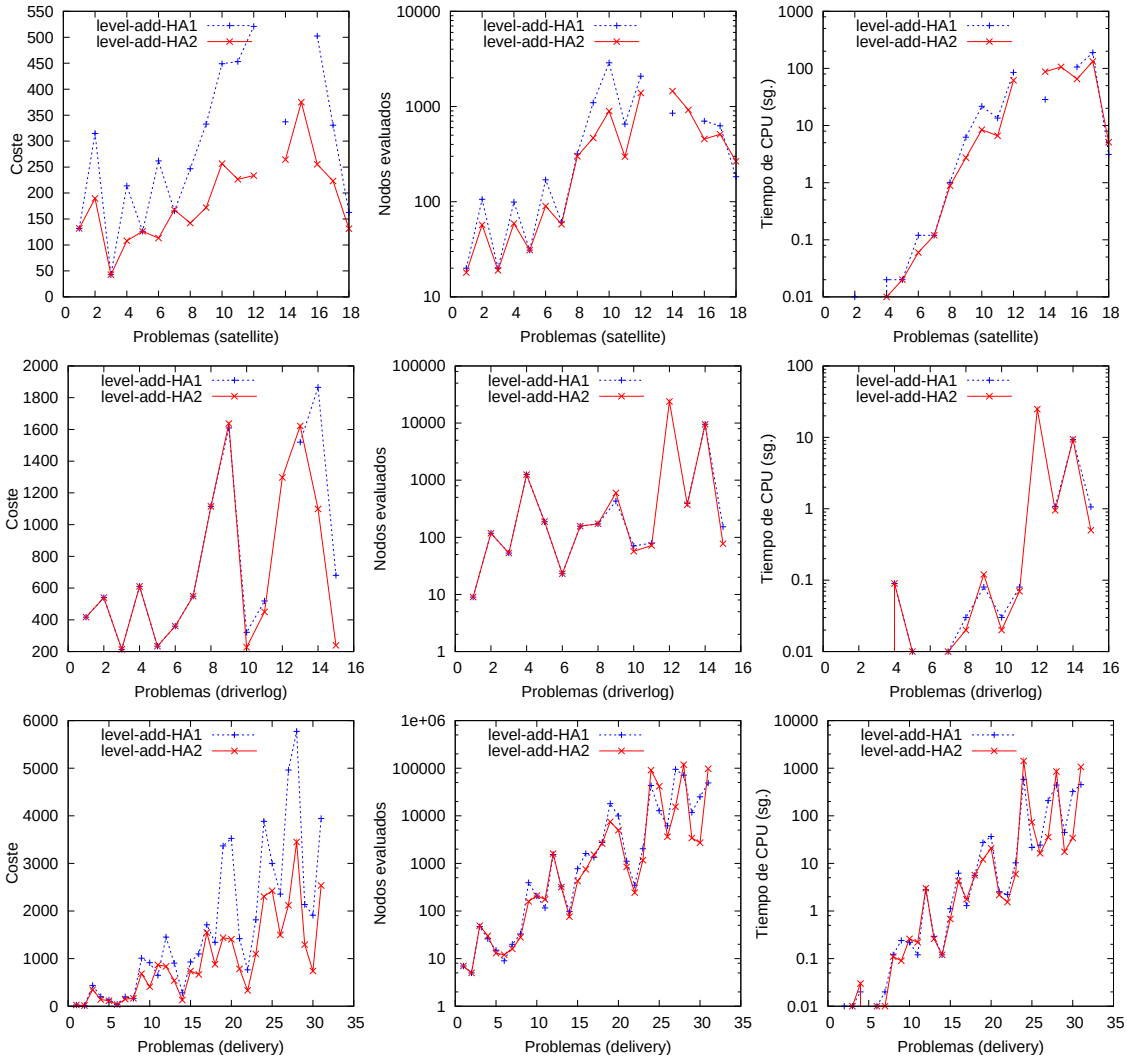

Figura 8.4: Resultados de la comparación entre distintas acciones *helpful* con $h_{level\text{-}add}$.

El comportamiento de CEHC depende tanto de las acciones *helpful* como de la forma en que éstas se ordenan. En realidad las acciones *helpful* que se obtienen con HA2 son un subconjunto de las acciones que se obtienen con HA1. El hecho de que las acciones *helpful* se ordenen en orden creciente de costes implica que la relación de orden entre las acciones HA2 se cumpla también en HA1. Sin embargo, en HA1 entre estas acciones pueden aparecer otras. La Figura 8.5 muestra un pequeño ejemplo en el que los círculos representan las (sub)metas de un proceso de extracción. Supongamos que la única meta del problema es la marcada en azul. Supongamos también que la forma más barata de conseguir esta meta es a través del camino que inicia a_1, y que a_1 es la acción que el proceso de expansión selecciona para generar la última (sub)meta del camino. Sin embargo, existen otras acciones aplicables más caras que a_1, como son $a_2, \ldots, a_n$, capaces de generar esta (sub)meta y que pueden no aparecer en el *RPG*. Además existen acciones aplicables, también más caras que a_1, que son $b_1, \ldots, b_m$ capaces de generar la meta del problema. En este caso, la primera acción *helpful* tanto para HA1 como para HA2 es a_1. Tanto en HA1 como en HA2, las acciones $a_2, \ldots, a_n$ estarán ordenadas siguiendo la misma relación de orden, que viene determinada por sus costes. Sin embargo, las acciones $b_1, \ldots, b_m$ no forman parte de HA2 y sí de HA1. La relación de orden entre cada acción de este conjunto y cada acción $a_2, \ldots, a_n$ depende de la distribución de costes del problema, por lo que se pueden intercalar. En este caso, utilizando HA1 se explorarán nodos generados por acciones de tipo b antes de explorar todos los nodos generados por las acciones tipo a.

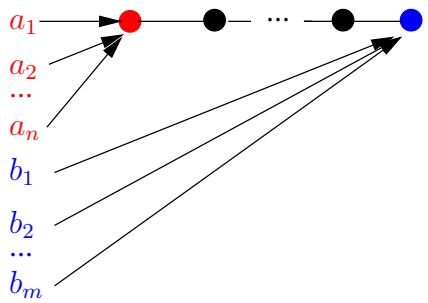

Figura 8.5: Acciones *helpful* HA1 vs. HA2.

Por otro lado, si se da el caso de que ninguna acción de HA2 consigue un estado con mejor evaluación heurística que el padre, con el esquema HA2 se iniciará el siguiente nivel de profundidad en la búsqueda en amplitud, mientras que en HA1 es posible que quede alguna acción por aplicar que sí consigue mejorar la evaluación del padre. En este caso, se continuará por el estado generado y el comportamiento de CEHC será distinto.

Si en todos los casos, la acciones que HA1 y HA2 comparten al comienzo de la ordenación consiguen estados con evaluación heurística mejor que el padre (o son necesarias para conseguirlos en la búsqueda en amplitud) el comportamiento de CEHC será el mismo tanto con HA1 como con HA2.

En general, según los experimentos, el uso del esquema HA2 no empeora los resultados que se consiguen utilizando el esquema HA1 y en algunos dominios incluso mejora considerablemente la calidad de los planes encontrados. Por lo tanto, HA2 es el esquema que se aplicará en los experimentos posteriores.

8.1.2.2. Experimento 2: heurísticas en CEHC

El objetivo de este experimento es contrastar el comportamiento de CEHC utilizando distintas heurísticas numéricas (lo que implica también utilizar distintas acciones *helpful*). Para ello se ha ejecutado CEHC con las siguientes heurísticas: $h_{level\text{-}max}$, $h_{level\text{-}add}$, h_{mff}, h_{add}, h_{max}, h_{add} con las acciones *helpful* que proporciona $h_{level\text{-}add}$ ($h_{add\text{-}HA}$), y h_{max} con las acciones *helpful* que proporciona $h_{level\text{-}max}$ ($h_{max\text{-}HA}$). En todas las configuraciones, el esquema para calcular las acciones *helpful* es el que propone la opción 2, HA2.

Las Figuras 8.6 y 8.7 muestran los resultados de los experimentos relativos al coste de las soluciones encontradas y al número de nodos evaluados. Las gráficas de tiempo se encuentran en el apéndice A. Todas las gráficas de las Figuras 8.6 y 8.7 son *gráficas de probabilidad*. En las gráficas de probabilidad, se representa el porcentaje de problemas resueltos a medida que la variable que se mide (situada en el eje X) va incrementando. Por ejemplo, en las gráficas de coste, cada punto indica el porcentaje de problemas que se resuelven con coste menor o igual al valor que indica el eje X (este porcentaje se puede interpretar como una probabilidad si se considera que el número de problemas es representativo). Los puntos a partir de los cuales se forma cada curva son los diferentes costes que se encuentren en el conjunto de problemas. Dada una curva, el valor máximo que se alcanza en el eje Y indica el porcentaje total de problemas resueltos. Se considera que las curvas que aparecen por encima de otras se corresponden con un mejor comportamiento respecto a la variable que indica el eje X, ya que el hecho de que una curva aparezca por encima significa que con esa configuración se resuelven más problemas con coste (nodos evaluados o tiempo, dependiendo de lo que indique el eje X) menor o igual que el valor que indica el eje X.

El tipo de gráficas explicado, que se utilizará en también en los experimentos posteriores, aporta una visión general clara de los resultados respecto al número de problemas resueltos y a la variable que indica el eje X, cuando se compara un número de configuraciones elevado (más de tres o cuatro). En este caso, en las clásicas gráficas en las que en el eje X se sitúan los problemas y en el eje Y la variable que se mide, puede ser bastante complicado distinguir cada una de las configuraciones si no se presentan valores acumulados. Para esto es necesario utilizar sólo los valores de los problemas que todas las configuraciones resuelven en común, lo cual no tiene sentido cuando hay configuraciones que resuelven muy pocos problemas (como ocurre en algunos casos en este experimento).

Los principales inconvenientes de las gráficas que se presentan son: (1) no aportan información relativa a cada problema particular, por lo cuál se deben complementar con las gráficas clásicas para contar con esta información; y (2) hay que ser cauteloso al interpretarlas cuando el número de problemas resueltos por dos configuraciones es distinto. En este caso el hecho de que una curva siempre aparezca por debajo de otra no implica necesariamente que en todos los problemas resueltos en común la configuración que se encuentra por debajo encuentre valores mayores para la variable que se mide. Por este motivo, en el

apéndice A se incluyen también las gráficas usuales, en las que se representan los problemas en el eje X, y el coste de la solución encontrada para cada uno de ellos en el eje Y.

Las Tablas 8.2 y 8.3 resumen las diferencias entre la calidad de las soluciones encontradas con las heurísticas h_{max} y h_{add} respectivamente, con y sin acciones *helpful*. En cada tabla se muestra el porcentaje de problemas (sobre los resueltos en común) en que se obtiene igual, mejor y peor calidad utilizando acciones *helpful* que no utilizándolas. A la vista de estas tablas y de las gráficas anteriores se pueden realizar las siguientes observaciones generales:

- La utilización de acciones *helpful* es determinante para la búsqueda CEHC, no sólo para encontrar la solución, sino también para encontrar soluciones de calidad. Tanto para h_{max} como para h_{add} se obtienen por lo general resultados considerablemente mejores en calidad utilizando acciones *helpful*, aunque con h_{max} se resuelven muchos menos problemas. Por otro lado, esta mejora de la calidad no supone un incremento en el número de nodos evaluados, sino todo lo contrario. Por último, al igual que en planificación clásica, se resuelven más problemas utilizando acciones *helpful* dado que la búsqueda en amplitud es demasiado pesada (requiere muchas evaluaciones) si no se reduce el factor de ramificación. Las excepciones respecto a la mejora de calidad utilizando acciones *helpful* son los dominios *MST* donde el comportamiento es el mismo con y sin acciones *helpful*; y *Pipesworld*, donde el uso de acciones *helpful* empeora la calidad de las soluciones encontradas. En *MST*, CEHC pasa siempre al nodo generado por la acción más barata (primer sucesor), ya que este nodo siempre tiene mejor evaluación heurística que su nodo padre. Por este motivo, la solución se encuentra rápidamente. La acción *helpful* más barata siempre es la acción aplicable más barata. Por este motivo el comportamiento es el mismo con y sin acciones *helpful*.
- Las heurísticas con un comportamiento más adecuado en general, respecto a la calidad de los planes obtenidos, son $h_{level\text{-}max}$, $h_{level\text{-}add}$ y $h_{add\text{-}HA}$. Entre estas dos últimas, las propiedades teóricas garantizan que $h_{add\text{-}HA} \geq h_{level\text{-}add}$. El tiempo de CPU en los tres casos es bastante parecido y se encuentra por lo general en el mismo orden de magnitud. Sin embargo este tiempo puede llegar a ser algún orden de magnitud mayor que el utilizado por h_{mff}. Ésta es la heurística que por lo general resuelve los problemas más rápidamente. Sin embargo, en algunos dominios la calidad de los planes que encuentra es considerablemente peor que las calidades obtenidas con las otras tres.

Otras observaciones interesantes son:

- En algunos dominios, las heurísticas basadas en propagación de costes aditiva $h_{level\text{-}add}$ y $h_{add\text{-}HA}$ evalúan menos nodos que $h_{level\text{-}max}$. Sin embargo, la diferencia en tiempo es menor. En algunos casos (como en *Zenotravel*) el hecho de evaluar menos nodos no implica utilizar menos tiempo, sino lo contrario ($h_{level\text{-}max}$ es la heurística que utiliza menos tiempo después de h_{mff}). Este efecto se debe a que el tiempo de cálculo de las heurísticas aditivas es por lo general mayor que el de las que utilizan una propagación de costes con el máximo. En los *RPGs*, las acciones se retrasan más debido a que su coste acumulado es mayor, lo que da lugar a *RPGs* con más capas. Este hecho se produce claramente en los dominios *Zenotravel Satellite Driverlog* y *Pipesworld*.

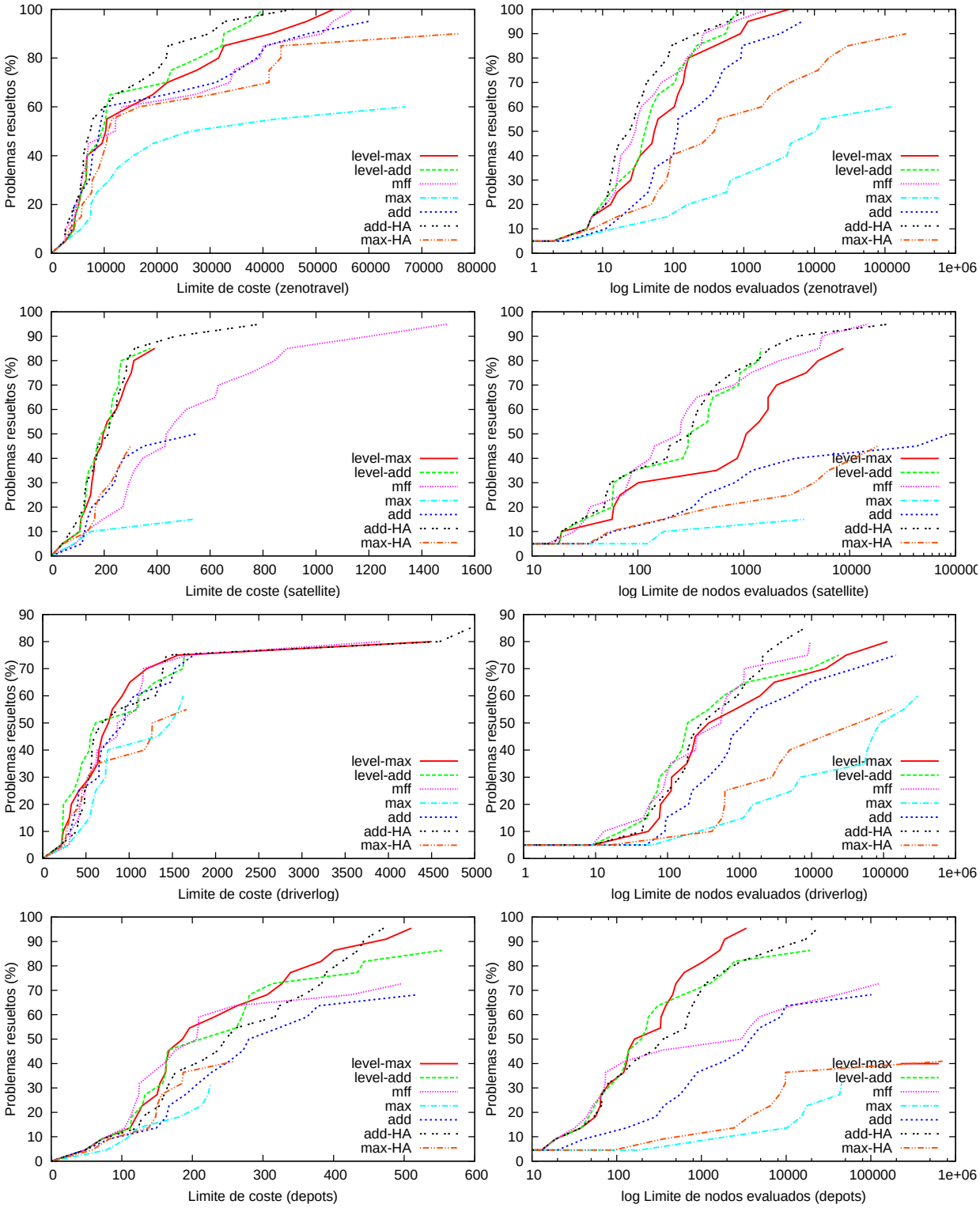

Figura 8.6: Resultados de la comparación entre distintas heurísticas en CEHC (I).

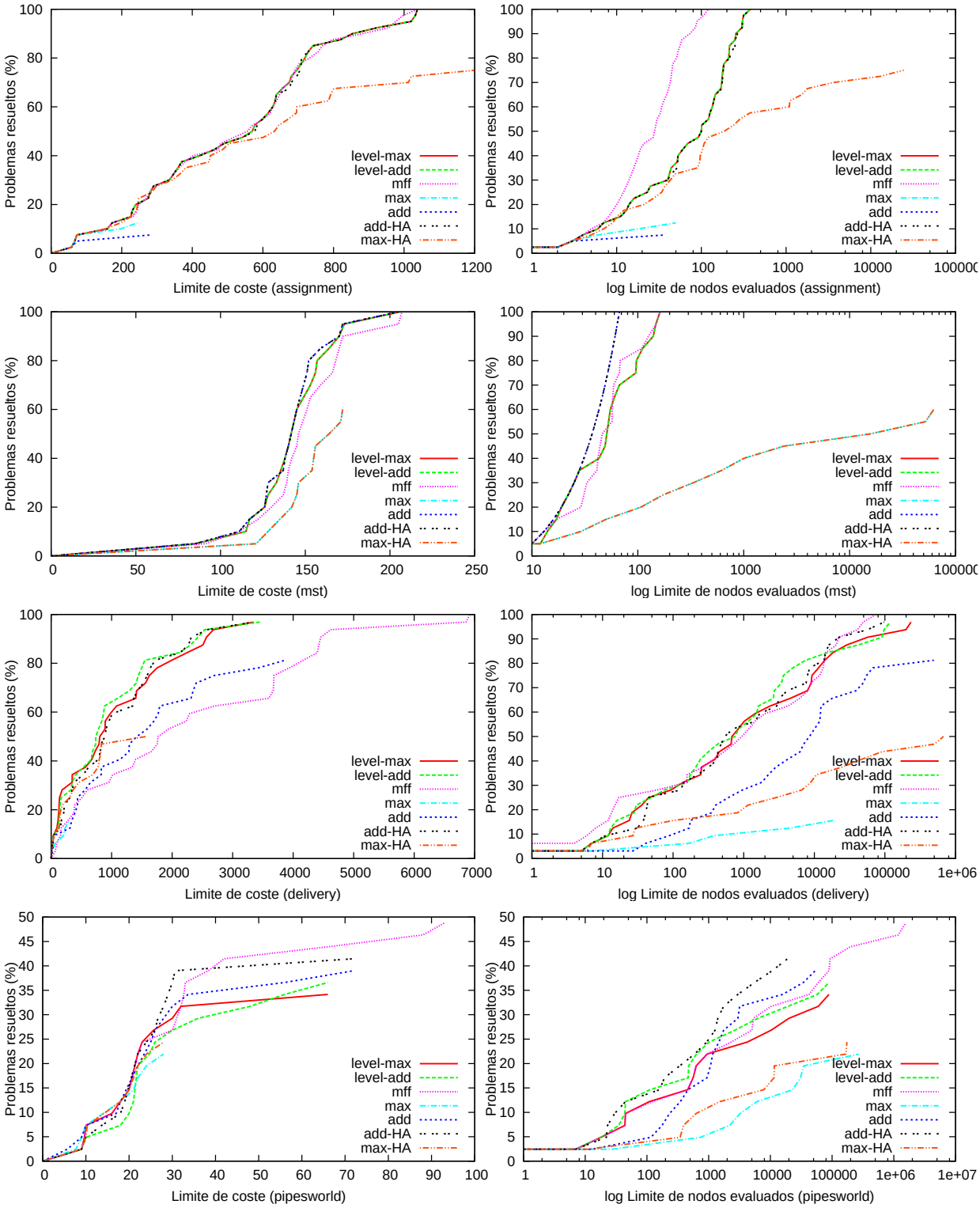

Figura 8.7: Resultados de la comparación entre distintas heurísticas en CEHC (II).

	nprobs	calidad (HA vs. no HA)			resueltos HA	resueltos no HA	resueltos comunes
		igual	mejor	peor			
zenotravel	20	2	7	3	18	12	12
satellite	20	0	3	0	9	3	3
driverlog	20	0	8	2	11	11	10
depots	22	0	6	1	9	7	7
assignment	40	2	1	0	30	3	3
mst	20	12	0	0	12	12	12
delivery	32	1	4	0	16	5	5
pipesworld	41	8	0	1	10	9	9
TOTAL	215	41,0 %	47,5 %	11,5 %	53,5 %	28,8 %	61

Tabla 8.2: h_{max} en CEHC: mejora en calidad utilizando acciones *helpful* (HA).

	nprobs	calidad (HA vs. no HA)			resueltos HA	resueltos no HA	resueltos comunes
		igual	mejor	peor			
zenotravel	20	3	15	1	20	19	19
satellite	20	1	8	1	18	10	10
driverlog	20	2	8	5	15	15	15
depots	22	1	13	1	21	15	15
assignment	40	3	1	1	40	5	5
mst	20	20	0	0	20	20	20
delivery	32	3	23	0	31	26	26
pipesworld	41	6	0	6	17	14	12
TOTAL	215	32,0 %	55,7 %	12,3 %	84,7 %	57,7 %	122

Tabla 8.3: h_{add} en CEHC: mejora en calidad utilizando acciones *helpful* (HA).

- En algunos dominios, como *Driverlog*, *Assignment*, *Pipesworld* y algunos problemas de *Depots*, la heurística h_{mff}, que calcula a través de un *RPG* que no contempla costes, ofrece buenas calidades, siendo superior a las demás en tiempo utilizado y en algunos casos en número de nodos evaluados. En estos dominios no considerar los borrados de las acciones supone perder mucha información. El hecho de considerar costes para contruir el plan relajado con información tan sesgada produce resultados que se asemejan a los resultados que se obtienen sin considerar costes. Esto se puede observar más claramente en el dominio *Assignment*, que es el más sencillo de los tres. En este dominio, las heurísticas $h_{level\text{-}max}$, $h_{level\text{-}add}$ y h_{add} son iguales, y se calculan como la suma de distancias resultante de situar cada bola (de forma independiente) en la posición libre cuya distancia a la posición actual de la bola es menor. h_{mff}, es la suma de distancias resultante de situar cada bola en cualquier posición libre. Para el dominio, un hecho relevante es que dos bolas no pueden estar en la misma posición, cosa que ignoran los planes relajados. Con este tipo de planes relajados, puede ser tan impreciso utilizar costes como no utilizarlos. Por otro lado, también influye el hecho de que en un dominio existan pocas formas de conseguir las metas. En este caso, el hecho de considerar costes puede que no aporte mucho.

Aunque las heurísticas con mejor comportamiento general son $h_{level\text{-}add}$, $h_{level\text{-}max}$ y

$h_{add\text{-}HA}$, es complicado determinar cuál de ellas es la más apropiada para un determinado dominio. Primero porque el resultado es dependiente de las acciones *helpful* que se obtengan en cada caso, y segundo porque el comportamiento de las heurísticas depende no sólo de características propias del dominio sino de la distribución de costes propia de cada problema. Estas dos cuestiones se tratan en los dos apartados siguientes. El primero contiene un experimento en los cuatro primeros dominios orientado a evaluar el impacto de utilizar las mismas acciones *helpful* con distintas heurísticas numéricas. En el siguiente, se ilustra el impacto que produce sobre los resultados el hecho de que en el mismo problema del mismo dominio varíe la distribución de costes.

8.1.2.3. Experimento 3: impacto de las acciones *helpful*

Para evaluar el impacto de utilizar distintas acciones *helpful*, en este experimento se introduce una heurística más que se obtiene utilizando los valores numéricos que proporciona $h_{level\text{-}max}$ pero combinados con las acciones *helpful* que se obtienen con $h_{level\text{-}add}$ (la denominaremos $h_{level\text{-}max}$(HA)). Las gráficas de la Figura 8.8 muestran los resultados en los cuatro primeros dominios.

Como se puede observar los resultados tienden a unificarse. Es decir, $h_{level\text{-}max}$ utilizando las acciones *helpful* de $h_{level\text{-}add}$ tiende a proporcionar resultados similares a ésta. Con lo cual, se puede decir que en CEHC en los dominios y problemas utilizandos, la determinación de acciones *helpful* es tanto o incluso más importante que los valores numéricos de la heurística. Esto da a entender que aún teniendo información, las heurísticas numéricas no tienen suficiente calidad como para guiar por sí mismas al algoritmo de búsqueda y, por lo tanto, deben ser complementadas con heurísticas adicionales.

8.1.2.4. Experimento 4: impacto de la distribución de costes

De los experimentos anteriores se podría extraer la conclusión de que en el dominio *Zenotravel* la heurística con mejor comportamiento es $h_{add\text{-}HA}$. Sin embargo, hacer esta generalización al dominio puede ser errónea, dado que el comportamiento depende directamente de la distribución de costes del problema particular. El ejemplo a continuación ilustra el efecto de un cambio en la distribución de costes de un pequeño problema.

La Figura 8.9 muestra un ejemplo de un problema en el dominio *Zenotravel*. Hay cuatro ciudades, una persona y dos aviones. Los valores de los arcos indican la distancia entre las ciudades. El avión 1 gasta tanto combustible como indica esta distancia, mientras que el avión 2 gasta el doble. La métrica que se aplica es la suma del combustible total más el número acciones. El objetivo es que la persona, inicialmente en la ciudad B, se encuentre en la ciudad D.

Las Figuras 8.10 y 8.11 muestran las acciones del plan relajado mediante el extracto del *RPG* del que se obtienen y las acciones *helpful* (en azul) para $h_{level\text{-}max}$ y $h_{level\text{-}add}$ respectivamente. $h_{level\text{-}max}$ sugiere usar el avión 2 para transportar la persona hasta C y luego el avión 1 para transportarla hasta D, mientras que $h_{level\text{-}add}$ únicamente utiliza el avión 2. La estimación en ambos casos es errónea debido al efecto de no tener en cuenta los borrados de las acciones. $h_{level\text{-}add}$ apuesta por el avión 2 porque la estimación del coste

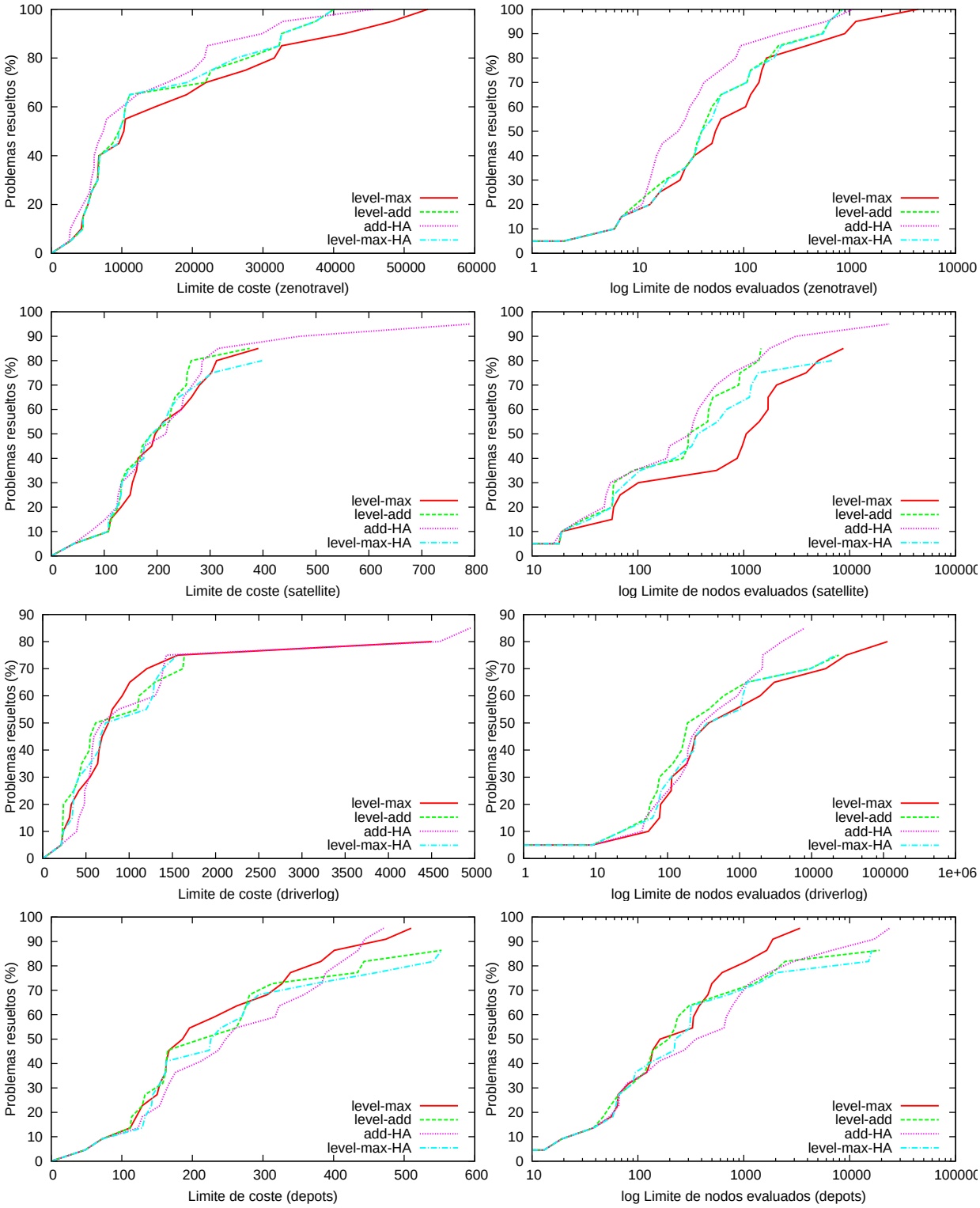

Figura 8.8: Resultados unificando las acciones *helpful*.

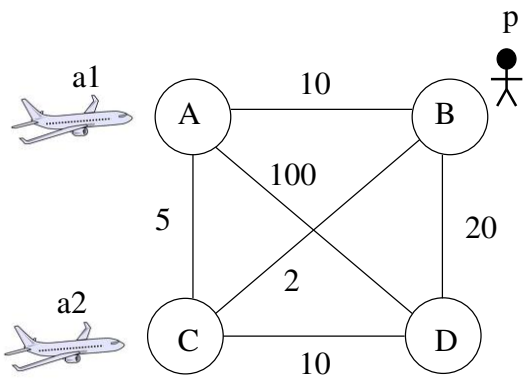

Figura 8.9: Ejemplo de un problema en el dominio *Zenotravel*.

de aplicar (*debark p a1 D*) es más elevada que la de aplicar (*debark p a2 D*), debido a la propagación aditiva.

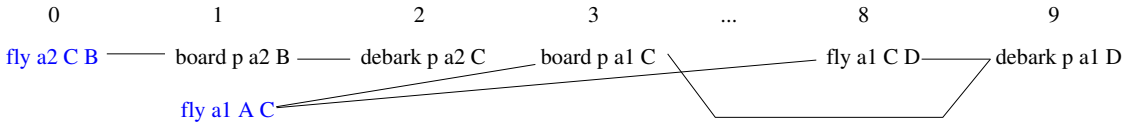

Figura 8.10: Extracto del *RPG* para el estado inicial con $h_{level\text{-}max}$.

Figura 8.11: Extracto del *RPG* para el estado inicial $h_{level\text{-}add}$.

Al resolver el problema con CEHC se obtienen las siguientes calidades:

- Con $h_{level\text{-}max}$ se encuentra un plan de coste 25, que es el óptimo en este problema: *(fly a1 A C), (fly a1 C B), (board p a1 B), (fly 1 B C), (fly a1 C D) (debark p a1 D).*
- Con $h_{level\text{-}add}$ se encuentra un plan de coste 33: *(fly a2 C B), (board p a2 B), (fly a2 B C), (fly a2 C D) (debark p a2 D).*
- Con $h_{add\text{-}HA}$ se encuentra un plan de coste 75: *(fly a2 C D), (fly a2 D C), (fly a2 C B), (board p a2 B), (fly a2 B C), (fly a2 C D) (debark p a2 D)*

Con lo cual, éste es un problema en el que la mejor solución se encuentra con $h_{level\text{-}max}$ y la peor con $h_{add\text{-}HA}$. Resultado contrario a los que se han producido en los experimentos

anteriores. Sin embargo, si la distancia entre A y C fuera 10 en lugar de 5, la heurística $h_{level\text{-}max}$ daría lugar al plan *(fly a2 C B), (fly a1 A C), (fly a1 C B), (board p a1 B), (fly a1 B C), (fly a1 C D) (debark p a1 D)* con coste 35, mientras que $h_{level\text{-}add}$ y $h_{add\text{-}HA}$ producen el mismo plan en este caso, que coincide con el que antes producía $h_{level\text{-}add}$ de coste 33.

Por lo tanto, el mero hecho de cambiar un coste en el mismo problema es capaz de dar la vuelta a los resultados, así que es prácticamente imposible determinar que heurística ofrecerá un comportamiento mejor sin analizar el problema concreto y su distribución de costes. Para demostrar esto se ha realizado un pequeño experimento en el dominio *Zenotravel*, tomando un problema (el problema 7) y generando 1000 instancias de este problema. Entre unas instancias y otras, lo único que varía es la distribución de costes. En cada instancia el coste de cada acción con coste mayor que cero (es decir de las acciones *fly* y *zoom*) se ha generado aleatoriamente (tomando valores entre 0 y 3000). A continuación, cada instancia se ha solucionado utilizando CEHC con las heurísticas $h_{level\text{-}max}$, $h_{level\text{-}add}$ y $h_{add\text{-}HA}$.

La Figura 8.12 muestra el contraste (por pares de heurísticas) entre la calidad de los planes encontrados para cada instancia. Cada punto de las gráficas corresponde a una instancia. Los ejes de coordenadas indican el coste del plan obtenido para cada instancia (punto) utilizando la heurística situada en el propio eje. La diagonal en cada caso se corresponde con instancias para las cuales ambas heurísticas obtienen el mismo coste. Los valores por encima y debajo de la diagonal corresponden con instancias para las que la heurística en el eje X obtiene planes con coste menor y mayor que la heurística en el eje Y respectivamente. La Figura 8.13 muestra las gráficas equivalentes para el número de estados evaluados.

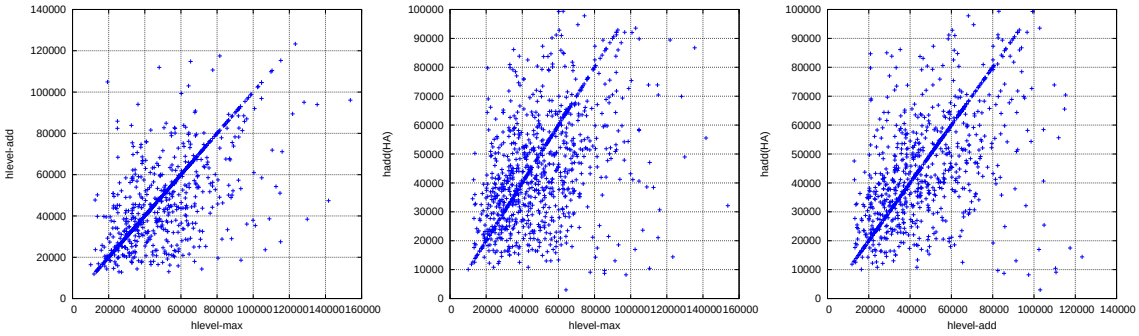

Figura 8.12: Contraste entre el coste de los planes con distribuciones de coste aleatorias.

Como se puede observar, en las tres gráficas de la Figura 8.12 existe un número de casos considerable en los que cada heurística encuentra planes con coste menor y mayor que las demás. Lo mismo ocurre respecto al número de nodos evaluados. Por lo tanto, si este mismo efecto ocurriera frecuentemente en muchos problemas de distintos dominios, se podría afirmar que el hecho de que una heurística tenga mejor comportamiento que otra depende directamente de la distribución de costes del problema, lo que hace prácticamente imposible obtener conclusiones generales relativas a características propias del dominio.

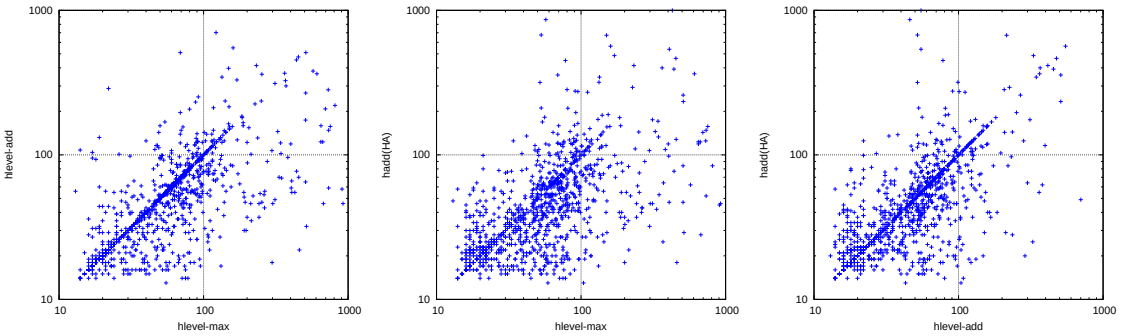

Figura 8.13: Contraste entre el número de nodos evaluados con distribuciones de coste aleatorias.

8.1.2.5. Experimento 5: variación a CEHC

El objetivo de este experimento es contrastar el comportamiento de CEHC con la variación propuesta, en la que el primer nivel de sucesores de cada nodo se ordena en orden creciente de $h(s) + coste(a)$, donde a es la acción que genera el sucesor. Se han realizado experimentos utilizando los dos esquemas de búsqueda con las heurísticas que ofrecen mejor comportamiento en CEHC: $h_{level\text{-}add}$, $h_{level\text{-}add}$, y $h_{add\text{-}HA}$. Las variables dependientes en este caso serán la calidad de los planes encontrados y el tiempo de CPU. Las Figuras 8.14 y 8.15 muestran los resultados. En estas gráficas, los casos etiquetados con (1) se refieren a la aproximación inicial y los etiquetados con (2) a la variación.

Para cada heurística, los resultados muestran que en general tanto el tiempo en conseguir la solución como la calidad empeoran utilizando la variación. Esto ocurre con más claridad en los dominios *Zenotravel*, *Satellite*, *Depots*, *Assignment*, y *MST*. En *Driverlog* y *Delivery* los resultados son más similares, mientras que *Pipesworld* es el único dominio donde hay resultados en que la variación ofrece un mejor comportamiento. Por ejemplo en los problemas más complejos que se resuelven con $h_{add\text{-}HA}$, la variación produce mejores calidades y resuelve algún problema más. Sin embargo el número de problemas resueltos en este dominio es pequeño (poco más del 45 %).

Es normal que el tiempo en encontrar la solución empeore ya que con la variación se evalúan todos los sucesores del primer nivel cada vez que se elige un nodo por el que continuar la búsqueda. El hecho de que la calidad también empeore indica que es aparentemente mejor estrategia ordenar los sucesores por el coste de la acción que los genera que ordenarlos según el coste de esta acción más la evaluación heurística. Esto puede ser un efecto de la no admisibilidad de las heurísticas, o dicho de otro modo, de que las heurísticas sobrestimen en algunas ocasiones y subestimen en otras. Cuando la heurística subestima se puede seleccionar un sucesor aunque la acción que lo genera tenga coste alto. Cuando sobreestima puede que se descarte un sucesor generado por una acción de bajo coste.

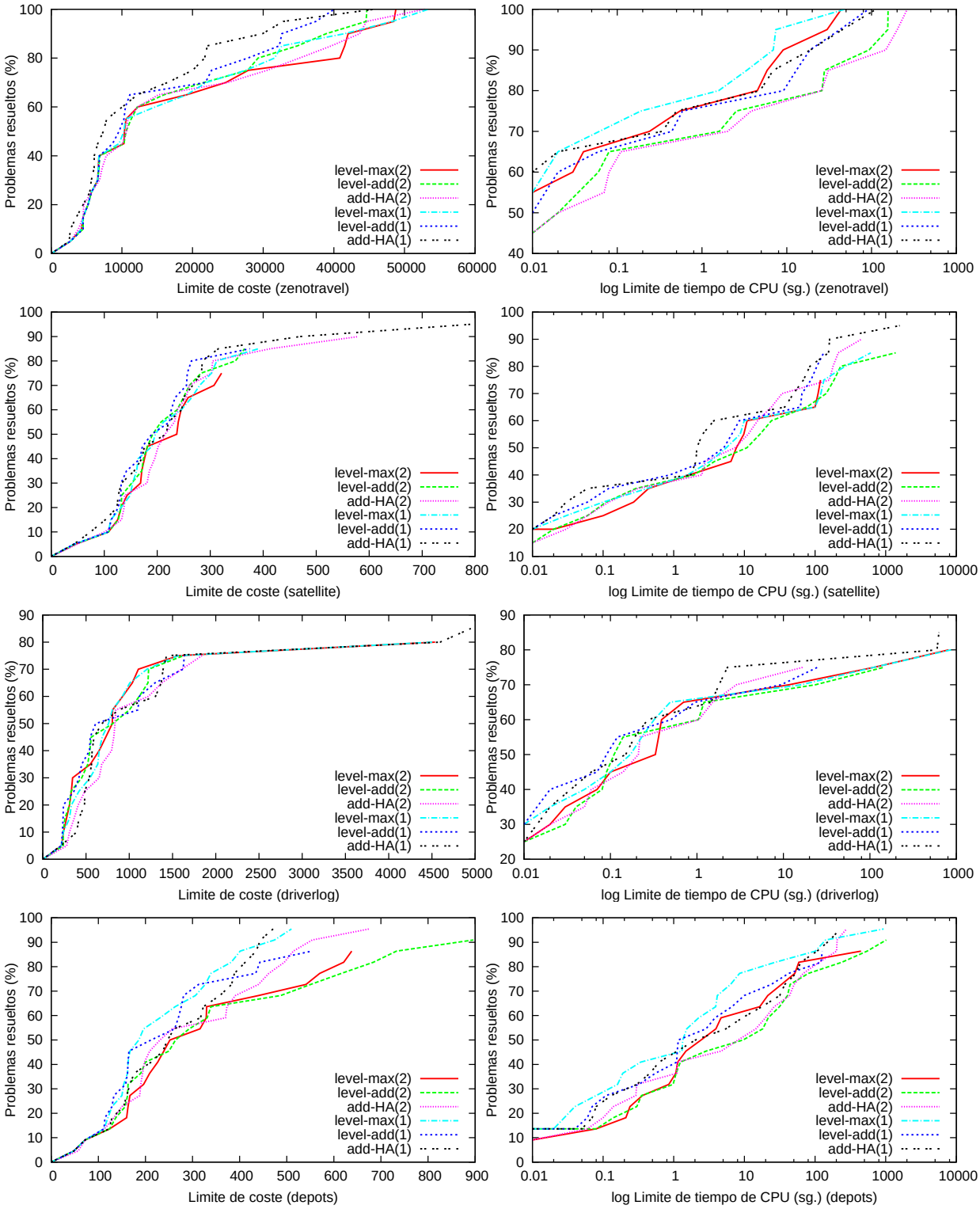

Figura 8.14: Resultados incluyendo la variación propuesta en CEHC (I).

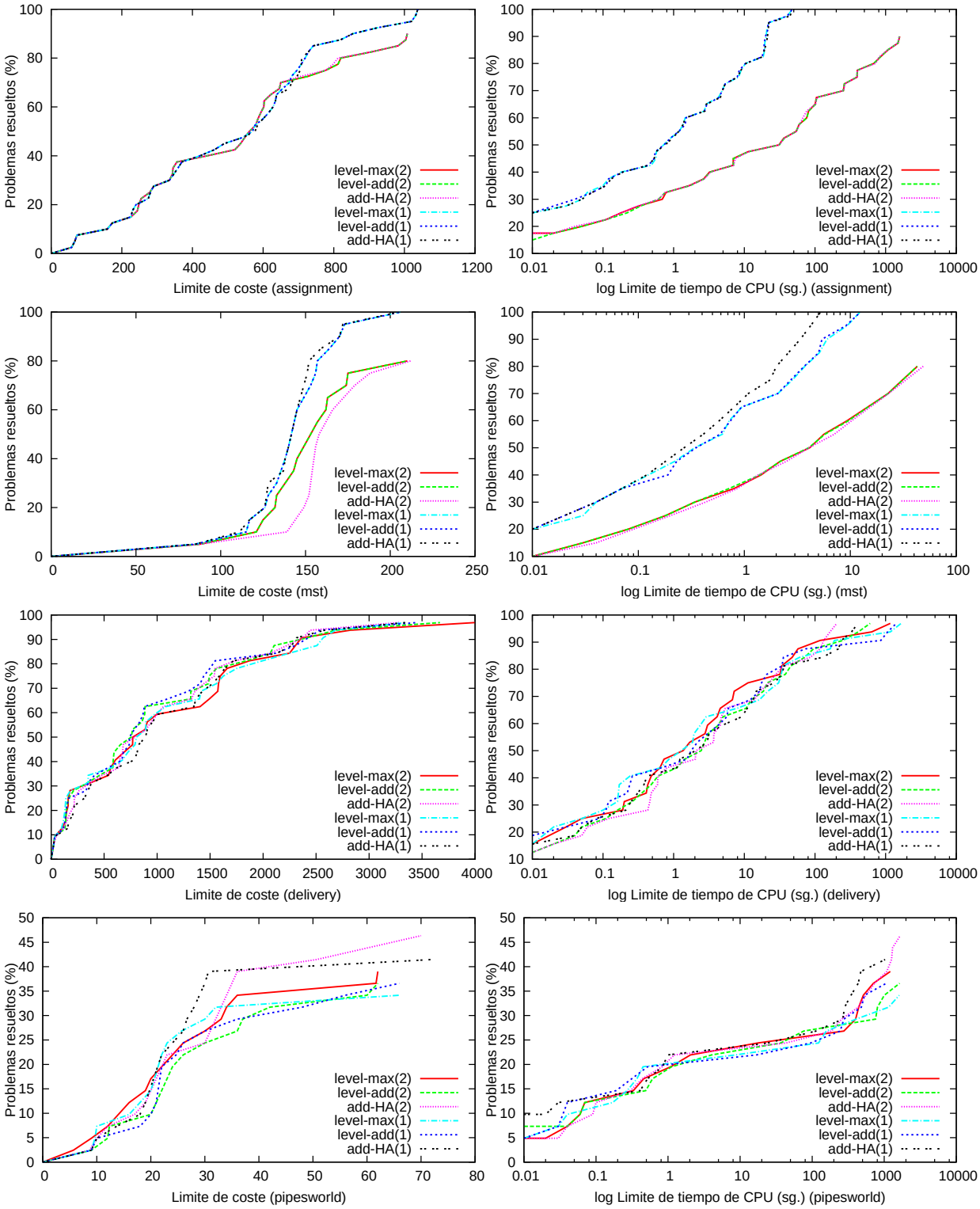

Figura 8.15: Resultados incluyendo la variación propuesta en CEHC (II).

8.2. Hill-Climbing con *backtracking* (HC-B)

En este apartado se plantea utilizar un algoritmo búsqueda en escalada, modificado en algunos aspectos respecto a la descripción original del algoritmo *Hill-Climbing* (Russell and Norvig, 2002). La idea del algoritmo *Hill-Climbing* es pasar siempre a estados con una evaluación heurística mejor que el padre. Sin embargo, esto puede hacer que el algoritmo falle cuando ninguno de los sucesores de nodo padre tenga una evaluación heurística mejor. En este caso, se podría establecer algún mecanismo para continuar la búsqueda. Una opción es la que planteaba CEHC: continuar la búsqueda en niveles posteriores hasta encontrar un nodo que mejore la evaluación del nodo raíz actual. La posibilidad que se plantea en este apartado es eliminar la restricción de que el nodo elegido tenga mejor evaluación que el padre, y simplemente elegir el nodo con mejor evaluación heurística de entre todos los sucesores. En las heurísticas con las que se está trabajando se puede dar el caso de que un plan solución (incluso el plan óptimo) se encuentre mediante un nodo que tiene peor evaluación que el padre.

Al igual que CEHC, en dominios con estados sin salida la búsqueda que se plantea fallará sino se establece un mecanismo para continuar la búsqueda. En este caso se plantea mantener en memoria los estados por explorar (con lo que se pierde la idea de la búsqueda local) y permitir que el algoritmo realice un *backtracking* cronológico.

Por otro lado, los experimentos realizados hasta el momento demuestran que las acciones *helpful* son útiles en planificación basada en costes y además restringen considerablemente el número de sucesores de cada nodo. Puesto que en el esquema que se plantea es necesario evaluar todos los sucesores para poder seleccionar uno de ellos, para restringir el número de evaluaciones sólo se considerarán los sucesores generados por acciones *helpful*. Así el espacio de estados queda limitado a aquéllos que se generan con acciones *helpful*, y por lo tanto la búsqueda no es completa.

La Figura 8.16 muestra el pseudocódigo del algoritmo. Inicialmente, el estado actual es el estado inicial. Éste se evalúa y a continuación se generan y evalúan los sucesores a los que dan lugar las acciones *helpful*. Estos sucesores se ordenan en orden creciente de la suma del valor heurístico más el coste de la acción que los genera. Con la ordenación resultante se introducen al principio de la lista ABIERTA. A continuación se extrae el primer nodo de esta lista, incluyendo en el plan la acción que lo genera y se repite el proceso, tomando este nodo como nuevo estado actual. El algoritmo termina cuando la evaluación en términos de operadores del estado actual es cero o bien cuando la lista ABIERTA se queda vacía.

Una posible variación al algoritmo consiste en ordenar los sucesores utilizando exclusivamente su valor heurístico sin realizar la suma con el coste de la acción que los genera. Denominaremos a esta variación HC2, mientras que la anterior será HC1.

```
function HC-B(I,G)
let plan = ∅;
let ABIERTA = I;
let estado_actual = pop(ABIERTA)
evaluar(estado_actual)
while h_ops(estado_actual) ≠ 0 do /* estado_actual no contiene todas las metas en G */
    sucesores = sucesores_helpful(estado_actual)
    forall sucesor ∈ sucesores do
        evaluar(sucesor)
    sucesores_ordenados=ordenar(sucesores) /*orden creciente de h_cost(s) + coste(a) */
    ABIERTA = sucesores_ordenados + ABIERTA
    if ABIERTA do
        estado_actual = pop(ABIERTA)
        plan = plan + {a} /* a: acción que genera estado_actual */
    else return fail
return plan
```

Figura 8.16: Búsqueda *Hill-Climbing* con *backtracking* cronológico.

8.3. Resultados experimentales (II). Experimento 6: HC-B vs. CEHC

Se ha realizado el siguiente experimento con el objetivo de contrastar el comportamiento de HC-B en sus dos variantes, HC1 y HC2, con el comportamiento de CEHC. Se han ejecutado los algoritmos HC1, HC2 y CEHC con las heurísticas $h_{level\text{-}add}$ y $h_{level\text{-}max}$. Las variables dependientes del experimento son el coste de los planes encontrados, el número de nodos evaluados, el tiempo total de CPU y el número de problemas resueltos. Las Figuras 8.17 y 8.18 muestran los resultados relativos al coste de la solución y al número de nodos evaluados. Los demás resultados se encuentran en el apéndice A.

A la vista de los resultados se puede afirmar que por lo general, para la misma heurística, ni HC1 ni HC2 son algoritmos que permitan encontrar planes de más calidad que CEHC, aunque hay alguna excepción. Por ejemplo, en *Zenotravel*, aunque la diferencia no es demasiado significativa, se obtienen por lo general mejores resultados en calidad utilizando HC2 para las dos heurísticas. En este dominio, el número de nodos explorados es por lo general menor utilizando CEHC.

En los demás dominios las soluciones de más calidad se obtienen utilizando CEHC. Aunque hay algunos puntos que destacar en relación con las otras variables:

- En *Driverlog*, el comportamiento de la heurística $h_{level\text{-}max}$ es especialmente malo tanto con HC1 como con HC2. Sin embargo. esto no es así utilizando CEHC. En este dominio, con CEHC se obtiene también un menor número de estados evaluados y se encuentra la solución en menos tiempo.

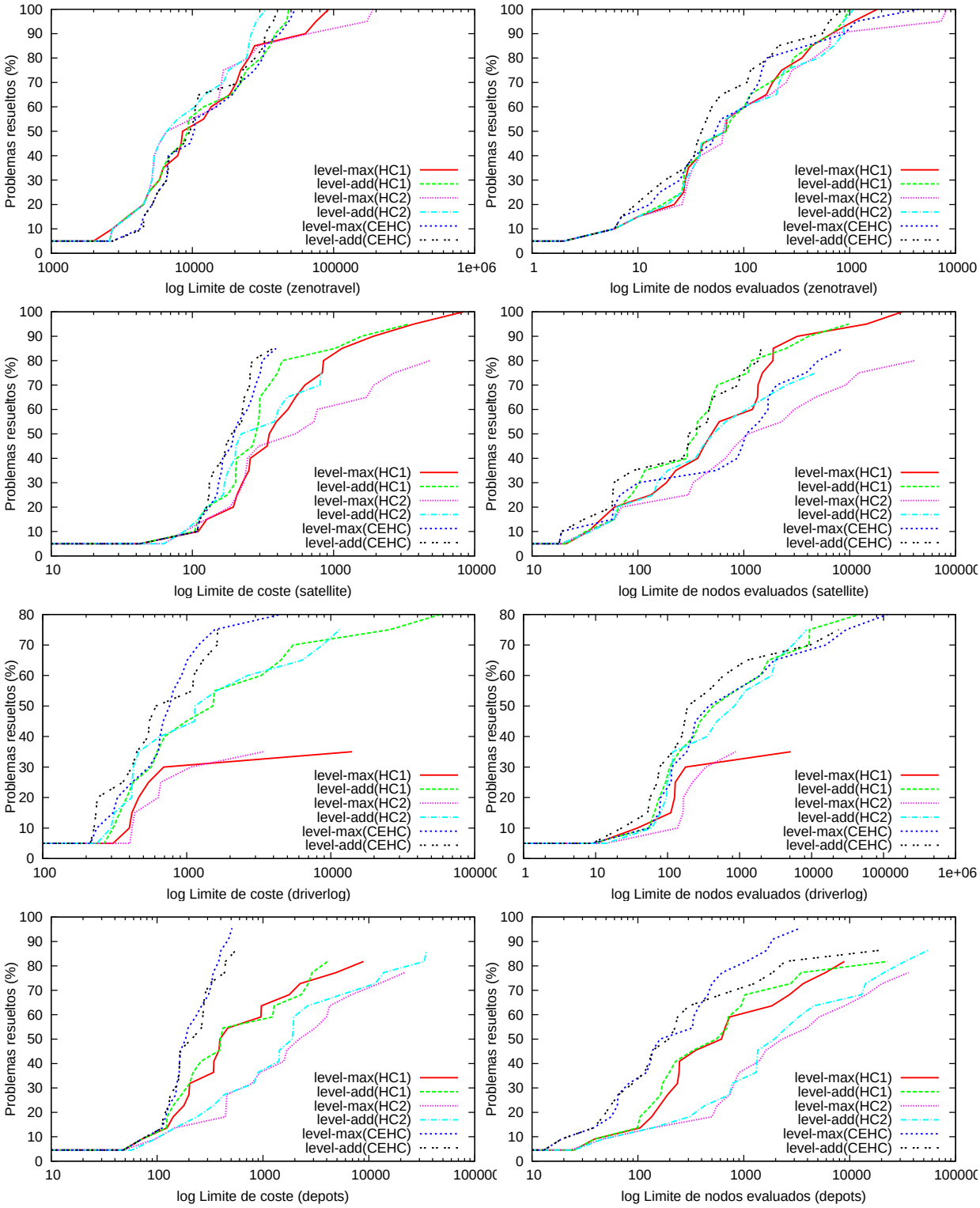

Figura 8.17: Resultados de la comparación entre HC y CEHC (I).

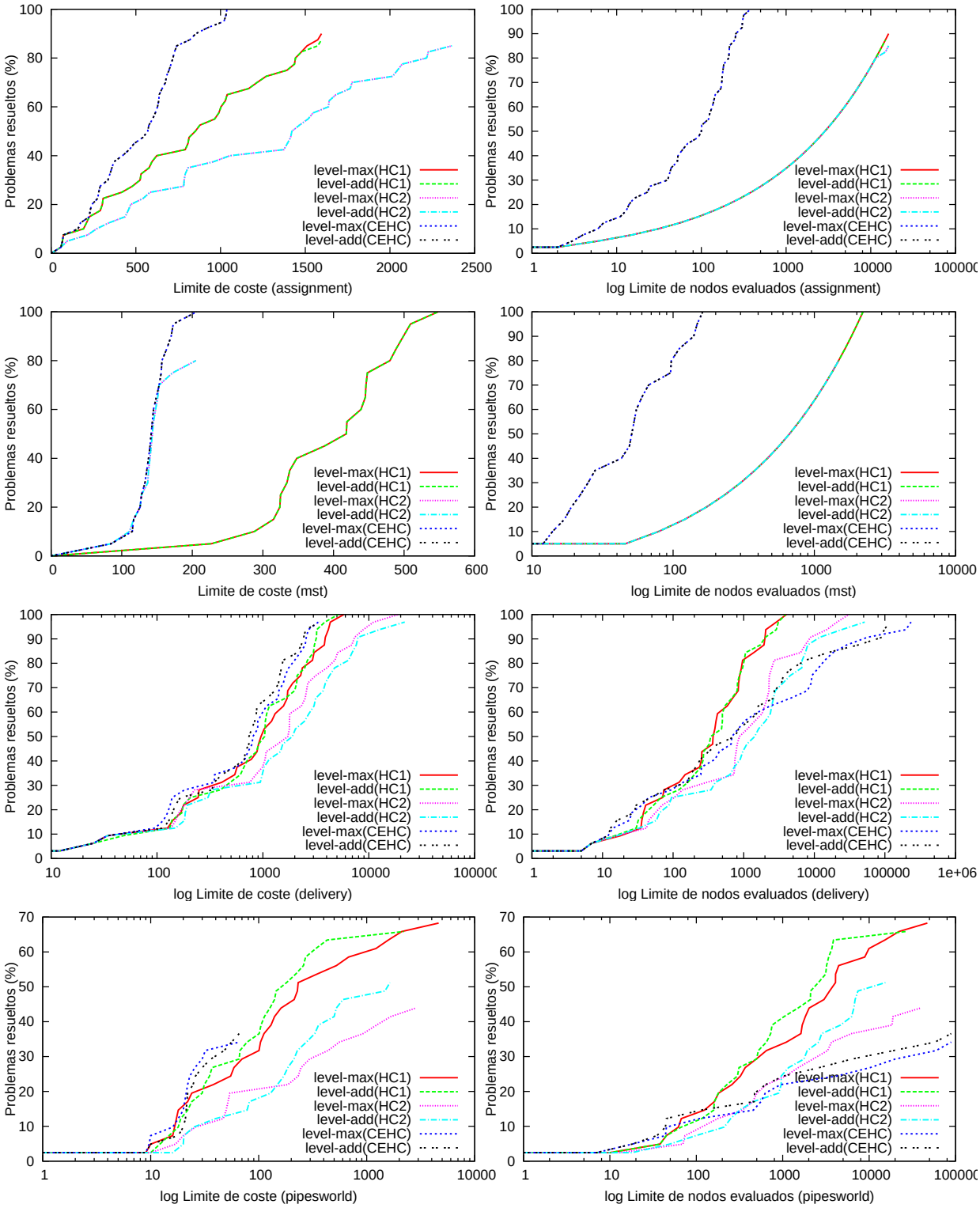

Figura 8.18: Resultados de la comparación entre HC y CEHC (II).

- En *Depots*, *Assignment* y *MST*, CEHC es la mejor aproximación tanto en calidad como en nodos evaluados y tiempo. En los dos últimos dominios el número de nodos evaluados es igual con HC1 y HC2. En *MST* se encuentran prácticamente las mismas calidades que con HC2 que con CEHC, aunque con HC2 se decrementa en un 20 % el número de problemas resueltos.
- En *Delivery*, CEHC obtiene las mejores calidades, pero la aproximación que menos nodos evalúa y menos tiempo utiliza es HC1.
- En *Pipesworld*, HC1 resuelve casi un 70 % de los problemas mientras que HC2 resuelve aproximadamente un 50 %, y CEHC un 35 %.

El hecho de que el número de nodos evaluados con CEHC sea menor que con las aproximaciones HC es esperable, siempre y cuando CEHC no tenga que expandir muchos niveles de búsqueda en amplitud, en cuyo caso pueden ser más rápidas las búsquedas HC, o CEHC encuentre un estado sin salida, en cuyo caso fallará. HC evalúa todos los sucesores *helpful* de cada nodo, mientras que CEHC sólo evalúa hasta encontrar el primer sucesor con heurística menor.

8.4. Resumen

En este capítulo se han presentado dos algoritmos basados en esquemas de búsqueda local: CEHC y HC-B. También se han propuesto algunas variaciones sobre los mismos: en CEHC la variación consiste en ordenar el primer nivel de sucesores teniendo en cuenta el valor heurístico y el coste de la acción que genera el sucesor; en HC, en hacer la ordenación de los sucesores de cada nodo, sumando y sin sumar el coste de la acción que genera el sucesor al valor heurístico, HC1 y HC2 respectivamente.

Tomando estos algoritmos y sus variaciones se han presentado experimentos con distintos objetivos, de los cuales se pueden extraer las siguientes conclusiones:

- Respecto a las acciones *helpful*, es aparentemente más adecuado utilizar el esquema de acciones helpful HA2. HA2 es más restrictivo que HA1, y por lo tanto reduce más el número de sucesores de cada nodo.
- En CEHC, el conjunto de acciones *helpful* obtenido mediante *RPGs* que propagan costes, es útil no solo para mejorar la escalabilidad, sino también para conseguir planes de más calidad.
- Respecto a las heurísticas incluidas en los experimentos:
  - Las que proporcionan mejores resultados con CEHC son $h_{level\text{-}max}$, $h_{level\text{-}add}$, y $h_{add\text{-}HA}$ (h_{add} utilizando las acciones *helpful* de $h_{level\text{-}add}$).
  - Utilizar las mismas acciones *helpful* para las heurísticas $h_{level\text{-}add}$ y $h_{level\text{-}max}$ tiende a unificar sus resultados, lo cual indica que el resultado de CEHC tiene gran dependencia del conjunto de acciones *helpful* y de su ordenación, y que las heurísticas numéricas no tienen calidad suficiente por si solas para guiar la búsqueda.

  - Sólo se podría determinar cuál de estas heurísticas es más adecuada en un determinado dominio, en base a características propias del dominio, si los cambios en la distribución de los costes de las acciones no afectaran al comportamiento de las heurísticas. Es decir, si el hecho de que una heurística se comporte mejor que otra es independiente de la distribución de costes del problema concreto. En otro, caso habría que caracterizar no solo el dominio sino también las distintas distribuciones de costes posibles. Por este motivo es muy difícil determinar, dado un dominio, qué heurística se debe elegir entre $h_{level\text{-}max}$, $h_{level\text{-}add}$, y h_{add}.

- Respecto a los distintos algoritmos, dada la misma heurística:

  - En CEHC, la variación que consiste en elegir en el nivel de sucesores directos el nodo con menor $coste(a) + h(s)$, siempre y cuando este nodo tenga mejor evaluación heurística que el padre por lo general empeora los resultados tanto en calidad como en tiempo que obtiene el algoritmo CEHC planteado inicialmente.
  - Las aproximaciones derivadas de *Hill-Climbing* tampoco mejoran la calidad de los planes encontrados respecto a CEHC, aunque en algunos casos hay mejoras respecto al número de problemas resueltos.

Aunque es justificable el uso de esquemas derivados de búsqueda local desde el punto de vista de la escalabilidad, es menos intuitivo utilizarlos para optimización. Sin embargo, es necesario que la escalabilidad sea suficientemente buena para poder plantearse el problema de optimización. En este capítulo, se ha planteado el uso de esquemas derivados de búsqueda local con la siguiente idea intuitiva: ser avaro para conseguir buena escalabilidad, pero consciente en cierta medida de que las acciones tienen coste. En el siguiente capítulo se analiza la aplicación de esquemas derivados de algoritmos *mejor primero*, más orientados desde el punto de vista intuitivo, a tareas de optimización.

Capítulo 9

Esquemas derivados de algoritmos *mejor primero*

Los algoritmos de tipo *mejor primero* (o *Best-First Search, BFS*) siguen un esquema de búsqueda global, que se diferencia de los esquemas de búsqueda local principalmente en los siguientes aspectos: (1) mantienen todos los nodos por explorar (es decir, si no se realizan podas adicionales, son completos), (2) estos nodos están ordenados globalmente, y (3) pueden utilizar una función de evaluación que tenga en cuenta el coste desde el nodo inicial a cada nodo, $g(n)$. Estas características hacen que sean aparentemente algoritmos más orientados a optimizar la solución que los esquemas basados en búsqueda local. A cambio los esquemas de búsqueda local suelen emplear menos tiempo y consumir menos memoria (lineal vs. exponencial), excepto en los casos en que no se emplea el algoritmo en su forma más pura, como en CEHC, que a veces puede degenerar en una búsqueda en amplitud.

En este capítulo, se parte de un búsqueda *mejor primero* y se proponen algunas variaciones sobre el mismo, con el objetivo de que pueda ser utilizado con éxito en planificación basada en costes. Se presentan también distintos tipos de experimentos, orientados a contrastar las variaciones y comparar los comportamientos obtenidos con algunos esquemas de búsqueda local del capítulo anterior.

9.1. Branch and Bound Best-First Search (BB-BFS)

Muchas de las heurísticas que se utilizan en planificación basada en costes son no admisibles e inconsistentes (o al menos lo contrario no se puede demostrar teóricamente y de forma general para todos los dominios). Sin embargo desde el punto de vista empírico, muchas veces subestiman el coste óptimo. Esto es debido a que la mayoría de las heurísticas ignoran total o parcialmente los efectos negativos de las acciones. La subestimación es más acentuada en los dominios en que los efectos negativos son más relevantes (es decir, en dominios con interacciones complejas entre (sub)metas). Por estas razones cuando en planificación automática se utiliza un algoritmo mejor primero con estas heurísticas, la solución

encontrada es por lo general subóptima. Por otro lado, con este tipo de heurísticas no aplican las propiedades teóricas de las heurísticas admisibles para determinar si una heurística es más informada que otra. La intuición, sin embargo, indica que heurísticas cuyo valor se aproxima más al óptimo (tanto por encima como por debajo) deberían proporcionar mejores resultados, puesto que tienen más precisión.

La aplicación de un algoritmo *mejor primero* puro (*Best-First Search*, BFS) con heurísticas independientes del dominio en planificación automática presenta serios problemas de escalabilidad. Las causas de estos problemas son principalmente tres: el gran tamaño del espacio de estados, la precisión de las heurísticas que se utilizan, el tiempo que se utiliza en realizar evaluaciones heurísticas.

El tiempo de cálculo de las funciones heurísticas que se aplican es polinomial en relación con el tamaño de los estados y el número de acciones del dominio. Sin embargo, el número de evaluaciones que BFS realiza con las heurísticas existentes es elevado, típicamente exponencial. En definitiva, una cantidad considerable del tiempo de búsqueda se utiliza en realizar evaluaciones heurísticas. Por estas causa, ante problemas de complejidad media BFS agota el límite de tiempo antes de haber encontrado solución alguna al problema. En la mayoría de los casos el fallo se produce por falta de tiempo y no de memoria.

Aunque estos problemas se han identificado en planificación clásica, es esperable que en planificación basada en costes se acentúen, dado que el rango del error que cometen las heurísticas en este caso puede ser mucho mayor. Es este capítulo se parte de un esquema de búsqueda BFS y se analizan distintas variaciones sobre el mismo orientadas a: encontrar soluciones de menor coste (o mayor calidad), y mejorar la escalabilidad del algoritmo sin penalizar demasiado en coste.

Existen varias opciones para orientar al algoritmo BFS hacia soluciones menos costosas. Una de ellas, común también a los demás algoritmos, es mejorar la precisión de la heurística numérica que se utiliza. Para mejorar la precisión, un primer paso, que se ha tratado en la segunda parte de la tesis, es calcular la heurística mediante un proceso que razone sobre costes. Otras posibilidades van desde incorporar en las heurísticas más información, lo que en principio supone incluir conocimiento acerca de la información de la que carecen: las interacciones positivas y negativas entre (sub)metas; hasta combinar de una forma adecuada diferentes heurísticas. En cualquier caso, más información suele suponer más tiempo de cálculo de la heurística, aunque si la información es adecuada lo normal es que resulte en algoritmos más rápidos. Otra opción es seguir

buscando otras soluciones. Aunque las dos direcciones son complementarias, cuando el tiempo es limitado, hay que decidir en si emplearlo en calcular una heurística mejor o en hacer más búsqueda (información vs. exploración). En este apartado se analiza el segundo caso, con una aproximación sencilla: permitir al algoritmo que siga buscando hasta un tiempo límite. De esta forma, el usuario puede decidir si se conforma con la primera solución encontrada, o por el contrario desea emplear más tiempo con el objetivo de obtener una solución de más calidad.

Suponiendo problemas resolubles, la exploración sólo tiene sentido si el algoritmo es

capaz encontrar alguna solución. Por lo tanto, sólo tiene sentido si se combina con estrategias para mejorar la escalabilidad. La escalabilidad se puede mejorar tanto dotando a las heurísticas numéricas de más información, como utilizando otras heurísticas y estrategias que ayuden al algoritmo de búsqueda, que por lo general pasan por incluir alguna estrategia que lo haga más avaro.

La aproximación que se plantea en este apartado consiste en continuar la búsqueda hasta un límite de tiempo fijado (*límite_tiempo*) en lugar de parar en la primera solución. El objetivo es encontrar una secuencia de soluciones de manera que cada una siempre mejore el coste de la anterior. Para esto, se utiliza una estrategia *Branch and Bound* en la que cada vez que se encuentra una solución, el coste de la misma (*límite_coste*) se utiliza para podar el espacio de estados en la búsqueda a continuación. Recientemente, han surgido algunos trabajos que también aplican esta estrategia para conseguir un comportamiento *anytime* (Hansen and Zhou, 2007; Benton et al., 2007).

Existen dos opciones para podar el espacio de estados:

- Podar todos los estados con $g(n)$ mayor o igual al coste de la última solución encontrada.
- Podar todos los estados con $f(n) = \omega \times h(n) + g(n)$ mayor o igual al coste de la última solución encontrada.

Aunque ambas opciones habría que valorarlas experimentalmente, con heurísticas no admisibles podar por $f(n)$ implica perder la completitud del algoritmo. La completitud es una propiedad deseable, siempre y cuando la carga computacional no sea tan elevada como para no encontrar ninguna solución, por lo que elegiremos podar por $g(n)$.

Respecto a la escalabilidad del algoritmo, también puede ser importante evitar las reevaluaciones de estados que se vuelven a abrir. El hecho de utilizar heurísticas inconsistentes puede dar lugar a que aparezcan estados con menor $g(n)$ que los ya expandidos. Así, cuando aparece un estado repetido n_2, que tiene los mismos hechos que un estado ya expandido, n_1, si $g(n_2) \geq g(n_1)$, n_2 se poda. En caso contrario, n_2 se incluye en la lista ABIERTA sin volver a evaluarlo y tomando la evaluación heurística de n_1.

Otra posibilidad, que también puede mejorar la escalabilidad del algoritmo, es detectar si un nodo es meta en el momento de la generación en lugar de en la expansión. Sin embargo, esto puede afectar negativamente a la calidad de la solución. En este sentido se han realizado algunas pruebas y parece más adecuado realizar la comprobación de nodo meta en el momento de la expansión, aunque habría que llevar a cabo una experimentación más exhaustiva.

La Figura 9.1 muestra el algoritmo *Brach and Bound Best First Search, (BB-BFS)*. Inicialmente la lista *ABIERTA* contiene el estado inicial y la lista CERRADA está vacía. El límite de coste, *límite_coste*, es infinito; y el conjunto de planes que constituirá la salida del algoritmo está vacío. A continuación el algoritmo entra en un bucle que sólo termina en caso de que la lista ABIERTA se quede vacía o se alcance el límite de tiempo (*límite_tiempo*). En cada iteración del bucle se extrae de la lista ABIERTA el nodo con menor valor para

$f(x)$ y se expande (se introduce en la lista CERRADA). Este nodo es el estado_actual. Si el estado actual es un nodo meta, se extrae el camino hasta él desde el nodo inicial (el plan), que constituye una nueva solución. Este nuevo plan se introduce en la lista de planes, y a continuación se fija su coste como límite de coste para llevar a cabo el *Branch and Bound*. En caso de que el estado actual no sea un nodo meta, y sólo si su valor de $g(n)$ es menor que el límite de coste, se generan sus sucesores. Por cada sucesor se pueden dar tres casos:

- Que el sucesor ya esté generado (es decir, que se encuentre en la lista ABIERTA). En este caso, el sucesor sólo se tiene en cuenta si su valor $g(sucesor)$ es menor que el valor g del nodo ya generado. Cuando esto ocurre, el nodo ya generado actualiza su valor de g a $g(sucesor)$ (puesto que se ha encontrado un camino de menor coste para llegar a él). Después actualiza su valor de $f(n)$, tomando la nueva g y sin necesidad de reevaluar. Esto supone que el nodo se reordene en ABIERTA según su nueva $f(n)$.
- Que el sucesor ya esté expandido (es decir, que se encuentre en la lista CERRADA). En este caso, cuando el valor g del sucesor es menor que el valor g del nodo expandido, el nodo se elimina de CERRADA y se vuelve a introducir en ABIERTA sin necesidad de reevaluación, pero actualizando su función de evaluación a $\omega \times h(expandido) + g(sucesor)$. Esto permite que el nuevo valor de $f(n)$ se propague a los sucesores.
- Que el sucesor sea un nuevo nodo (ni generado ni expandido), en cuyo caso se evalúa y se introduce en ABIERTA.

9.2. Variaciones a BB-BFS

En este apartado se analizan varias opciones para mejorar la escalabilidad de BFS sin penalizar demasiado la calidad de las soluciones encontradas. Principalmente se trasladan a planificación basada en costes dos heurísticas no numéricas que se han aplicado con éxito en planificación clásica: la consideración de acciones *helpful* y el uso de estados *lookahead*. Tanto las acciones *helpful* como los estados *lookahead* se extraen del mismo proceso, sensible a costes, de cálculo de la heurística. Por este motivo cuentan con cierta información relacionada con la calidad, por lo que su uso no debería empeorarla demasiado. Por otro lado, los experimentos realizados con el algoritmo CEHC, demuestran que en determinados dominios las acciones *helpful* obtenidas de *RPGs* sensibles a costes son útiles para encontrar planes de calidad.

9.2.1. Prioridad absoluta a *helpful actions* (AHA)

La incorporación de acciones *helpful* a algoritmos BFS se ha planteado en planificación clásica de dos maneras: (1) la que se planteó para el planificador YAHSP (estado de la cuestión, página 39) , y (2) la que se planteó para el planificador FAST-DOWNWARD (estado de la cuestión, página 38). En ambos casos la idea es dar a las acciones *helpful* una prioridad relativa respecto a las que no lo son. Otra posibilidad, que se explica a continuación, es que las acciones *helpful* sean prioritarias de forma absoluta. Las aproximaciones existentes son más robustas en todos los dominios, mientras que ésta última es más adecuada a los dominios en los que las acciones *helpful* están bien informadas.

```
function BB-BFS(I, G)
let ABIERTA = I;
let CERRADA = ∅;
let límite_coste = ∞;
let planes = ∅;
while ABIERTA ≠ ∅ and not límite_tiempo do
    estado_actual ← arg min_{x∈ABIERTA} f(x)
    ABIERTA ← ABIERTA \ {estado_actual}
    CERRADA ← CERRADA ∪ {estado_actual}
    if nodo_meta(estado_actual, G) then
        plan ← extraer_plan(estado_actual)
        planes ← planes ∪ {plan}
        límite_coste ← coste(plan)
    else if g(estado_actual) < límite_coste then
        forall sucesor ∈ sucesores(estado_actual) do
            if sucesor = generado ∈ ABIERTA
                if g(sucesor) < g(generado) then
                    g(generado) ← g(sucesor)
                    f(generado) ← ω × h(generado) + g(sucesor)
            else if sucesor = expandido ∈ CERRADA
                if g(sucesor) < g(expandido) then
                    CERRADA ← CERRADA \ {expandido}
                    f(sucesor) ← ω × h(expandido) + g(sucesor)
                    ABIERTA ← ABIERTA ∪ {sucesor}
            else
                f(sucesor) ← ω × h(sucesor) + g(sucesor)
                ABIERTA ← ABIERTA ∪ {sucesor}
return planes
```

Figura 9.1: Búsqueda *Branch and Bound Best-First*.

Una forma de priorizar de forma absoluta las acciones que son *helpful* frente aquéllas que no lo son es utilizar dos listas abiertas: una lista ABIERTA con los sucesores generados por acciones *helpful* y otra lista SECUNDARIA con los sucesores generados por las demás acciones. El contenido de la lista SECUNDARIA se vuelca en la lista ABIERTA, sólo si ésta se queda eventualmente vacía. De esta forma se mantiene la completitud del algoritmo, aunque cuando el proceso de búsqueda es muy pesado es posible que el paso de los nodos de la lista secundaria a la lista abierta nunca suceda.

Los estados de la lista SECUNDARIA no se evalúan sino que su evaluación se retrasa hasta que son introducidos en ABIERTA. Este retraso en la evaluación de estados implica un ahorro considerable de tiempo respecto a las opciones con prioridad relativa en caso de que se puedan encontrar soluciones utilizando acciones *helpful*. En caso contrario, es necesario explorar todo el espacio de estados que viene dado por acciones *helpful* para que el algoritmo aplique acciones que no lo son. Por lo tanto el resultado dependerá directamente de la bondad de las acciones *helpful* en el dominio en que se aplique este algoritmo.

Por otro lado, cuando un nodo en la lista SECUNDARIA se va a introducir en la lista ABIERTA, se realiza el mismo tratamiento que en el caso de los sucesores en el algoritmo de la Figura 9.1. Es decir, se tiene en cuenta si el estado ya está generado o expandido para no reevaluar. Cuando se da alguno de estos casos, si el coste g del estado en la lista secundaria es mayor que el coste g del existente, el estado de la lista secundaria se poda. El algoritmo para dar prioridad absoluta a las acciones *helpful* (AHA), que incluye lo explicado en este apartado se muestra en la Figura 9.2. En esta figura se han marcado en rojo las modificaciones respecto al algoritmo BB-BFS de la Figura 9.1.

9.2.2. Incorporación de estados *lookahead*

La incorporación de estados *lookahead* en el algoritmo se puede llevar a cabo de diversas formas. En este apartado se plantea una de ellas. Principalmente hay que tener en cuenta, que los estados *lookahead*, al ser sucesores más profundos que los sucesores habituales, tienen por lo general un coste g más alto. Por otro lado, las heurísticas que utilizaremos, normalmente proporcionan subestimaciones, aunque esto no ocurre en todos los casos, ya que no se puede demostrar que sean admisibles, ocurre con frecuencia. Esto significa que si se utiliza como función de evaluación $f(n) = \omega \times h(n) + g(n)$, probablemente el estado *lookahead* se sitúe en la lista ABIERTA después de los demás sucesores. La solución que se plantea está orientada a evitar este comportamiento, sobre todo al principio de la ejecución. El objetivo es que si es posible llegar a una solución rápidamente, mediante la obtención sucesiva de estados *lookahead*, el hecho de que estos tengan una g elevada no lo impida. Sin embargo, a medida que va avanzando el algoritmo, en la lista ABIERTA se encontraran tanto estados *lookahead* como estados generados normalmente.

Cada vez que se expande un nuevo nodo de la lista ABIERTA, y antes de generar sus sucesores debidos a acciones *helpful* se realiza un bucle en el que se generan sucesivamente todos los estados *lookahead* posibles. Cada uno de estos estados *lookahead* se introduce en la lista *ABIERTA* realizando el mismo tratamiento para estados repetidos que se realiza para los sucesores habituales. Si en un momento dado un estado *lookahead* es un nodo solución, en lugar de esperar a extraerlo de la lista *ABIERTA*, se genera la nueva solución y se actualiza el límite de coste (*límite_coste*). Los estados *lookahead* se generan utilizando el algoritmo que se explicó en el capítulo 6 (sección 6.2.2, página 146). La Figura 9.3 muestra el pseudocódigo que incorpora los estados *lookahead* en el proceso de búsqueda. Este pseudocódigo, que se ejecutará en el algoritmo en la Figura 9.2, dentro del primer **else** y antes del **forall** que se realiza para generar los sucesores, dando lugar al algoritmo BB-BFS-AHA-L.

9.3. Resultados experimentales (III)

A continuación se detallan los distintos experimentos que se han llevado a cabo. Estos incluyen el estudio del comportamiento de distintas heurísticas con el algoritmo BFS tanto sin utilizar acciones *helpful*, como utilizándolas con el esquema AHA descrito en la sección anterior (prioridad absoluta a acciones *helpful*); la comparación entre este algoritmo y el mismo algoritmo pero incorporando estados *lookahead*; y la comparación entre este último caso, la búsqueda CEHC y el comportamiento de otros planificadores.

```
function BB-BFS-AHA(𝓘, 𝓖)
let ABIERTA = 𝓘;
let SECUNDARIA = ∅;
let CERRADA = ∅;
let límite_coste = ∞;
let planes = ∅;
while ABIERTA ≠ ∅ and not límite_tiempo do
    estado_actual ← arg min_{x∈ABIERTA} f(x)
    ABIERTA ← ABIERTA \ {estado_actual}
    CERRADA ← CERRADA ∪ {estado_actual}
    if nodo_meta(estado_actual, 𝓖) then
        plan ← extraer_plan(estado_actual)
        planes ← planes ∪ {plan}
        límite_coste ← coste(plan)
    else if g(estado_actual) < límite_coste then
        forall sucesor ∈ sucesores_helpful(estado_actual) do
            if sucesor = generado ∈ ABIERTA
                if g(sucesor) < g(generado) then
                    g(generado) ← g(sucesor)
                    f(generado) ← ω × h(generado) + g(sucesor)
            else if sucesor = expandido ∈ CERRADA
                if g(sucesor) < g(expandido) then
                    CERRADA ← CERRADA \ {expandido}
                    f(sucesor) ← ω × h(expandido) + g(sucesor)
                    ABIERTA ← ABIERTA ∪ {sucesor}
            else
                f(sucesor) ← ω × h(sucesor) + g(sucesor)
                ABIERTA ← ABIERTA ∪ {sucesor}
        forall sucesor ∈ sucesores_NO_helpful(estado_actual) do
            SECUNDARIA ← SECUNDARIA ∪ {sucesor}
    if ABIERTA = ∅ then
        forall estado ∈ SECUNDARIA do
            SECUNDARIA ← SECUNDARIA \ {estado}
            if estado = generado ∈ ABIERTA
                if g(estado) < g(generado) then
                    g(generado) ← g(estado)
                    f(generado) ← ω × h(generado) + g(estado)
            else if estado = expandido ∈ CERRADA
                if g(estado) < g(expandido) then
                    CERRADA ← CERRADA \ {expandido}
                    f(estado) ← ω × h(expandido) + g(estado)
                    ABIERTA ← ABIERTA ∪ {estado}
            else
                f(estado) ← ω × h(estado) + g(estado)
                ABIERTA ← ABIERTA ∪ {estado}
return planes
```

Figura 9.2: Búsqueda *Branch and Bound Best-First con prioridad absoluta a acciones helpful* (AHA).

```
if NOT estado_lookahead(estado_actual) then
    estado_lookahead = obtener_estado_lookahead(estado_actual, estado_actual.RP)
while estado_lookahead do
    f(estado_lookahead) ← ω × h(estado_lookahead) + g(estado_lookahead)
    ABIERTA ← ABIERTA ∪ {estado_lookahead}
    if nodo_meta(estado_lookahead) then
        plan ← extraer_plan(estado_lookahead)
        planes ← planes ∪ {plan}
        límite_coste ← coste(plan)
    estado_lookahead = obtener_estado_lookahead(estado_lookahead, estado_lookahead.RP)
```

Figura 9.3: Pseudocódigo para incorporar estados *lookahead*.

9.3.1. Experimento 7: BFS puro, primera solución

En este experimento el objetivo es demostrar que la escalabilidad de BFS puro es insuficiente. Para ello se ha ejecutado el algoritmo con $\omega = 3$ con distintas heurísticas (en planificación clásica se suele tomar 3 ó 5). En todos los experimentos posteriores se tomará este valor para ω.

Las variables dependientes de este experimento son el número de problemas resueltos, la calidad de las soluciones encontradas y el número de nodos evaluados. Las heurísticas utilizadas son $h_{level\text{-}max}$, $h_{level\text{-}add}$, h_{mff}, h_{add} y h_{max}.

Las Figuras 9.4 y 9.5 muestran los resultados. Como se puede observar, el porcentaje de problemas resueltos es relativamente bajo. Algunos dominios en los que es especialmente bajo son:

- *Zenotravel*, en el que el esquema CEHC con algunas heurísticas resolvía el 100 % de los problemas, mientras que BFS sólo consigue resolver el 70 %.
- *Satellite*, en el que el esquema CEHC con algunas heurísticas resolvía el 95 % de los problemas, mientras que BFS sólo consigue resolver el 30 %.
- *Assignment*, en el que el esquema CEHC con algunas heurísticas resolvía el 100 % de los problemas, mientras que BFS sólo consigue resolver el 77 %.
- *Delivery*, en el que el esquema CEHC con algunas heurísticas resolvía el 100 % de los problemas, mientras que BFS sólo consigue resolver el 37 %.

Respecto a las heurísticas, las que ofrecen un mejor comportamiento en relación con los calidades encontradas son:

- $h_{level\text{-}max}$ y $h_{level\text{-}add}$ en *Zenotravel*, *Depots* y *MST*.
- En *Satellite*, *Driverlog* y *Assignment*, h_{add} ofrece en algunos casos mejores o iguales calidades que las anteriores. En *Driverlog*, h_{add} permite resolver un 30 % más de problemas.

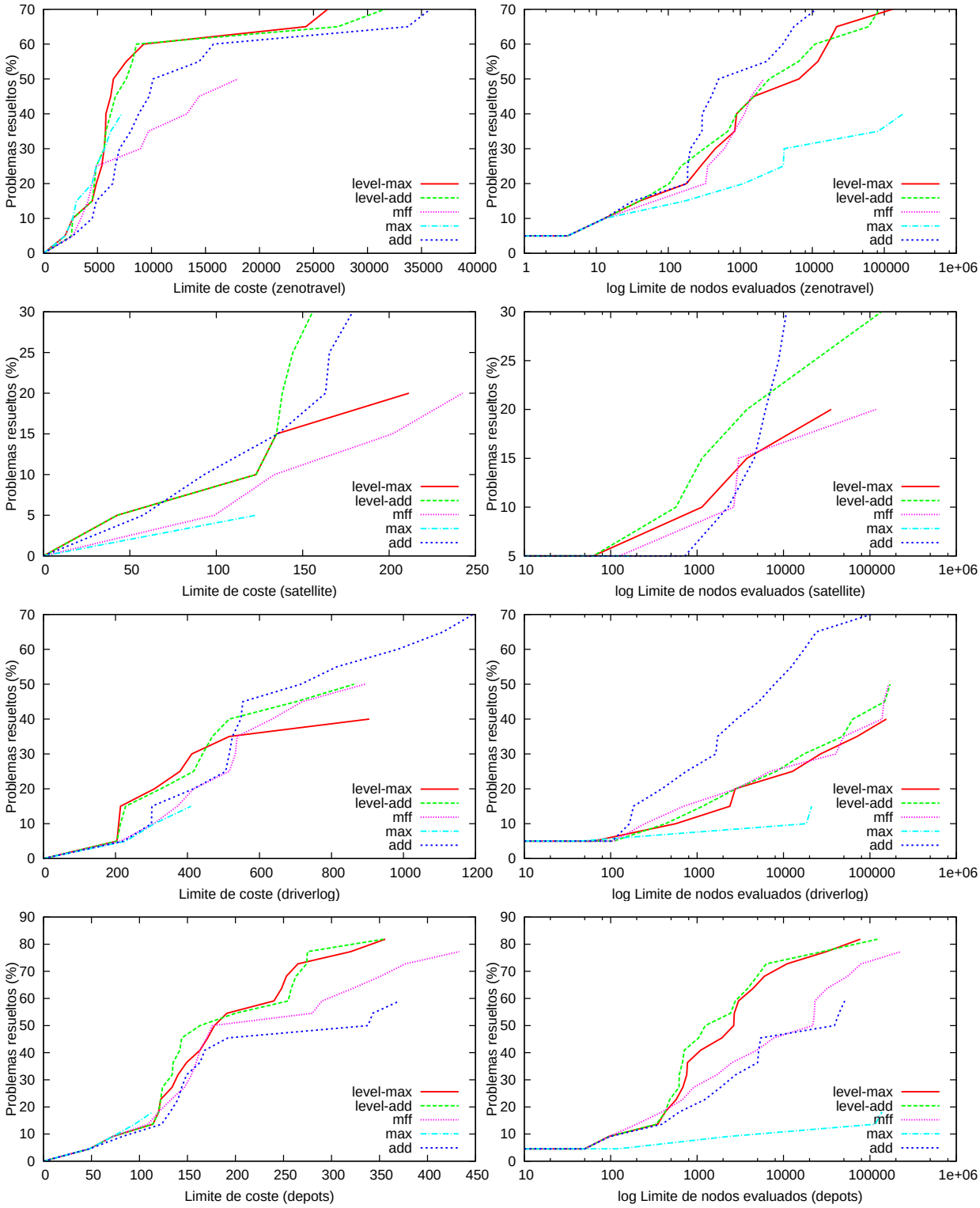

Figura 9.4: Resultados de la comparación entre distintas heurísticas en BFS (I).

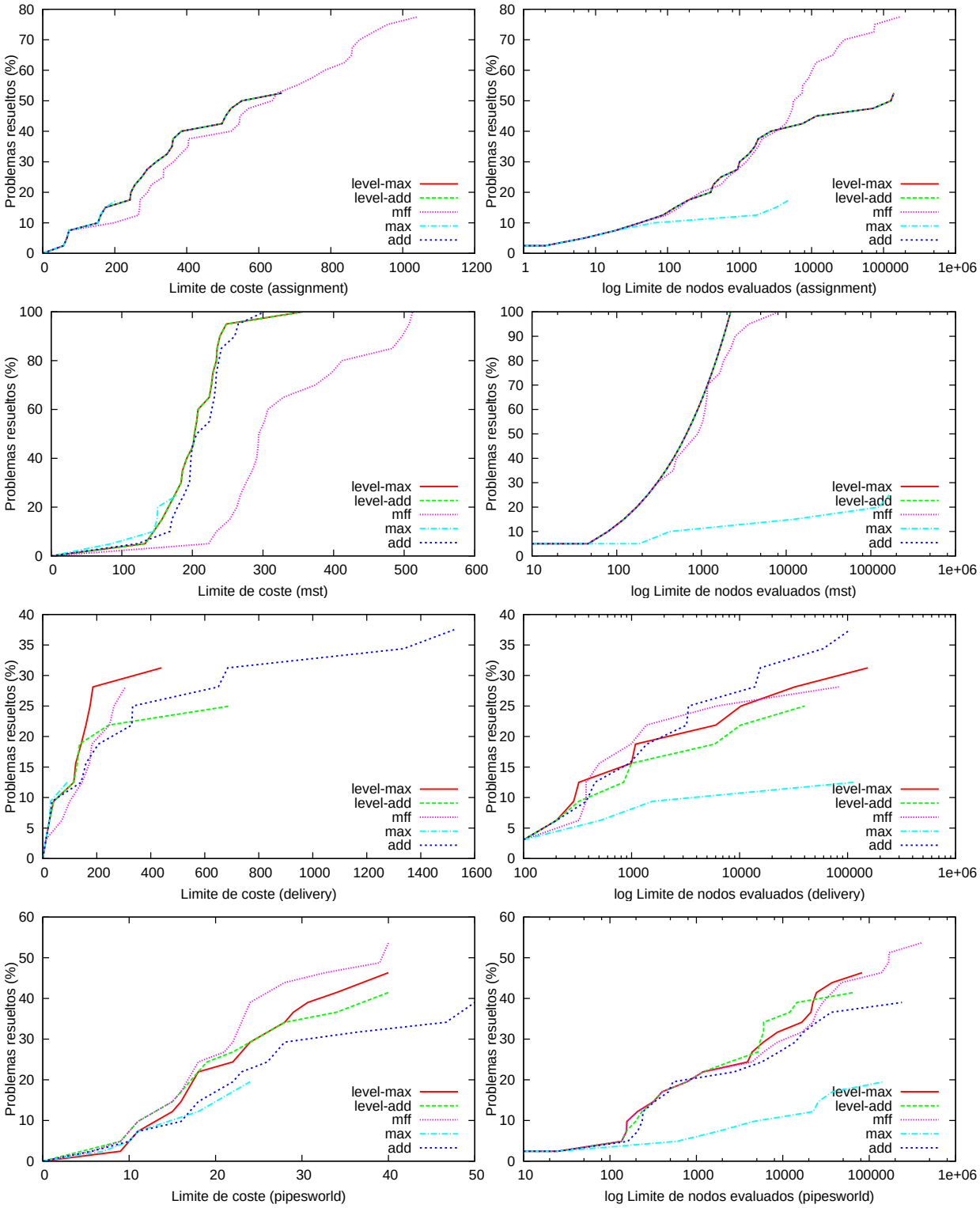

Figura 9.5: Resultados de la comparación entre distintas heurísticas en BFS (II).

- En *Delivery*, las mejores calidades se obtienen con $h_{level\text{-}max}$, aunque h_{add} resuelve algunos problemas más.
- En *Pipesworld*, las mejores calidades de obtienen con h_{mff}. Esta heurística es también la que más problemas resuelve en *Assignment*.

Estos resultados son coherentes con los de CEHC, ya que en general se puede decir que las mejores heurísticas son $h_{level\text{-}max}$, $h_{level\text{-}add}$ y h_{add}. Al igual que en CEHC existen algunas excepciones como el dominio *Pipesworld* donde las mejores calidades tanto con CEHC como con BFS se encuentran con h_{mff}.

9.3.2. Experimento 8: BFS puro vs. BFS-AHA (primera solución)

El objetivo de este experimento es determinar si la escalabilidad de BFS utilizando la misma heurística se mejora haciendo uso de la variación que da prioridad de forma absoluta a las acciones *helpful* (AHA). Como se explicó anteriormente, esta variación consiste en mantener una segunda lista abierta que contiene los nodos debidos a acciones no *helpful*, y en la que los nodos permanecen sin ser evaluados. Además, se estudia el efecto de esta variación sobre la calidad de las soluciones encontradas.

En este experimento se utilizan las heurísticas $h_{level\text{-}max}$ y $h_{level\text{-}add}$. Las variables dependientes del experimento son el número de problemas resueltos, la calidad de los planes encontrados, el número de nodos evaluados y el tiempo de CPU.

Los resultados respecto a calidad y nodos evaluados se muestran en las Figuras 9.6 y 9.7. Los resultados respecto al tiempo se encuentran en el apéndice A. En cada gráfica, las etiquetas que sólo contienen el nombre de la heurística se refieren a BFS mientras que las que están etiquetadas con AHA se refieren a la variación.

Como se puede observar, en las gráficas en los 8 dominios y para la misma heurística considerar la variación AHA mejora la escalabilidad del algoritmo a la vez que permite encontrar planes de mejor calidad sin que esto suponga un incremento ni en el número de estados evaluados, ni en el tiempo que se tarda en encontrar la solución, sino todo lo contrario.

El hecho de que se encuentren planes de mejor calidad supone que las acciones *helpful* son adecuadas y están informadas en relación con esta variable (lo que también ocurría en CEHC). La mejora en el tiempo se debe principalmente al ahorro de evaluaciones que se consigue. Como se observa en las gráficas, la variación AHA encuentra la solución realizando menos evaluaciones y esto implica una reducción del tiempo. Aunque la diferencia entre BFS y BFS-AHA es en general significativa, lo es en mayor medida en los dominios que tienen un factor de ramificación alto, como *Satellite* y *Delivery*. En estos dominios, el número de evaluaciones a realizar cada vez que se expande un nodo es muy alto, lo cual, si no se emplean acciones *helpful* para reducir el factor de ramificación, da lugar a que la búsqueda BFS sea muy lenta.

Hay dominios en los que la mejora en calidad no es muy significativa, como *Depots* y *Assignment*. Sin embargo en ambos casos se consigue mejor escalabilidad con la variación, ya que se resuelven más problemas utilizando menos tiempo.

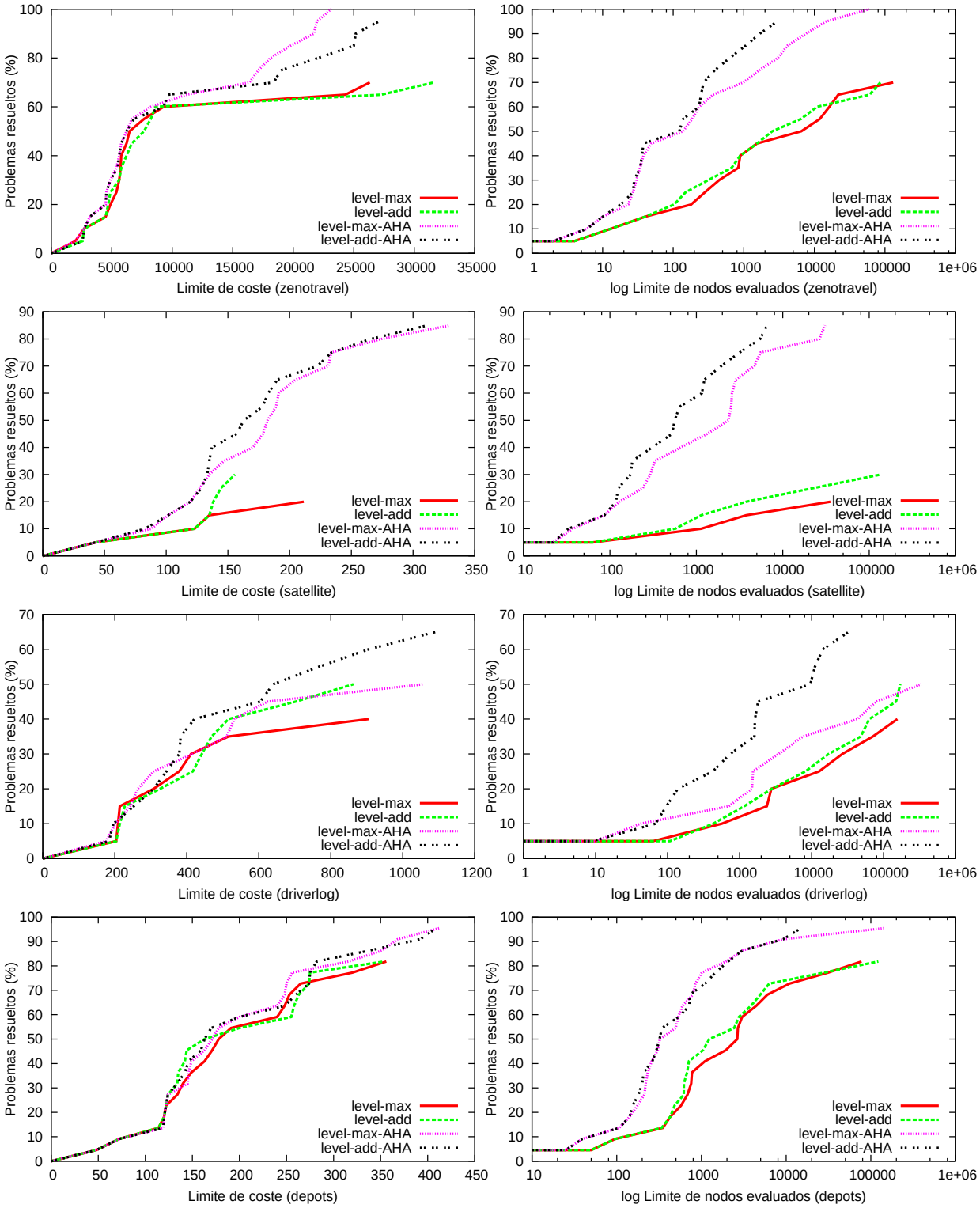

Figura 9.6: Resultados de la comparación entre BFS y BFS-AHA (I).

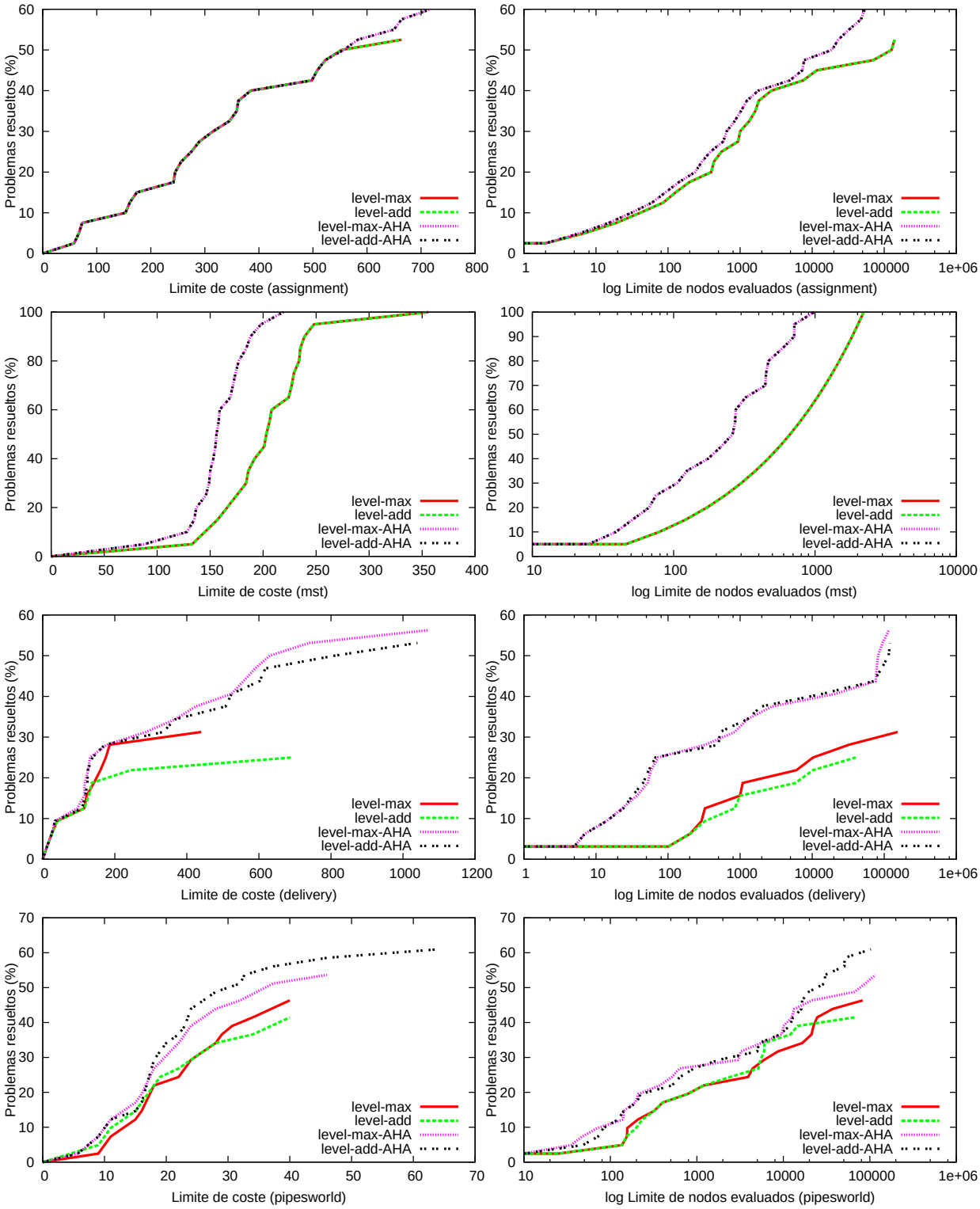

Figura 9.7: Resultados de la comparación entre BFS y BFS-AHA (II).

9.3.3. Experimento 9: BFS-AHA vs. CEHC (primera solución)

A pesar de que se ha demostrado que en los dominios utilizados BFS-AHA mejora la escalabilidad y las calidades de las soluciones encontradas respecto a BFS, por el momento no se ha determinado si esta mejora es suficiente y comparable a los resultados que se obtenían con CEHC en términos de escalabilidad. Por este motivo en este apartado se contrasta BFS-AHA con CEHC.

Al igual que en el experimento anterior, las heurísticas que se utilizan son $h_{level\text{-}max}$ y $h_{level\text{-}add}$. En este caso, las variables dependientes del experimento serán la calidad de las soluciones encontradas, el número de problemas resueltos y el tiempo empleado en encontrar la solución.

Las Figuras 9.8 y 9.9 muestran los resultados. En los dominios *Zenotravel*, *Satellite*, *Depots* y *Pipesworld*, utilizando la misma heurística, el esquema BFS-AHA proporciona en general mejores calidades que el esquema CEHC. Además, en los tres primeros dominios, la diferencia en tiempo no es muy significativa entre ambos esquemas. En *Pipesworld*, además el algoritmo BFS-HA es más escalable que CEHC porque resuelve considerablemente más problemas (casi el doble). Además en este dominio BFS-HA es el esquema más rápido con las dos heurísticas.

En los dominios *Driverlog*, *Assignment* y *Delivery*, se puede observar que aunque BFS-HA es capaz de encontrar planes con mayor calidad en algunos casos que CEHC, el problema de escalabilidad persiste ya que con BFS-HA se resuelven considerablemente menos problemas que con CEHC. Por ejemplo, en *Driverlog* con $h_{level\text{-}max}$ CEHC resuelve el 80 % de los problemas mientras que BFS-HA resuelve un 50 %. En *Assignment* se pasa de resolver un 100 % de los problemas con CEHC a resolver un 60 % con BFS-HA. Por último, en *Delivery* se pasa de resolver un 95 % de los problemas a resolver un 55 %. En estos dominios el algoritmo CEHC es considerablemente más rápido que BFS-HA.

El dominio *MST* es el único en el que claramente CEHC obtiene mejores calidades que BFS-HA. En este dominio en ambos casos se resuelven todos los problemas, pero CEHC es además más rápido que BFS-HA.

Como se ha demostrado, aunque escala mejor que el algoritmo BFS puro, BFS-HA sigue presentando problemas de escalabilidad. Sin embargo parece adecuado para encontrar planes de buena calidad. Este hecho motiva la inclusión de otras técnicas para intentar paliar el problema de escalabilidad. Como ya se ha venido explicando, en esta tesis se plantea utilizar estados *lookahead*. Estos estados permiten al algoritmo avanzar rápidamente y de una forma avara a estados más profundos y por lo tanto pueden ser útiles para mejorar la escalabilidad. Además al estar generados a partir de planes relajados que se han obtenido teniendo en cuenta los costes de las acciones, se espera que la calidad de los planes encontrados no se vea muy afectada.

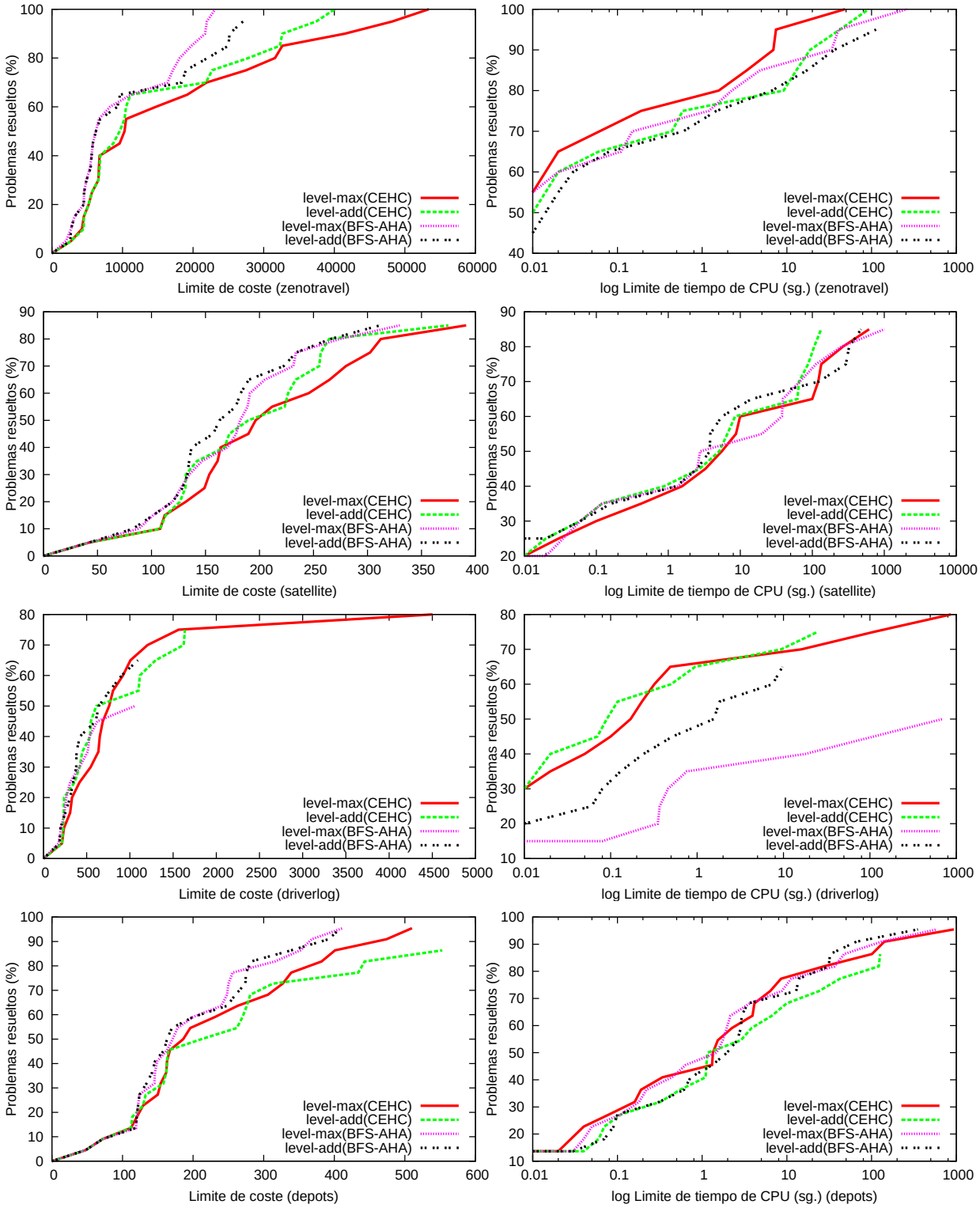

Figura 9.8: Resultados de la comparación CEHC vs. BFS-AHA(I).

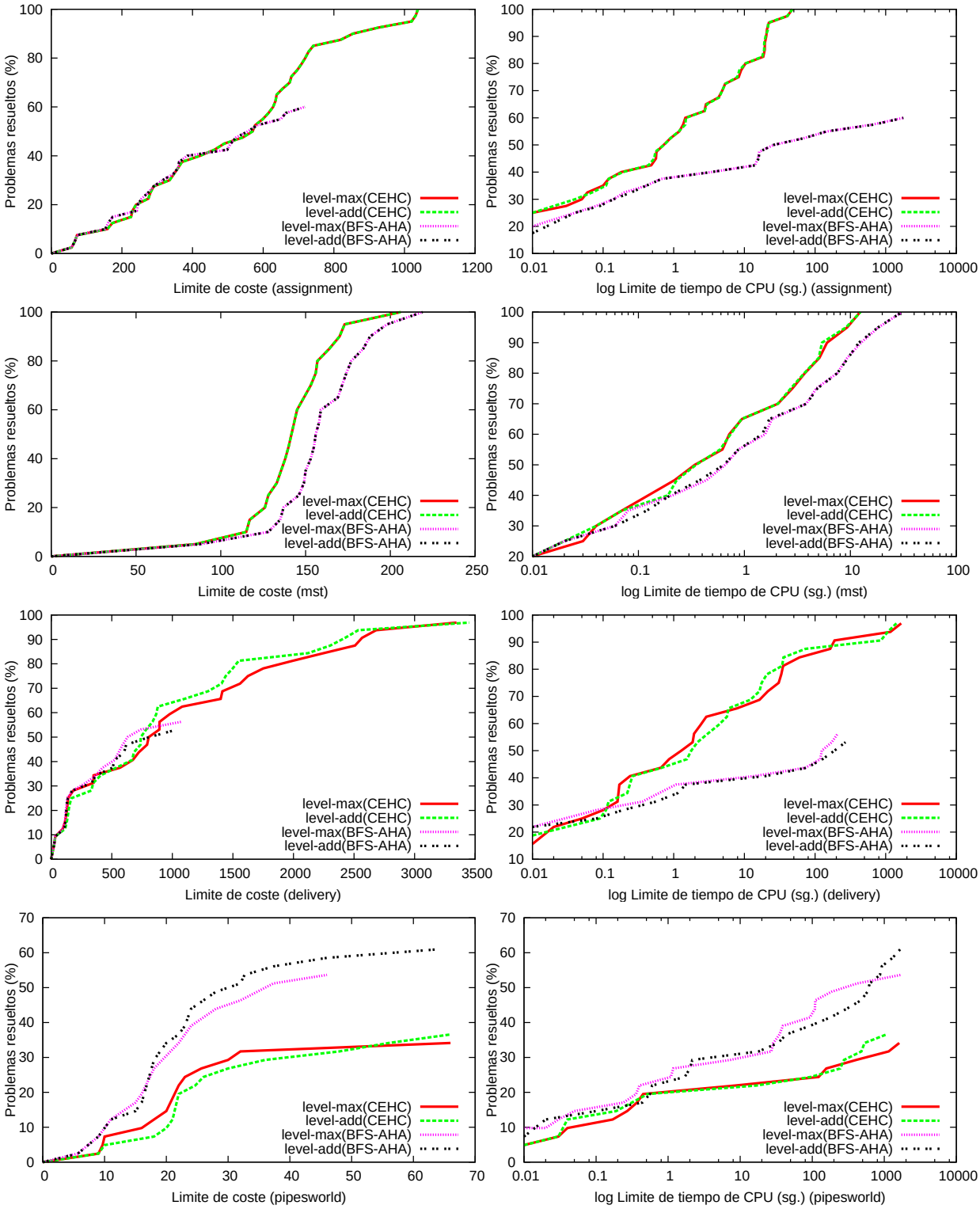

Figura 9.9: Resultados de la comparación CEHC vs. BFS-AHA(II).

9.3.4. Experimento 10: BB-BFS-AHA-L. Incorporación de estados *lookahead* y comportamiento *anytime* ($h_{level\text{-}max}$)

En este experimento se contrasta el comportamiento del algoritmo BB-BFS-AHA-L (definido en las Figuras 9.2 y 9.3) con otros algoritmos y planificadores. Este algoritmo incluye la incorporación de acciones *helpful* siguiendo el esquema AHA, la incorporación de estados *lookahead* siguiendo el esquema que se ha explicado previamente y el comportamiento *anytime* que se consigue haciendo un *Branch and Bound*, también como se ha explicado previamente. El experimento se ha realizado tomando como heurística $h_{level\text{-}max}$. Ésta es una de las heurísticas que ofrece mejor comportamiento de entre las estudiadas (junto con $h_{level\text{-}add}$ y h_{add}). Posteriormente, se contrastará el comportamiento de BB-BFS-AHA-L con las otras dos heurísticas.

Las variables dependientes son el número de problemas resueltos, la calidad de las soluciones encontradas y el tiempo de CPU. Los resultados de BB-BFS-AHA-L se contrastarán con: el mismo algoritmo pero sin la incorporación de estados *lookahead* (BB-BFS-AHA) y también con el comportamiento *anytime*; el algoritmo CEHC; y los otros planificadores: METRIC-FF (MFF), LPG, y LAMA. Este último sólo se incluirá en los casos que se mencionaron en el capítulo 7 debido al problema con el lenguaje que se mencionó en el mismo capítulo.

Valorar el comportamiento *anytime* es relativamente complicado, dado que se cuenta con una cantidad de soluciones indeterminada a priori, en distintos instantes de tiempo para cada problema. Esto hace más difícil obtener una síntesis adecuada de los resultados experimentales. Una posibilidad es fijarse únicamente en la última solución encontrada, como se optó en la última Competición Internacional de Planificación (2008). Sin embargo esto supone eliminar prácticamente la variable tiempo. En este caso se ha optado por discretizar el tiempo en distintos intervalos y valorar el comportamiento en cada uno de ellos. Se ha discretizado el tiempo en los siguientes 6 intervalos expresados en segundos: [0,1], (1,5], (5, 25], (25,125], (125,625] y (625, 1800]. Algunos algoritmos/planificadores sólo ofrecen una solución. En este caso la solución sólo aparecerá en el intervalo de tiempo en que haya sido encontrada y en los posteriores. Los resultados para los 8 dominios se muestran en las Figuras 9.10–9.17.

En *Zenotravel*, como se puede observar, BB-BFS-AHA-L presenta muy buena escalabilidad ya que se resuelven todos los problemas en menos de 1 segundo. Resuelve más problemas que BB-BFS-AHA hasta el intervalo (125-625], que CEHC y LPG hasta el intervalo (25,125], y que METRIC-FF en todos los intervalos. Además, junto con BB-BFS-AHA es la aproximación que encuentra planes con menor coste en todos los intervalos.

En *Satellite*, BB-BFS-AHA-L también presenta muy buena escalabilidad en tiempos pequeños, junto con LPG, aunque BB-BFS-AHA-L resuelve un problema menos hasta el intervalo (125,626]. En general BB-BFS-AHA-L es la aproximación que encuentra los planes de más calidad junto con BB-BFS-AHA. Sin embargo, este último es menos escalable. En algunos intervalos de tiempo LPG resuelve algún problema con más calidad que BB-BFS-AHA-L, pero finalmente BB-BFS-AHA-L mejora sus soluciones y en el último intervalo proporciona planes considerablemente mejores que LPG en todos los problemas. Por último,

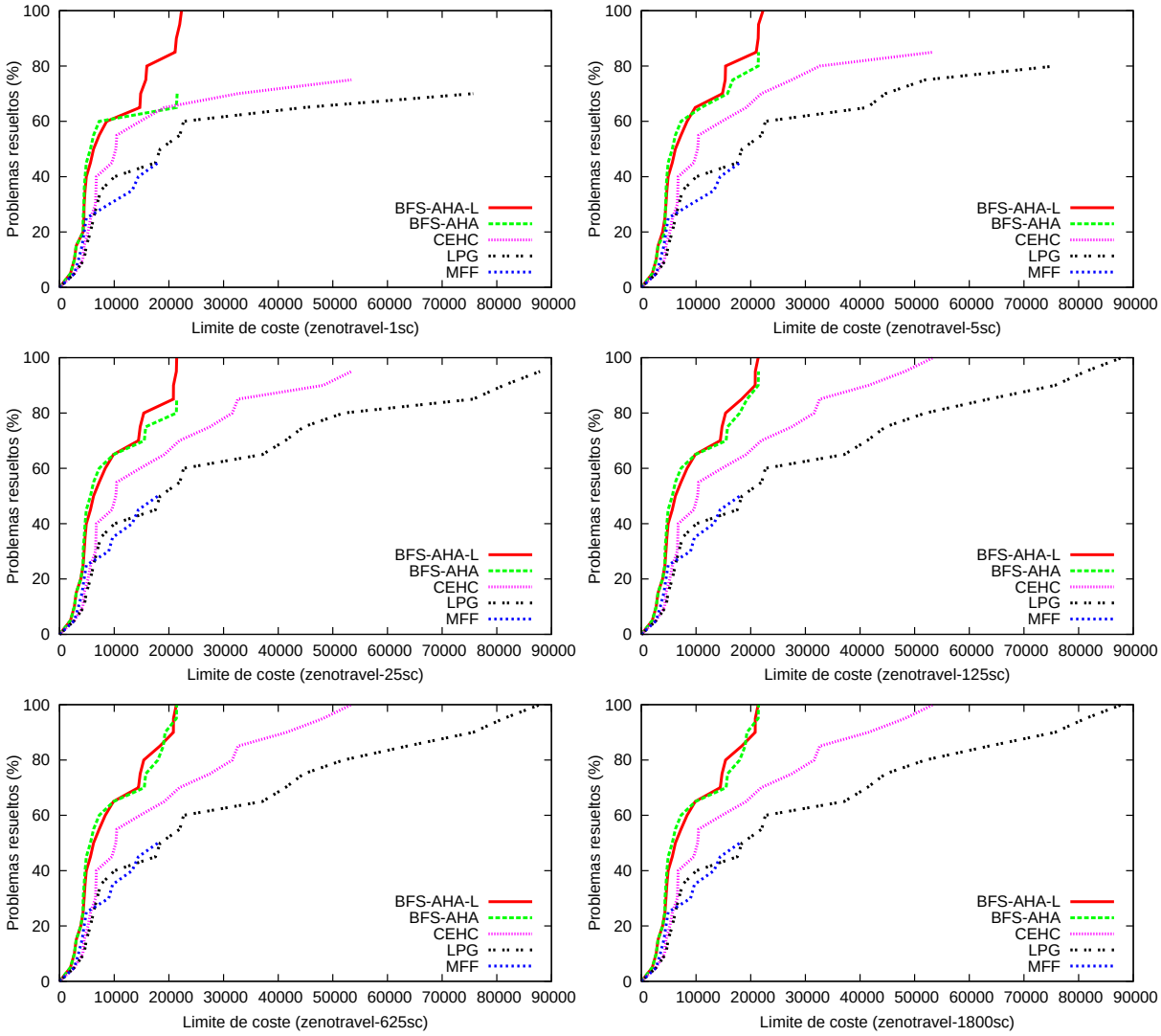

Figura 9.10: BB-BFS-AHA-L: Resultados en el domino *Zenotravel*.

BB-BFS-AHA-L es claramente mejor que CEHC y que METRIC-FF.

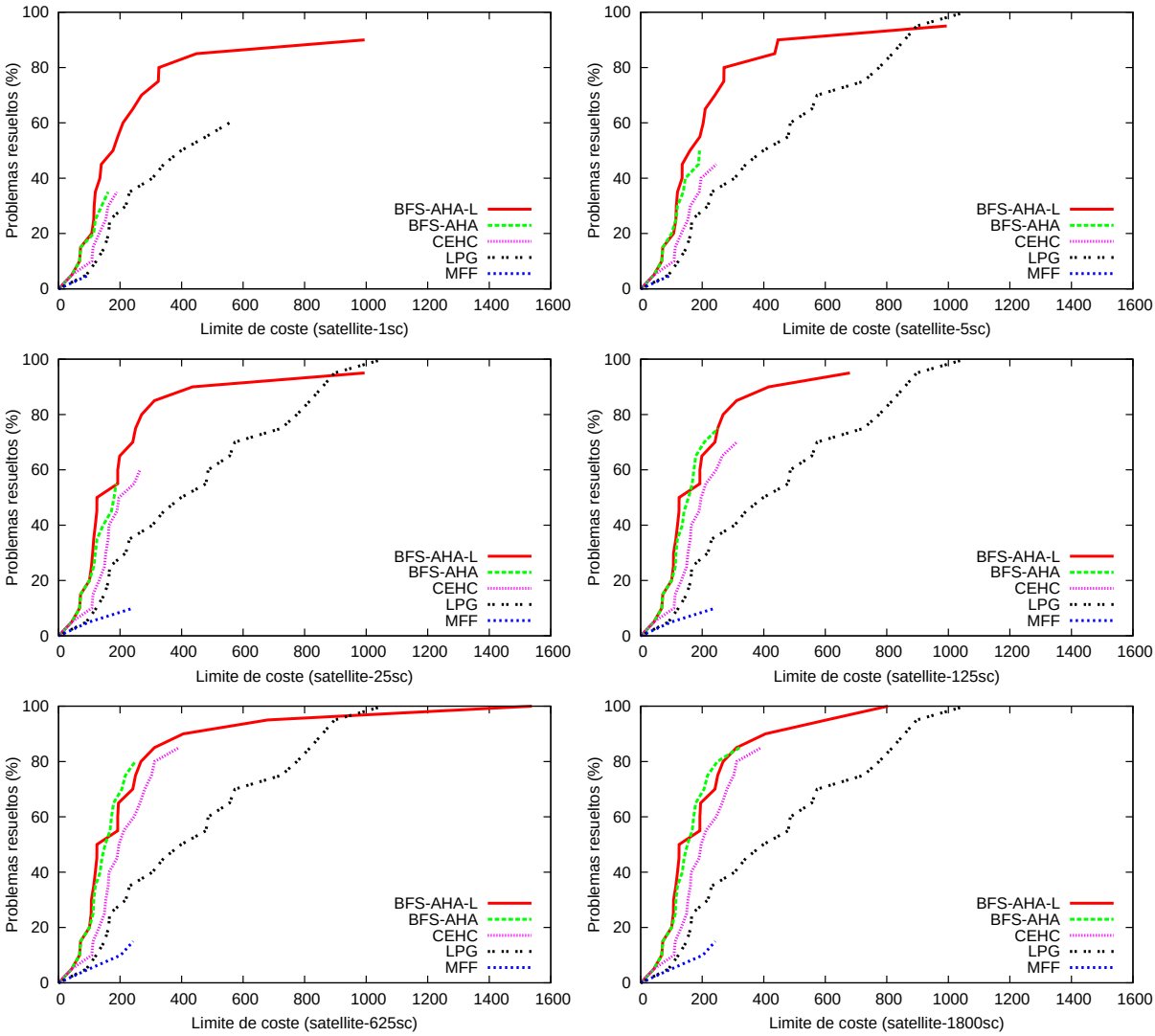

Figura 9.11: BB-BFS-AHA-L: Resultados en el domino *Satellite*.

En *Driverlog*, BB-BFS-AHA-L también escala mejor que las demás aproximaciones. Es la única que resuelve todos los problemas, a partir del intervalo (25,125]. Obtiene en todos los intervalos mejores calidades que LPG, METRIC-FF y CEHC. En relación con LAMA, en problemas pequeños encuentra calidades similares pero en los problemas más complejos encuentra planes de calidad considerablemente mejor, salvo en un problema en que LAMA encuentra una solución un poco mejor.

En *Depots* el uso de estados *lookahead* es menos adecuado, aunque en todos los instantes de tiempo ofrece mejores resultados que CEHC y METRIC-FF. En problemas sencillos BB-BFS-AHA-L resuelve algún problema más, y con más calidad que BB-BFS-AHA, pero en

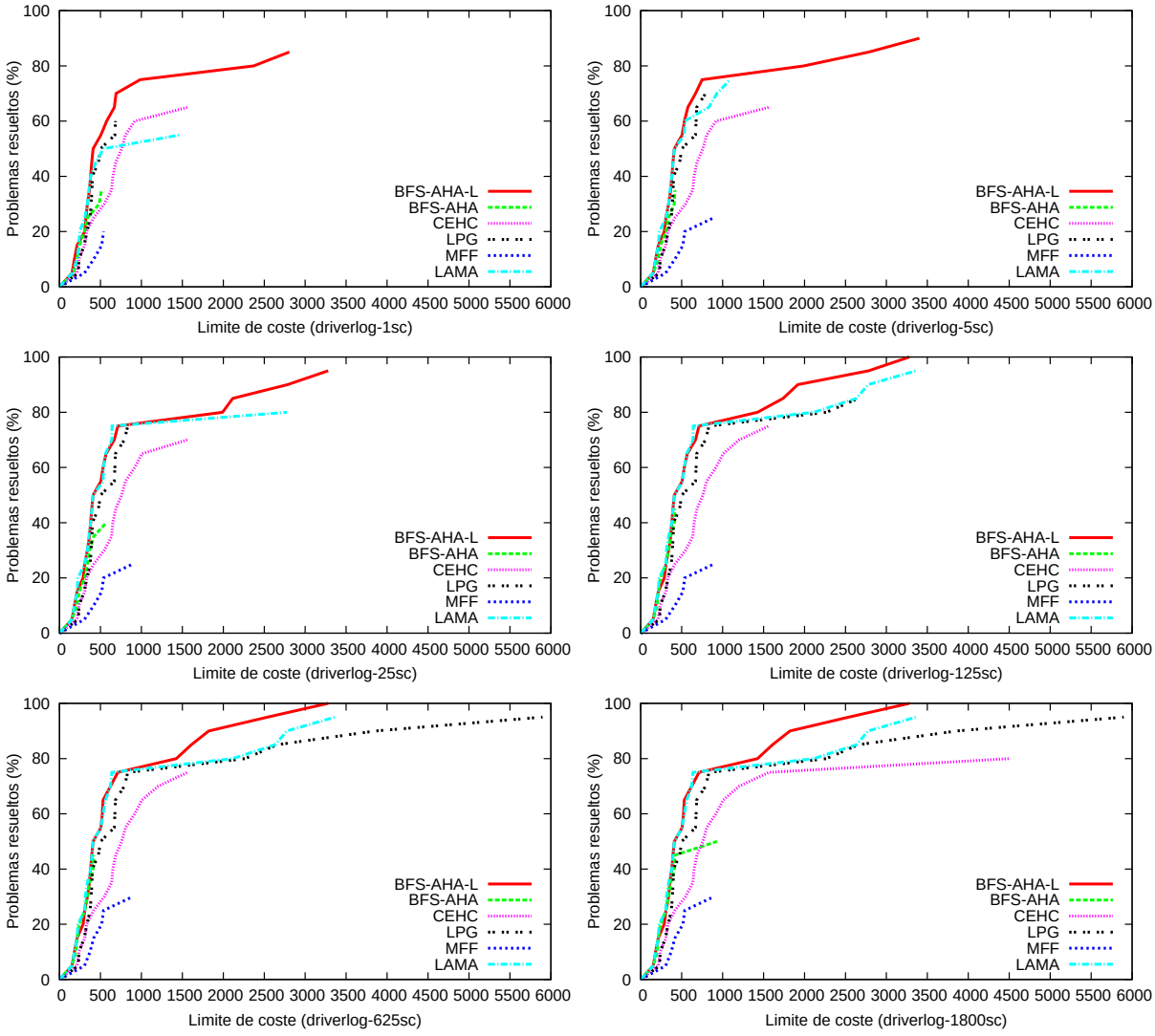

Figura 9.12: BB-BFS-AHA-L: Resultados en el domino *Driverlog*.

problemas de complejidad media BFS-AHA ofrece un mejor comportamiento en términos de calidad. Esto se debe a que con los planes relajados se pierde mucha información, al ser un dominio con interacciones complejas entre metas. Esto da lugar a que se produzcan mejores resultados no incorporando los estados *lookahead*. Hasta el intervalo (25,125] los mejores resultados se obtienen con LAMA. Sin embargo, en el último intervalo BB-BFS-AHA, LPG y LAMA ofrecen calidades muy parecidas. LPG junto con BB-BFS-AHA-L son las dos únicas aproximaciones que consiguen resolver todos los problemas.

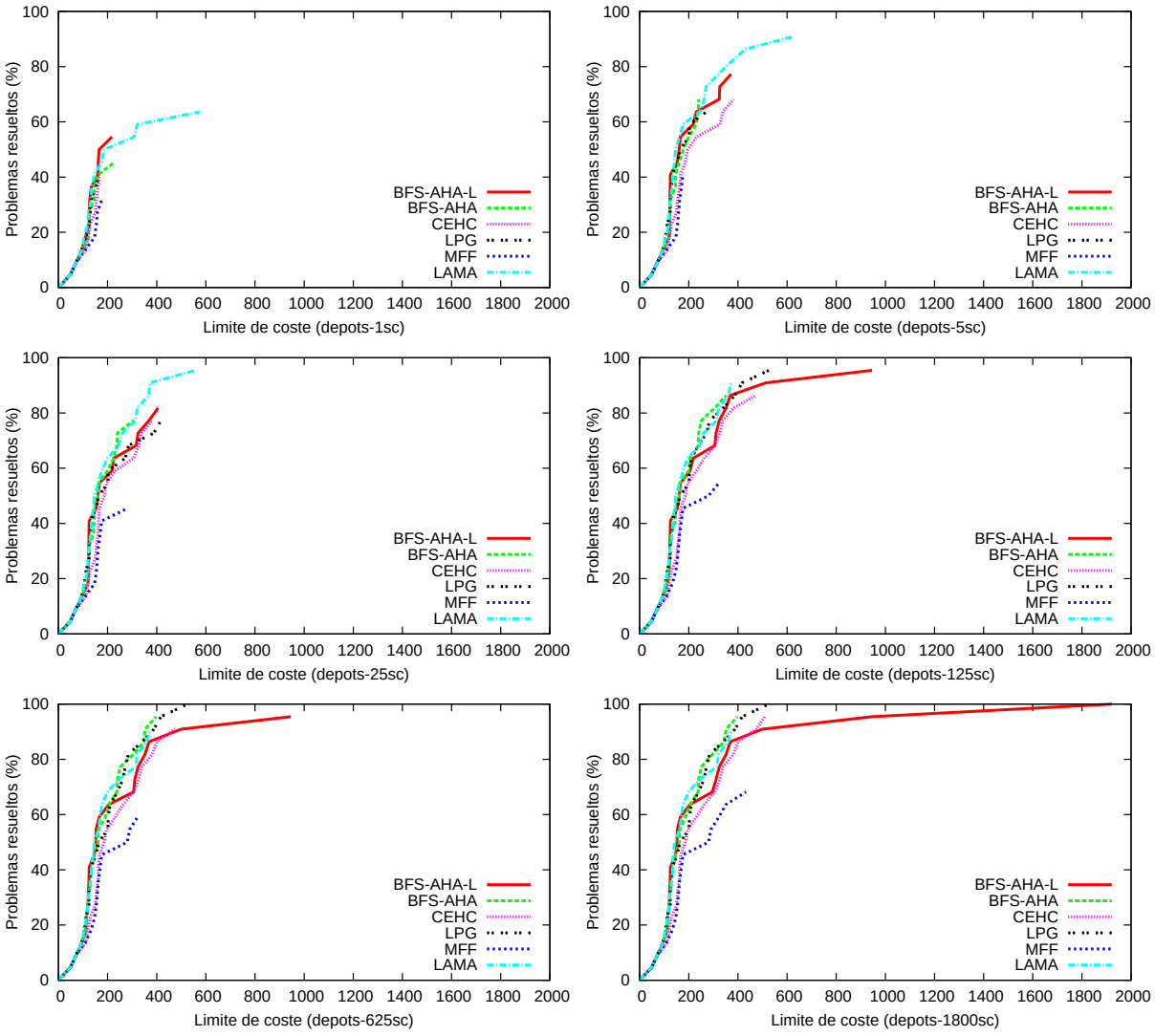

Figura 9.13: BB-BFS-AHA-L: Resultados en el domino *Depots*.

En *Assignment*, BB-BFS-AHA-L mejora notablemente la escalabilidad de BB-BFS-AHA, ya que resuelve muchos más problemas en todos los intervalos. Aunque en el primer intervalo BB-BFS-AHA-L es la mejor aproximación, y en el segundo lo es LAMA, a partir de

5 segundos tanto BB-BFS-AHA-L, como LAMA y CEHC ofrecen resultados muy similares, notablemente mejores que LPG, BB-BFS-AHA y METRIC-FF.

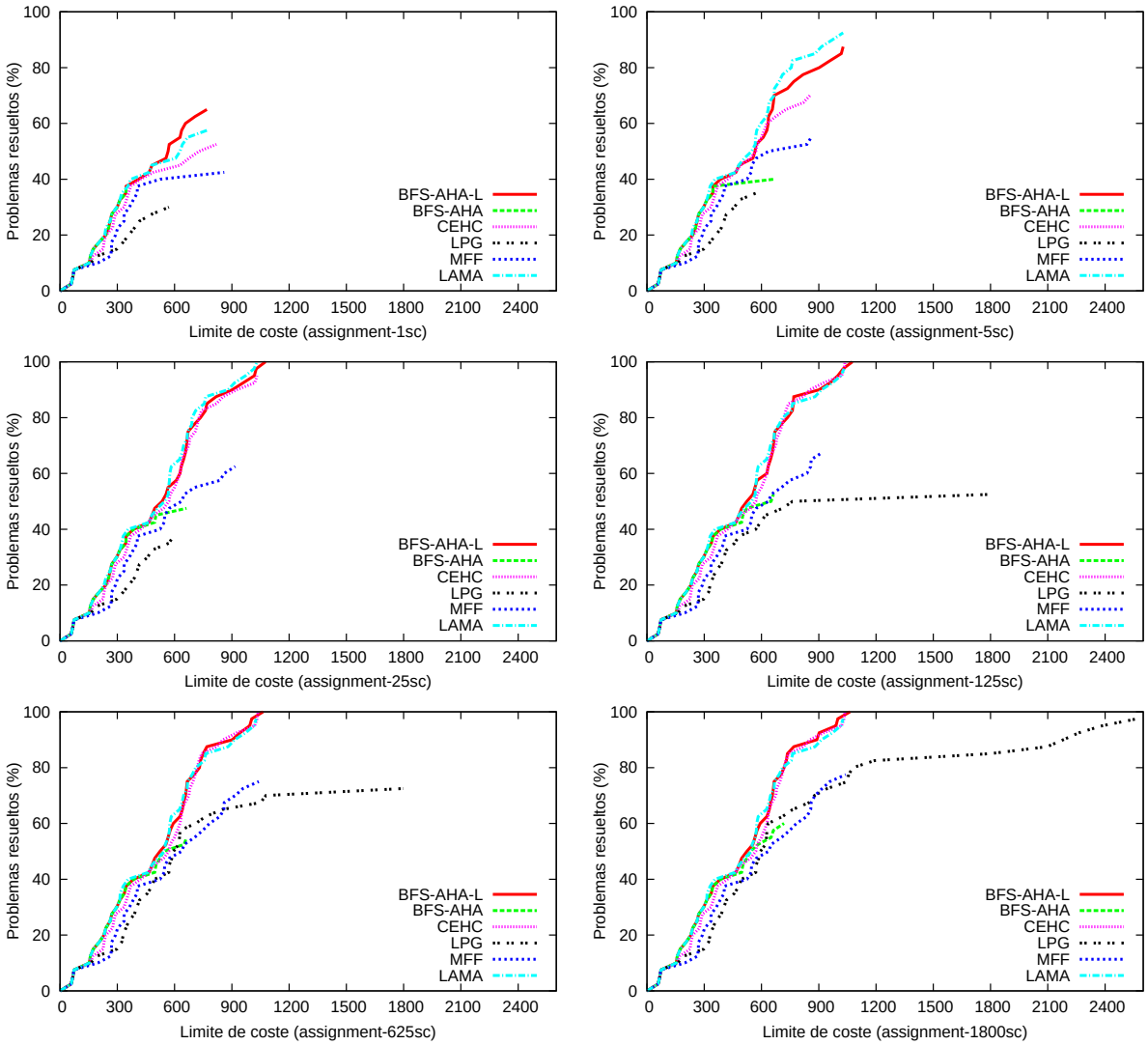

Figura 9.14: BB-BFS-AHA-L: Resultados en el domino *Assignment*.

En *MST* el uso de estados *lookahead* tampoco parece muy adecuado. En tiempos pequeños, BB-BFS-AHA-L mejora la escalabilidad de BB-BFS-AHA notablemente, a costa de empeorar la calidad. A partir del intervalo (25-15] BB-BFS-AHA es claramente mejor que BB-BFS-AHA-L, aunque este último es notablemente mejor que LPG y METRIC-FF en todos los intervalos. Cuando el tiempo es escaso, LAMA es la mejor aproximación. Sin embargo, a partir del intervalo (5,25] CEHC es comparable con LAMA e incluso mejor, ya que en los últimos problemas encuentra mejores calidades y resuelve algún problema más.

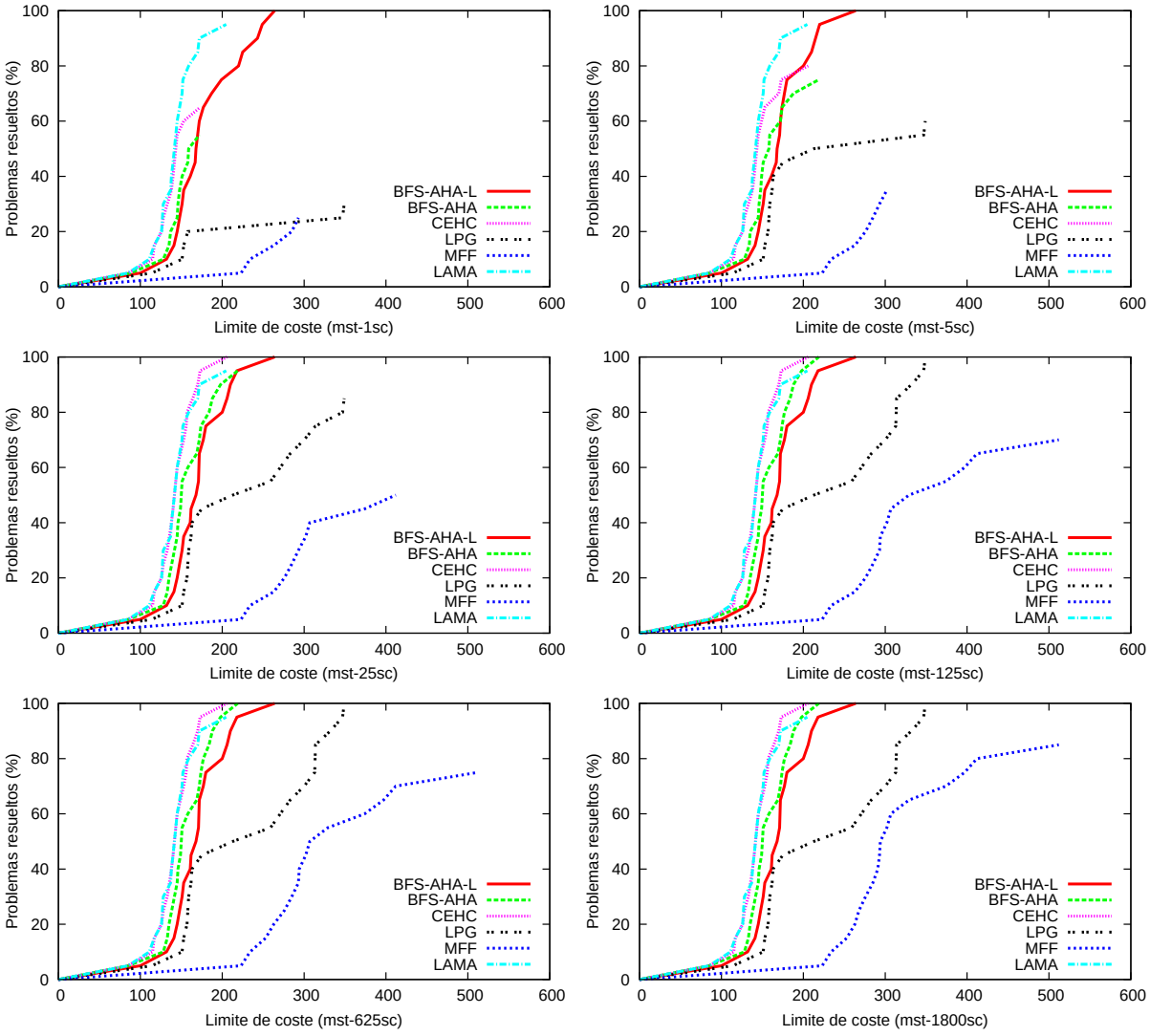

Figura 9.15: BB-BFS-AHA-L: Resultados en el domino *MST*.

En *Delivery*, BB-BFS-AHA-L es claramente la mejor aproximación en todos los intervalos, mejorando notablemente la escalabilidad de BB-BFS-AHA, y obteniendo calidades considerablemente mejores que cualquiera de las demás aproximaciones.

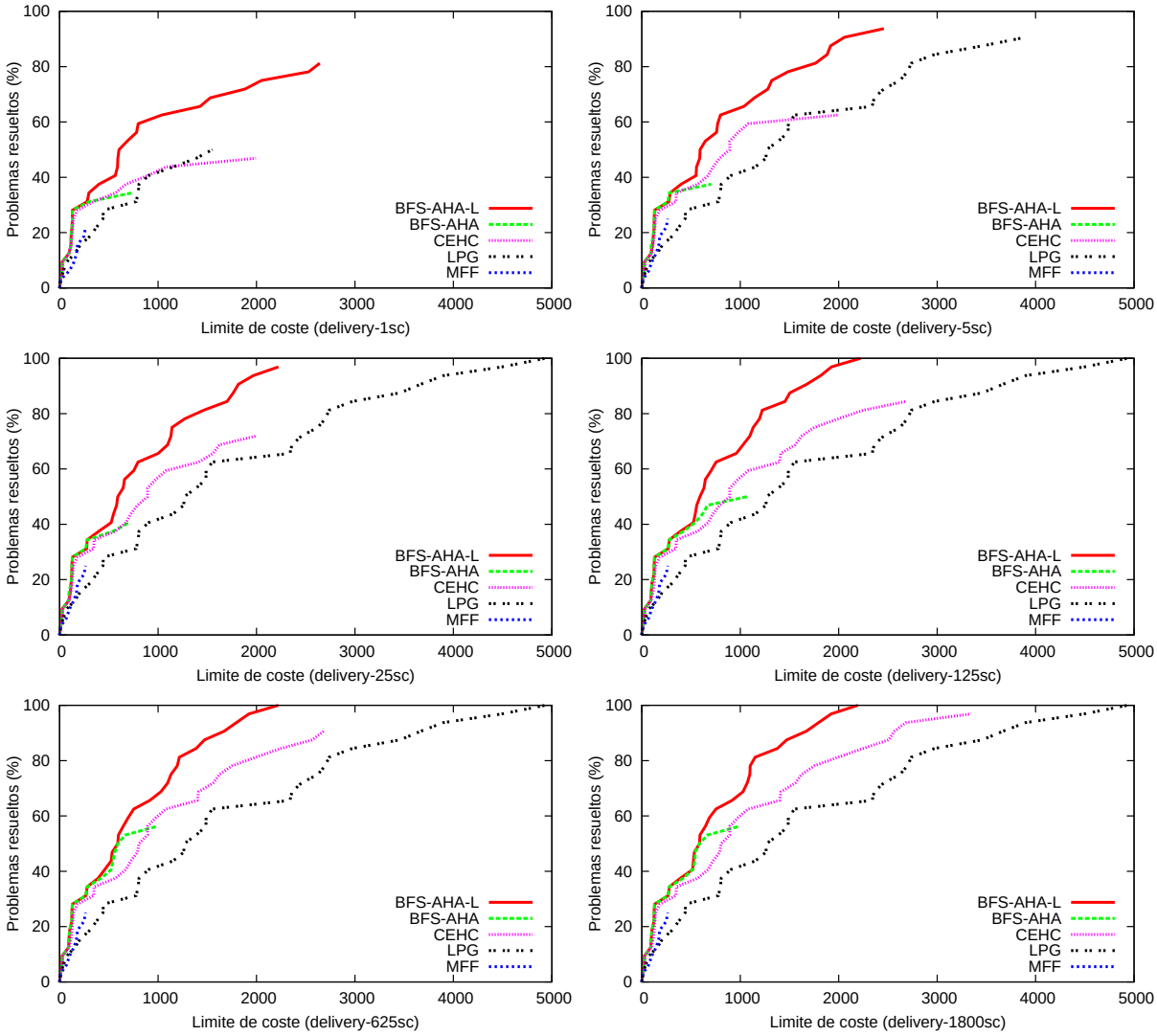

Figura 9.16: BB-BFS-AHA-L: Resultados en el domino *Delivery*.

Finalmente, en *Pipesworld*, BB-BFS-AHA-L también es la mejor aproximación. En este dominio el número de problemas resueltos es bajo en general. BB-BFS-AHA-L mejora un poco la escalabilidad de BB-BFS-AHA, generando planes de calidad similar. Las calidades encontradas son mejores que utilizando CEHC, LPG o METRIC-FF.

En resumen BB-BFS-AHA-L parece una muy buena aproximación para planificación basada en costes y en algunos dominios es comparable al planificador que ganó la última Competición Internacional de Planificación, LAMA. Los resultados son favorables en 6 de

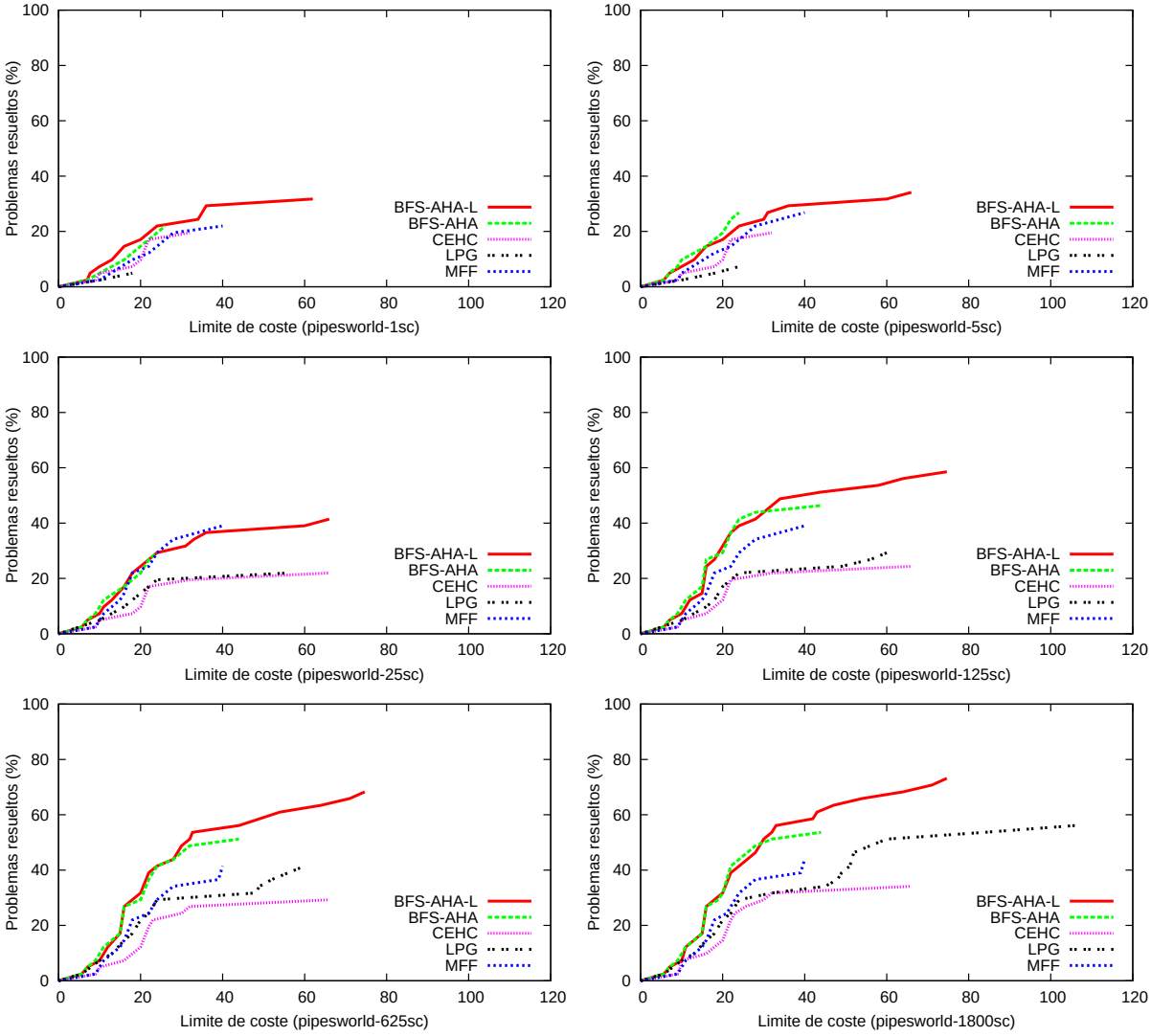

Figura 9.17: BB-BFS-AHA-L: Resultados en el domino *Pipesworld*.

los 8 dominios. En el dominio *Depots* es mejor aproximación BB-BFS-HA; y en *MST* es mejor aproximación CEHC. En ambos casos, el resultado es sutilmente mejor que LAMA.

Depots es un dominio con interacciones entre (sub)metas complejas, lo que hace que el plan relajado sea menos cercano al plan real. Por este motivo utilizar el plan relajado para generar el estado *lookahead* es menos adecuado que en otros dominios. Este problema también se producía en planificación clásica.

9.3.5. Capacidad de mejora de BB-BFS-AHA-L con $h_{level\text{-}max}$

En este apartado se estudia cuál es la capacidad de mejora del algoritmo BB-BFS-AHA-L con el tiempo. Al igual que en los experimentos anteriores se utiliza la heurística numérica $h_{level\text{-}max}$. La capacidad de mejora con el tiempo del algoritmo aporta información sobre el tiempo que se necesita para mejorar las soluciones encontradas previamente. Esta información es interesante, ya que desde el punto de vista del usuario puede que no merezca la pena utilizar la totalidad del tiempo (en el caso de los experimentos anteriores 1800 segundos) si el algoritmo no es capaz de obtener una mejora determinada de la calidad de la solución.

Aunque es posible determinar la capacidad de mejora de forma numérica, por ejemplo haciendo la media para todos los problemas de un dominio, se ha decidido hacerlo de forma gráfica para no perder información, utilizando la misma discretización del tiempo en intervalos del experimento anterior. Con este objetivo, se han generado las gráficas de la Figura 9.18. En estas gráficas (una por dominio) el eje x indica el problema correspondiente y el eje y el coste de la(s) solución(es) encontradas para ese problema. Cada punto para el mismo problema muestra una solución diferente. El color (y forma) del punto indica el intervalo de tiempo en que se ha encontrado esa solución. De esta forma se puede determinar en qué medida una solución mejora a las anteriores y cuánto tiempo (aproximado, ya que el tiempo está discretizado) se necesita para que se pueda conseguir la mejora.

Hay que resaltar que cuando una solución no se puede mejorar puede ser por dos causas: (1) porque es la solución óptima ó (2) porque el algoritmo es incapaz de mejorarla en el tiempo con el que cuenta. En los problemas sencillos lo más probable es que el motivo por el que no se produce mejora sea el primero. En este caso, el hecho de no encontrar una solución mejor no debe afectar negativamente a la capacidad de mejora del algoritmo.

Como se puede observar, en todos los dominios hay un número de problemas considerable en los que la solución se mejora haciendo el *Branch and Bound*, por lo que la técnica parece efectiva.

En algunos dominios, como *Zenotravel* y *MST* no se encuentra ninguna solución en los dos y tres últimos intervalos. En los demás dominios hay algunos problemas en los que la mejor solución se encuentra en el último intervalo.

9.3.6. Experimento 11: BB-BFS-AHA-L ($h_{level\text{-}max}$ v.s. $h_{level\text{-}add}$ vs. $h_{add\text{-}HA}$)

En este experimento se contrasta el resultado final (pasados 1800 segundos) que se obtiene utilizando BB-BFS-AHA-L con las heurísticas $h_{level\text{-}add}$ y $h_{add\text{-}HA}$. Los resultados se muestran en la Figura 9.19. Como se puede observar, la diferencia no es demasiado significativa en ninguno de los dominios. En los dominios *Assignment* y *MST* $h_{level\text{-}add}$ no se ha incluído porque proporciona los mismos resultados que $h_{level\text{-}max}$. Como ya se comentó anteriormente, en estos dominios ambas heurísticas coinciden.

En los dominios *Zenotravel* y *Driverlog*, $h_{level\text{-}max}$ tiene un comportamiento ligeramente mejor que $h_{level\text{-}add}$ y h_{add}. Lo mismo ocurre en *Assignment* y *MST* frente a h_{add}. Sin embargo, en *Satellite*, *Depots* y *Pipesworld*, $h_{level\text{-}add}$ obtiene calidades ligeramente mejores.

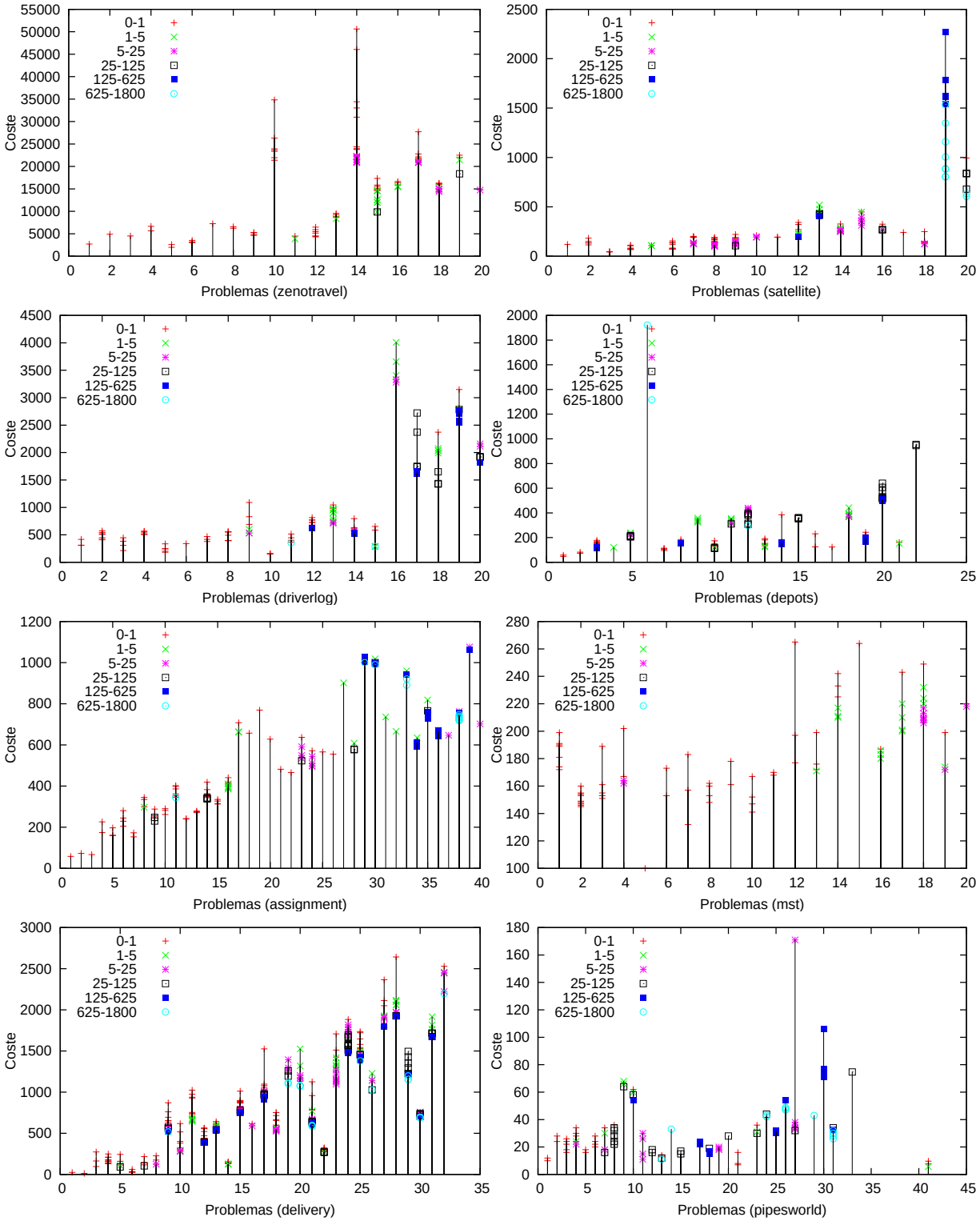

Figura 9.18: Capacidad de mejora de BB-BFS-AHA-L ($h_{level\text{-}max}$) en los distintos intervalos de tiempo.

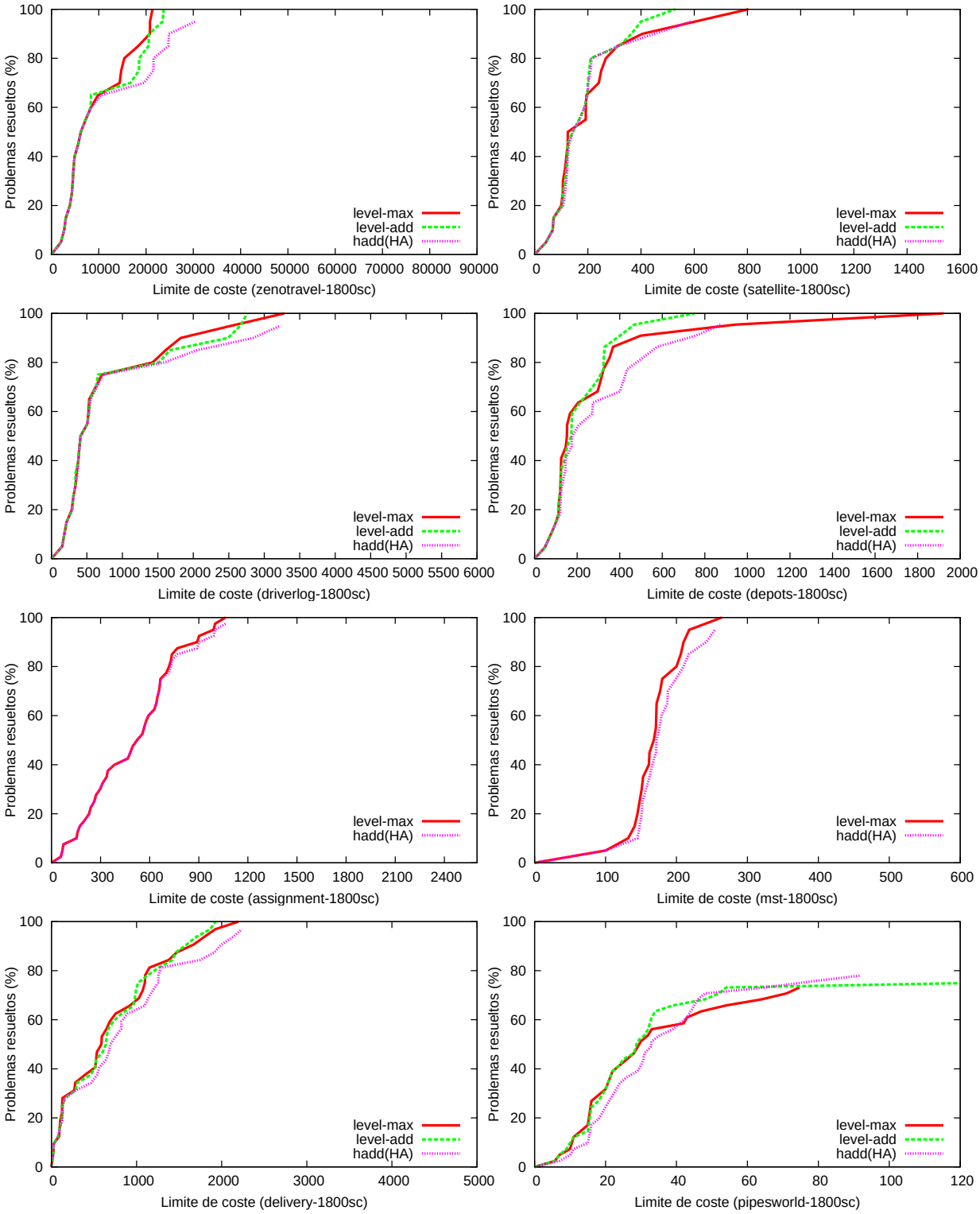

Figura 9.19: BB-BFS-AHA-L con distintas heurísticas.

9.4. Resumen

En este capítulo se han presentado modificaciones sobre un algoritmo base de tipo *mejor primero* para considerar acciones *helpful*, estados *lookahead* y contar con la capacidad de mejorar las soluciones encontradas hasta un cierto límite de tiempo. Después, se han presentado una serie de experimentos que justifican que estas modificaciones son interesantes porque mejoran la escalabilidad del algoritmo utilizando en todos los casos criterios informados respecto al coste de las acciones. Algunas conclusiones que se extraen de los experimentos son:

- BFS puro presenta una escalabilidad limitada, aunque para problemas sencillos obtiene soluciones con calidad.
- BFS-AHA (incluyendo acciones *helpful* con prioridad absoluta) mejora la escalabilidad de BFS encontrando soluciones razonables en términos de coste. Sin embargo en algunos dominios su escalabilidad no es comparable con la de un esquema de búsqueda local como CEHC.
- La aproximación final BB-BFS-AHA-L, que incluye además estados *lookahead* combinada con las heurísticas numéricas $h_{level\text{-}max}$, $h_{level\text{-}add}$, o $h_{add\text{-}HA}$ es en general muy escalable (tanto o más que los CEHC) y competitiva con otros planificadores.

El uso de este último esquema es menos adecuado en dominios con interacciones complejas entre (sub)metas, dado que el plan relajado, que se utiliza para obtener todas las heurísticas, es menos aproximado al plan real. En estos dominios es más recomendable utilizar el esquema sin estados *lookahead* o búsquedas de tipo CEHC.

Parte IV

Conclusiones y trabajos futuros

Capítulo 10

Conclusiones

En esta tesis se han estudiado tanto desde el punto de vista teórico como desde el punto de vista experimental distintas heurísticas y distintos algoritmos de búsqueda progresiva a aplicar en planificación basada en costes. Ambos puntos constituían los dos objetivos principales que se plantearon para la misma. A continuación se resume el trabajo realizado:

1. Respecto a las heurísticas, se ha confeccionado la heurística numérica h_{level} que se basa en una construcción del *RPG* en niveles incrementales de coste. A continuación se ha analizado la relación entre esta heurística y otras heurística numéricas, tanto desde el punto de vista procedural como desde el punto de vista declarativo.

2. El análisis de las heurísticas ha motivado la generación de una definición unificada de las mismas en la que se han fijado las partes comunes y se han generalizado las funciones que permiten hacer la propagación de costes hacia delante (de las precondiciones de las acciones a sus efectos) y la selección de acciones en el proceso de extracción. La definición unificada proporciona un marco teórico común para definir heurísticas.

3. Por otro lado, se han analizado dos heurísticas no numéricas, desde el punto de vista de su aplicación en planificación basada en costes: las acciones *helpful* y los estados *lookahead*. En ambos casos se han establecido mecanismos orientados a adaptar la idea a este tipo de planificación.

4. Respecto a los algoritmos de búsqueda se han planteado principalmente tres tipos de algoritmos y algunas variantes a los mismos: *Cost-Enforced Hill-Climbing* (CEHC), una variación de *Hill-Climbing* (HC-B) y una variación de *Best-First Search* con comportamiento *anytime*, en la que se incluyen acciones *helpful* y estados *lookahead* (BB-BFS-AHA-L). Estos algoritmos se han evaluado utilizando distintas heurísticas y se han comparado entre sí.

5. La evaluación se ha realizado utilizando un marco experimental común utilizando el planificador *CBP*, en el que se han implementado tanto las distintas heurísticas como los distintos algoritmos.

Las conclusiones principales son las siguientes:

- Respecto a las heurísticas numéricas:

  1. El análisis y y la posterior definición unificación de las heurísticas realizado indican que aunque algunas heurísticas son proceduralmente distintas, salvo resolución de ciertos empates, se pueden considerar equivalentes porque su definición declarativa coincide. Este resultado refuerza la idea de que las definiciones declarativas son tan necesarias como las procedurales.
  2. Las características que marcan la principal diferencia entre las heurísticas basadas en ignorar los efectos negativos de las acciones se pueden resumir en dos: cómo hacen la selección de acciones y cuántas veces tienen en cuenta cada acción. Todas las heurísticas que ignoran totalmente los efectos negativos de las acciones utilizan información del mismo tipo para la selección de acciones e ignoran ciertas interacciones positivas entre (sub)metas. En este sentido presentan carencias similares. Lo que varía es la forma en que esta información se agrega en las precondiciones de las acciones. Hay principalmente tres formas de agregar: mediante una suma, un máximo o una unión.
  3. Los resultados experimentales sugieren que es prácticamente imposible que una misma forma de agregar muestre siempre mejor comportamiento que las demás en un determinado dominio. El motivo es que esto puede depender no sólo de características propias del dominio, sino de la distribución de costes propia del problema.
  4. Las heurísticas que generalmente ofrecen mejores resultados tanto en calidad como en problemas resueltos y tiempo empleado son las heurísticas h_{level}, tanto con propagación con el máximo como con propagación aditiva y la heurística h_{add} utilizando acciones *helpful*.

- Respecto a las heurísticas no numéricas:

  1. Los resultados experimentales sugieren que el uso de acciones *helpful* tal y como se han planteado para planificación basada en costes es útil para mejorar la escalabilidad de los algoritmos contando con información no solo relacionada con la causalidad sino también con los costes de las acciones.
  2. Los resultados experimentales sugieren que el uso de estados *lookahead* tal y como se ha planteado es útil para mejorar la escalabilidad de los algoritmos, permitiendo conseguir un compromiso adecuado entre tiempo y calidad de las soluciones encontradas cuando se utilizan en un algoritmo *anytime*. El uso de estados *lookahead* en planificación basada en costes puede parecer contra-intuitivo, ya que aunque la generación de estos estados está relativamente informada respecto a los costes, su inclusión en un algoritmo de búsqueda tenderá a empeorar la calidad de los planes encontrados, aunque mejorará la escalabilidad del algoritmo. Esto es lo que por lo general ocurría en planificación clásica. En planificación basada en costes las heurísticas numéricas son mucho más variables y la magnitud del error que cometen es más grande debido a que los costes de las acciones pueden ser muy

diferentes entre sí. Esto hace que cualquier mecanismo relativamente informado que permita obtener soluciones de forma avara pueda ser útil para conseguir un buen balance entre calidad, escalabilidad y tiempo.

- Respecto a los algoritmos de búsqueda:

  1. Los resultados muestran que el algoritmo BB-BFS-AHA-L, que es un algoritmo de tipo *mejor primero* con mecanismos para dar prioridad a las acciones *helpful* y utilizar estados *lookahead*, presenta una escalabilidad muy buena, mejorando a los esquemas basados en búsqueda local como *Cost-Enforced Hill-Climbing* (CEHC). BB-BFS-AHA-L encuentra además los planes de más calidad en la mayoría de los dominios, tanto cuando el tiempo es escaso, como cuando se cuenta con más tiempo. Presenta un comportamiento considerablemente mejor que LPG y METRIC-FF. En algunos dominios se comporta mejor o de forma similar que LAMA, el planificador que ganó la última competición. Por lo tanto, se puede afirmar que las heurísticas numéricas basadas en ignorar los efectos negativos de son actualmente útiles en planificación basada en costes, y en combinación con otras heurísticas permiten obtener resultados competitivos.
  2. En los dominios con muchas interacciones entre (sub)metas, el uso de estados *lookahead* es menos adecuado y se obtienen resultados más competitivos con el mismo algoritmo de búsqueda pero sin incorporar estados *lookahead* (es decir, con BB-BFS-AHA) y con CEHC.
  3. El uso del esquema basado en *Hill-Climbing* que se ha propuesto no parece aportar ventajas sobre CEHC.

10.1. Aportaciones

Las principales aportaciones de la tesis son las siguientes:

1. La heurística h_{level} (Fuentetaja et al., 2006, 2008).
2. La comparación tanto teórica como experimental de ésta con otras heurísticas (Fuentetaja et al., 2008, 2009b).
3. Un marco teórico común (definición general) que permite definir las heurísticas basadas en ignorar total o parcialmente los efectos negativos de las acciones (Fuentetaja et al., 2009b).
4. La adaptación de otras heurísticas a planificación basada en costes, como son:
   - La heurística de la dificultad.
   - La heurística de las acciones *helpful*.
   - La heurística *lookahead*.
5. La adaptación y aplicación de tres tipos de algoritmos de búsqueda en planificación basada en costes y la valoración experimental de los mismos y algunas variantes, combinados con distintas heurísticas numéricas (Fuentetaja et al., 2009a).

6. El planificador CBP (Fuentetaja et al., 2009a; Fernández et al., 2007; Borrajo et al., 2005).

10.2. Publicaciones

Las siguientes publicaciones están relacionadas con el desarrollo de esta tesis:

Título:	A Look-ahead B&B Search for Cost-Based Planning
Autores:	Raquel Fuentetaja, Daniel Borrajo y Carlos Linares López
Publicación:	Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA)
Año:	2009

Título:	A Unified View of Cost-Based Heuristics
Autores:	Raquel Fuentetaja, Daniel Borrajo y Carlos Linares López
Publicación:	Workshop on Heuristics for Domain-independent Planning. Workshop of the International Conference on Automated Planning and Scheduling. ICAPS 2009
Año:	2009

Título:	A New Approach to Heuristic Estimations for Cost-Based Planning
Autores:	Raquel Fuentetaja, Daniel Borrajo y Carlos Linares López
Publicación:	The 21st International FLAIRS Conference
Año:	2008

Título:	PLTOOL: A Knowledge Engineering Tool for Planning and Learning
Autores:	Susana Fernández, Daniel Borrajo, Raquel Fuentetaja, Juan D. Arias y Manuela Veloso
Publicación:	**The Knowledge Engineering Review**
Año:	2007

Título:	Improving Control-Knowledge Acquisition for Planning by Active Learning
Autores:	Raquel Fuentetaja y Daniel Borrajo
Publicación:	17th European Conference on Machine Learning (ECML'06)
Año:	2006

Título:	Improving Relaxed Planning Graph Heuristics for Metric Optimization
Autores:	Raquel Fuentetaja, Daniel Borrajo y Carlos Linares
Publicación:	Heuristic Search, Memory Based Heuristics and its Application. Workshop of The Twenty-First National Conference on Artificial Intelligence. AAAI.
Año:	2006

Título:	Some Active Learning Schemes to Acquire Control Knowledge for Planning
Autores:	Raquel Fuentetaja and Daniel Borrajo
Publicación:	Learning for Search. Workshop of The Twenty-First National Conference on Artificial Intelligence. AAAI.
Año:	2006

Título:	Integrating Action Preconditions Difficulty within the Relaxed Plan Heuristic Measure
Autores:	Tomás de la Rosa y Raquel Fuentetaja
Publicación:	Heuristic Search, Memory Based Heuristics and its Application. Workshop of The Twenty-First National Conference on Artificial Intelligence. AAAI.
Año:	2006

Título:	Tool for automatically acquiring control knowledge for planning
Autores:	Daniel Borrajo, Susana Fernández, Raquel Fuentetaja, Juan D. Arias y Manuela Veloso
Publicación:	International Competition in knowledge Engineering for Planning (ICKEPS) in the **15th International Conference on Automated Planning and Scheduling (ICAPS'05)**
Año:	2005

Capítulo 11

Trabajos futuros

El trabajo que se ha presentado se puede ampliar en varios aspectos:

- Respecto a las heurísticas:
  1. La definición generalizada se puede utilizar para derivar nuevas heurísticas. Sin embargo, probablemente esta línea derivaría en heurísticas que ofrecen comportamiento similar a las anteriores si no se incluye al menos parte de la información de la que carecen. Principalmente interacciones negativas entre (sub)metas.
  2. Respecto a la inclusión de información relacionada con las interacciones negativas entre (sub)metas, sería interesante el estudio de los siguientes aspectos:
     - Orden de las precondiciones: en algunos dominios, ciertas interacciones negativas sirven para determinar en qué orden hay que conseguir las precondiciones de las acciones en un momento determinado. Un ejemplo claro se puede encontrar en los dominios que contienen acciones para transporte. Por ejemplo, en el dominio *Zenotravel* la acción para desembarcar (*debark*) tiene dos precondiciones: (1) que el avión esté en el lugar en que se quiere hacer el desembarco y (2) que la persona esté en el avión. Siempre es necesario conseguir primero (2), embarcando a la persona en el lugar en que se encuentre y luego (1), desplazando el avión al lugar de desembarco. Al relajar, se pierde esta restricción de orden, que en el dominio original viene modelada a través de borrados: si generamos antes (1) que (2), al conseguir (2) se borra (1), ya que hay que mover el avión al lugar en que se encuentra la persona. Sería interesante estudiar cómo incluir información parcial sobre el orden en que se tienen que conseguir las precondiciones en los *RPGs*, y si esta información se puede generar de forma automática, bien sistemáticamente o bien utilizando aprendizaje automático.
     - Cambio de representación: cómo afecta el cambio de representación en el cálculo de las heurísticas numéricas basadas en *RPGs*. Por ejemplo, una característica común a estas heurísticas es que consideran tanto las metas iniciales como las precondiciones de cada acción como si fueran independientes. Un

cambio de representación posible consiste transformar el dominio de manera que cada acción tenga una única precondición. Esto supone generar predicados nuevos con más aridad que los iniciales por un lado y por otro lado gestionar la generación de estos predicados en los efectos de las acciones. Sería inviable utilizar esta transformación para el proceso de búsqueda, ya que el proceso de instanciación daría lugar en general a un número de proposiciones demasiado alto. Sin embargo, sería interesante estudiar si tiene sentido utilizar esta representación en el cálculo de la heurística sin que sea necesario instanciarla en una etapa de preproceso.

3. Por otro lado, en relación con los estados *lookahead*, sería interesante estudiar algún mecanismo para evitar que se produzcan problemas de continuidad cuando se generan varios estados *lookahead* consecutivos.
4. Estudiar la relación, tanto teórica como experimental, entre las heurísticas que se han tratado en esta tesis y otras heurísticas más recientes como la heurística del grafo causal o las basadas en el estudio de *landmarks*.
5. Estudiar mecanismos para permitir acciones con coste dependiente del estado en el cálculo de las heurísticas mediante *RPGs*.

- Respecto a los algoritmos de búsqueda, hay una cantidad relativa de variantes por experimentar. Por ejemplo:

  1. En el *Branch and Bound* es posible podar utilizando $f(n)$ en lugar de $g(n)$. Podar por $f(n)$ implica perder la completitud del algoritmo, pero habría que valorar experimentalmente de una forma más exhaustiva la influencia que esto tiene repecto a la escalabilidad y al coste de las soluciones encontradas.
  2. Se podrían realizar experimentos con distintas ω o utilizar un esquema dinámico. Una posibilidad es que ω sea $h_{ops}(n)$, es decir, la estimación heurística en términos de número de operadores del estado evaluado. De esta manera, cuantos más operadores se estima que faltan por aplicar, mayor es el valor de ω, lo cual suele implicar que la búsqueda sea más avara.
  3. Valorar el comportamiento de realizar una aproximación estocástica en el tratamiento de las acciones *helpful* en BB-BFS. En lugar de dar prioridad absoluta a la lista ABIERTA, que contiene únicamente sucesores generados por acciones *helpful*, respecto a la SECUNDARIA, que contiene los sucesores no *helpful*, se puede extraer en cada iteración y con probabilidad ϵ un nodo de la lista *SECUNDARIA*.
  4. Comparar con otras técnicas que se han utilizado en planificación clásica para incluir las acciones *helpful* con prioridad relativa.
  5. Incluir en la experimentación dominios con estados sin salida, por ejemplo los utilizados en la última competición.
  6. Valorar la posibilidad de conseguir robustez en más dominios mediante el uso de una combinación de los algoritmos CEHC, BB-BFS-AHA y L-BB-BFS-AHA.

Apéndice A

Datos experimentales complementarios

En este apéndice se incluyen datos experimentales complementarios a algunos experimentos que se han presentado en el cuerpo de la tesis. Concretamente, los experimentos:

- Experimento 2: heurísticas en CEHC (página 170),
- Experimento 6: HC-B vs. CEHC (página 183), y
- Experimento 8: BFS puro vs. BFS-AHA (página 199).

En cada caso, se incluyen para cada dominio los resultados relativos a: coste de las soluciones encontradas, nodos evaluados y tiempo de CPU. Además, respecto al coste de la solución se muestran también las gráficas clásicas en las que se representa el coste de la solución encontrada para cada uno de los problemas. Como se mencionó en el capítulo 8, sección 8.1.2.2, en algunas ocasiones estas gráficas son útiles para interpretar las gráficas de probabilidad.

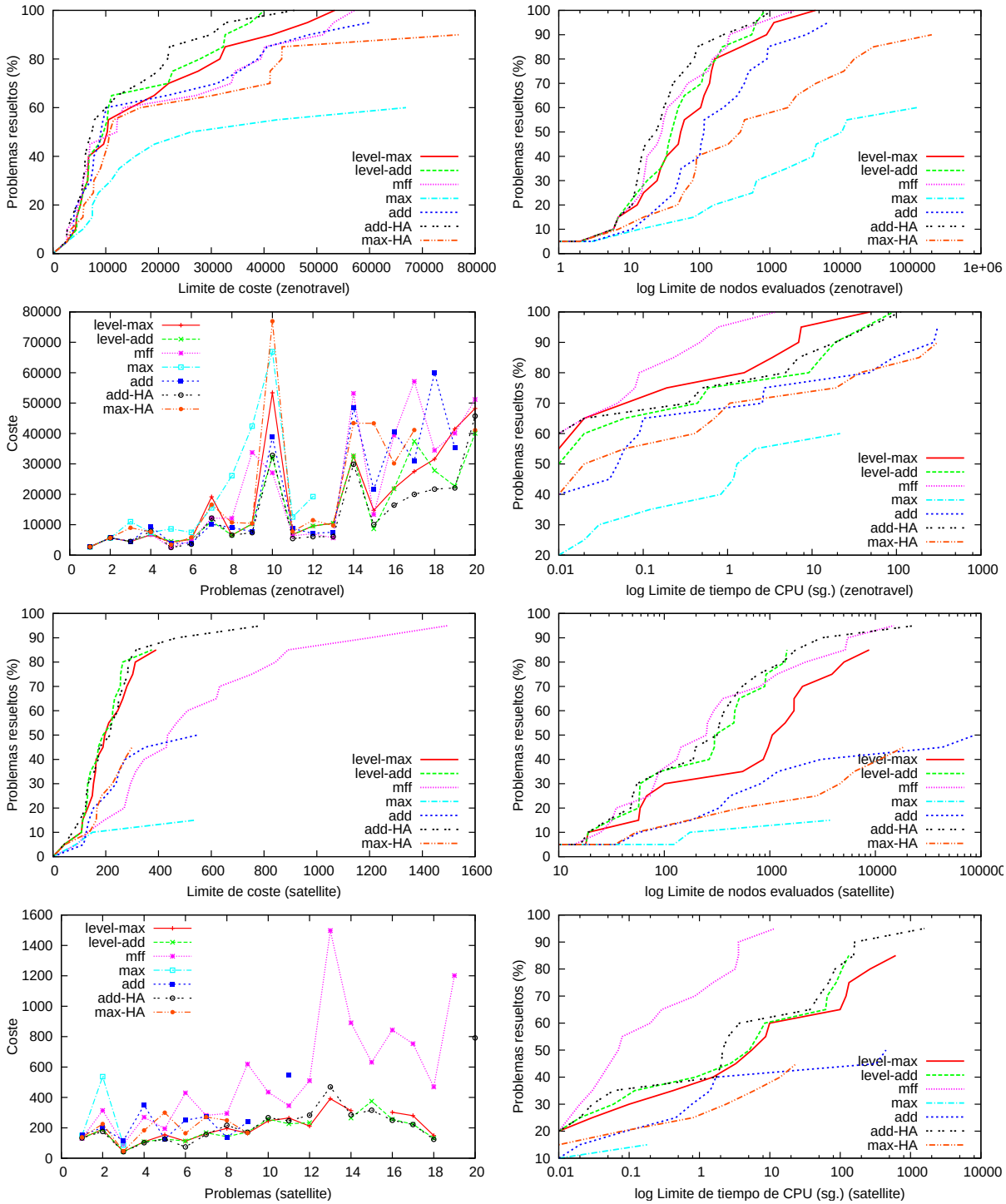

Figura A.1: Experimento 2: comparación entre distintas heurísticas en CEHC (I).

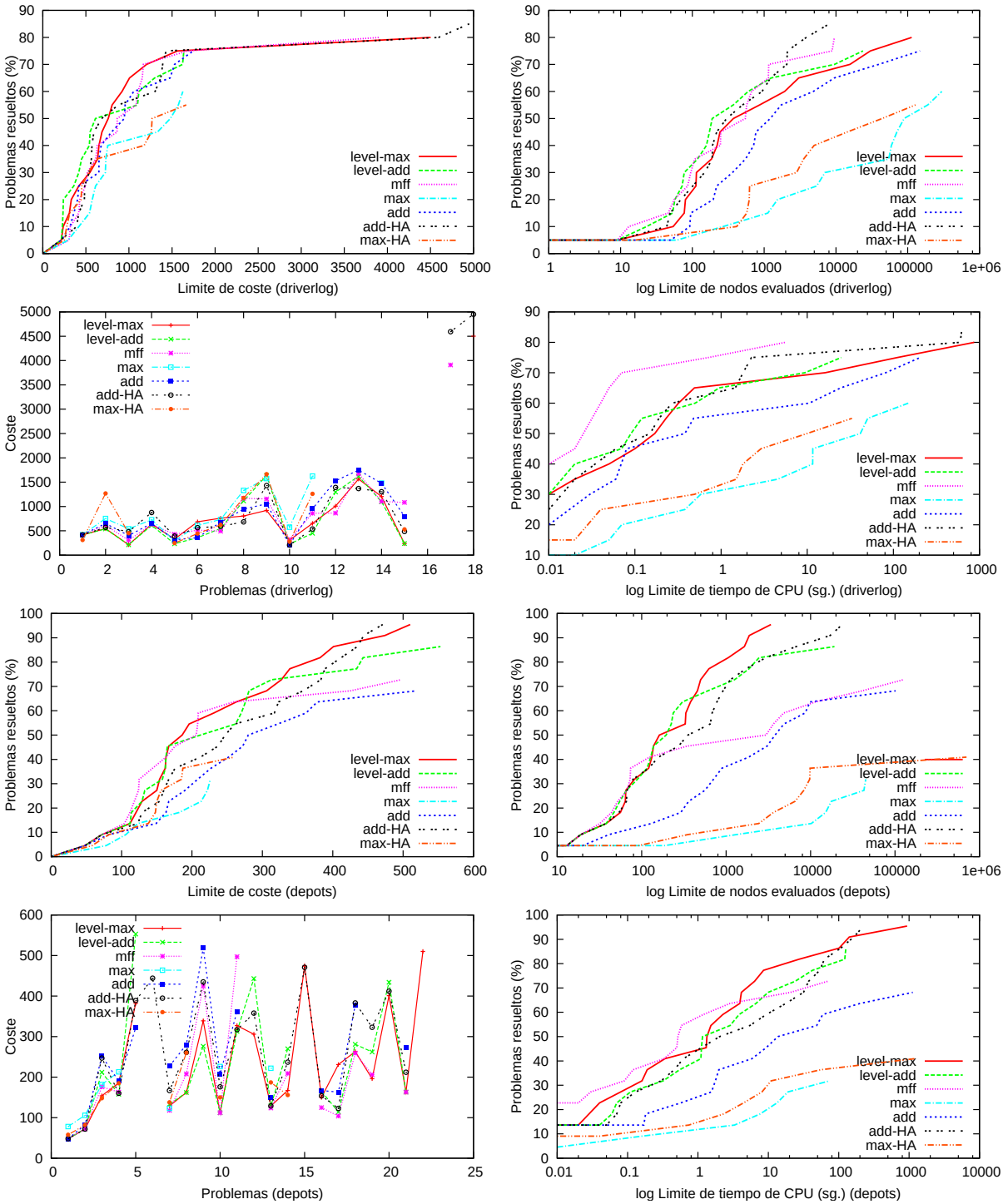

Figura A.2: Experimento 2: comparación entre distintas heurísticas en CEHC (II).

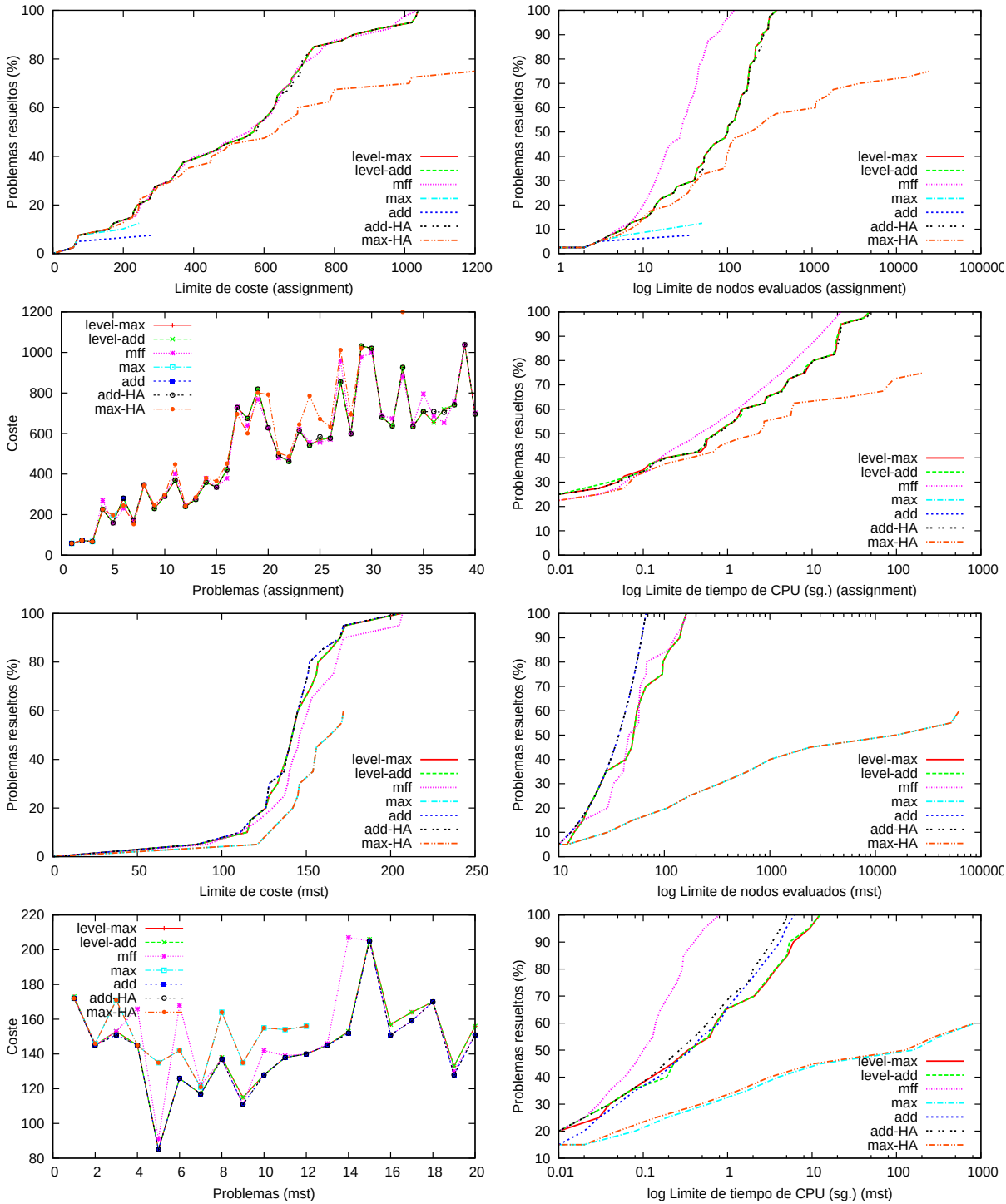

Figura A.3: Experimento 2: comparación entre distintas heurísticas en CEHC (III).

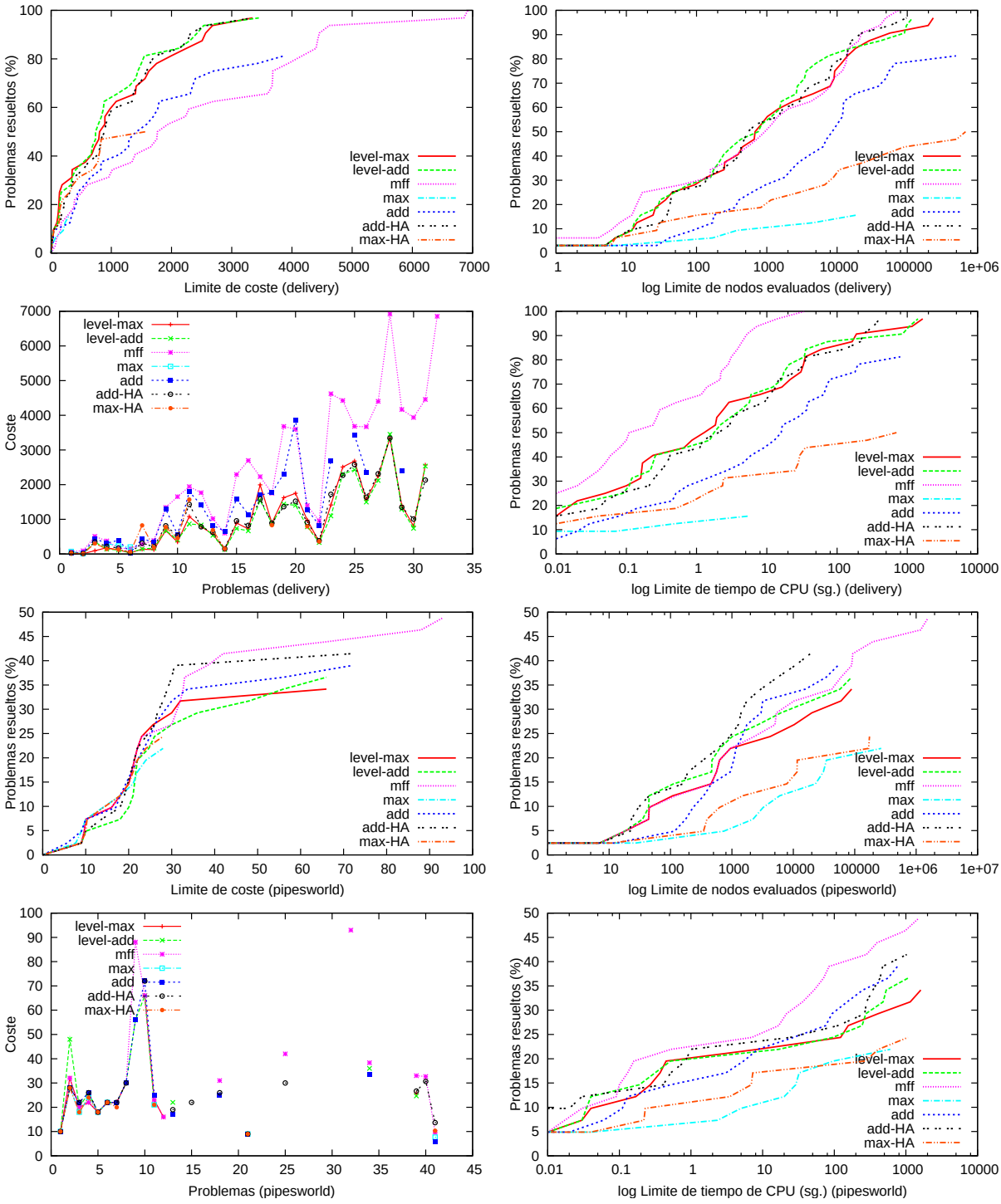

Figura A.4: Experimento 2: comparación entre distintas heurísticas en CEHC (IV).

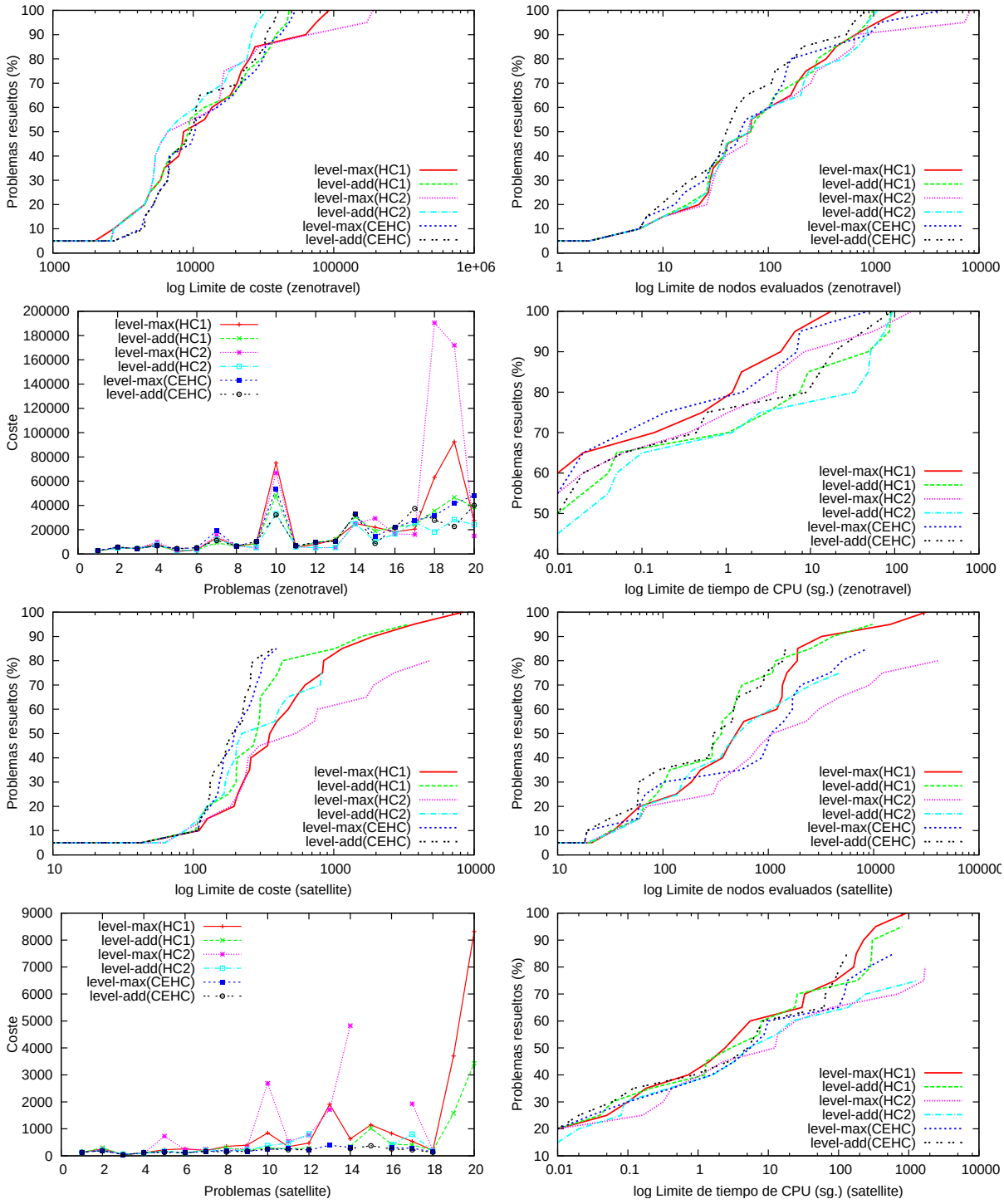

Figura A.5: Experimento 6: comparación entre HC y CEHC (I).

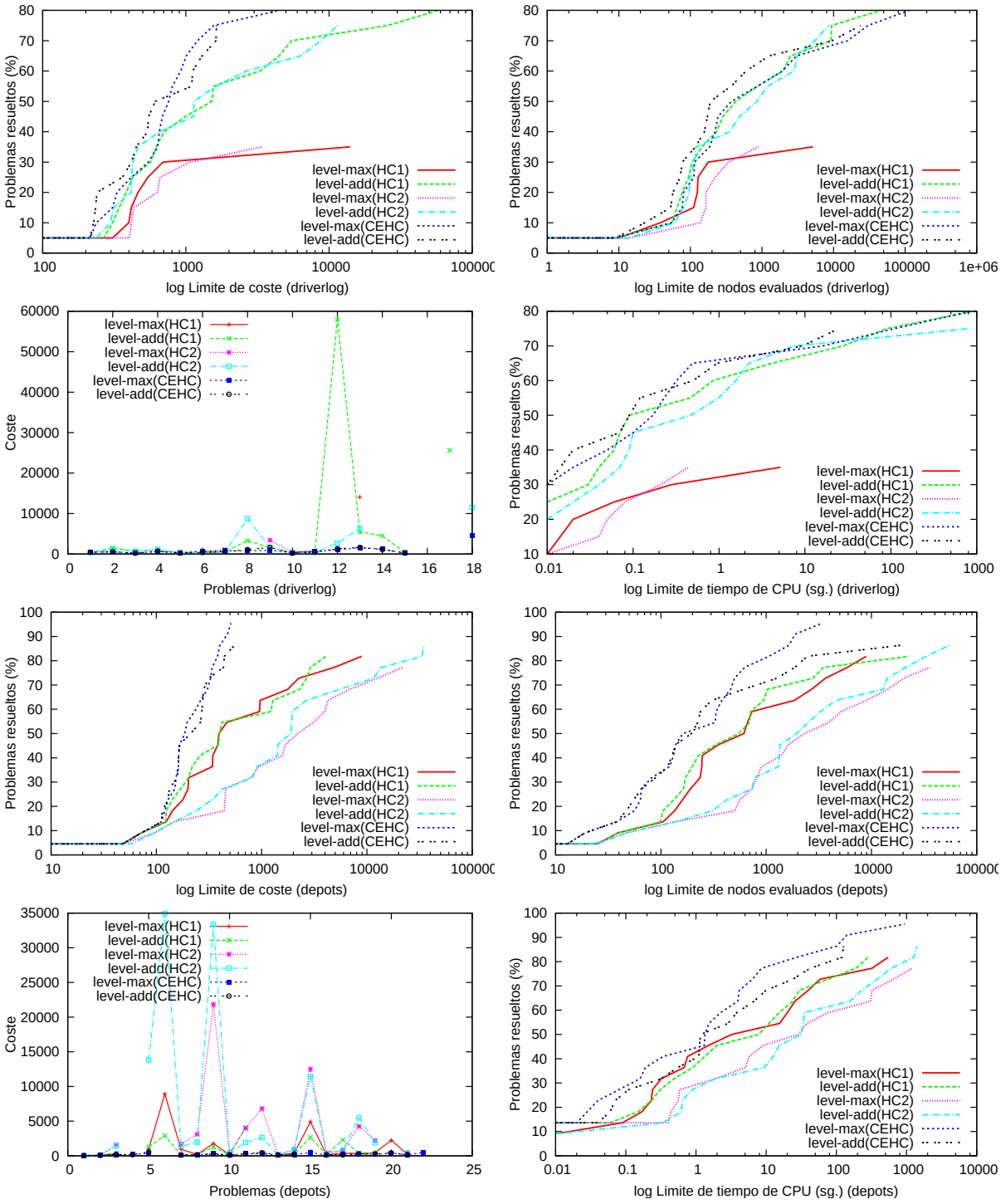

Figura A.6: Experimento 6: comparación entre HC y CEHC (II).

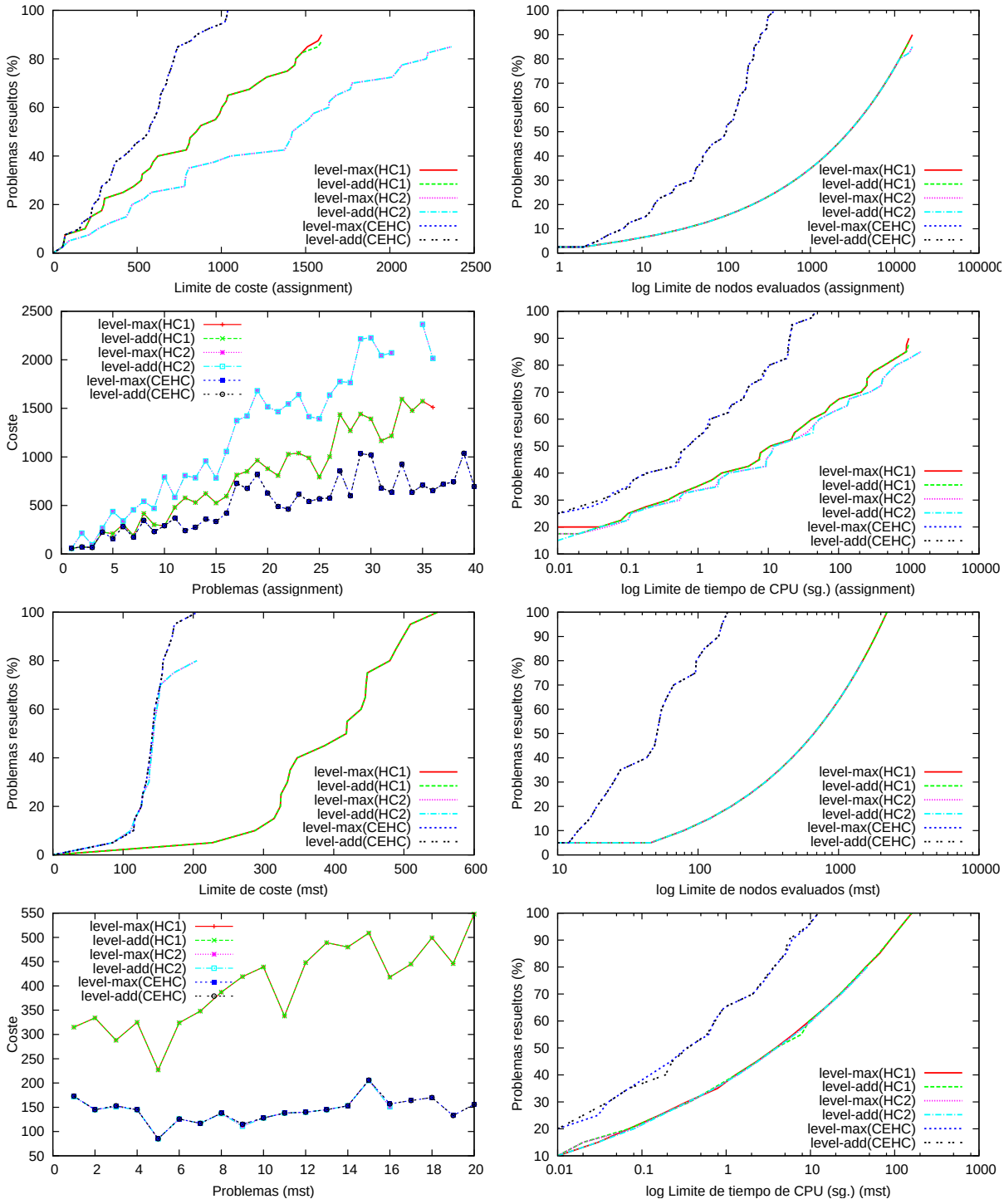

Figura A.7: Experimento 6: comparación entre HC y CEHC (III).

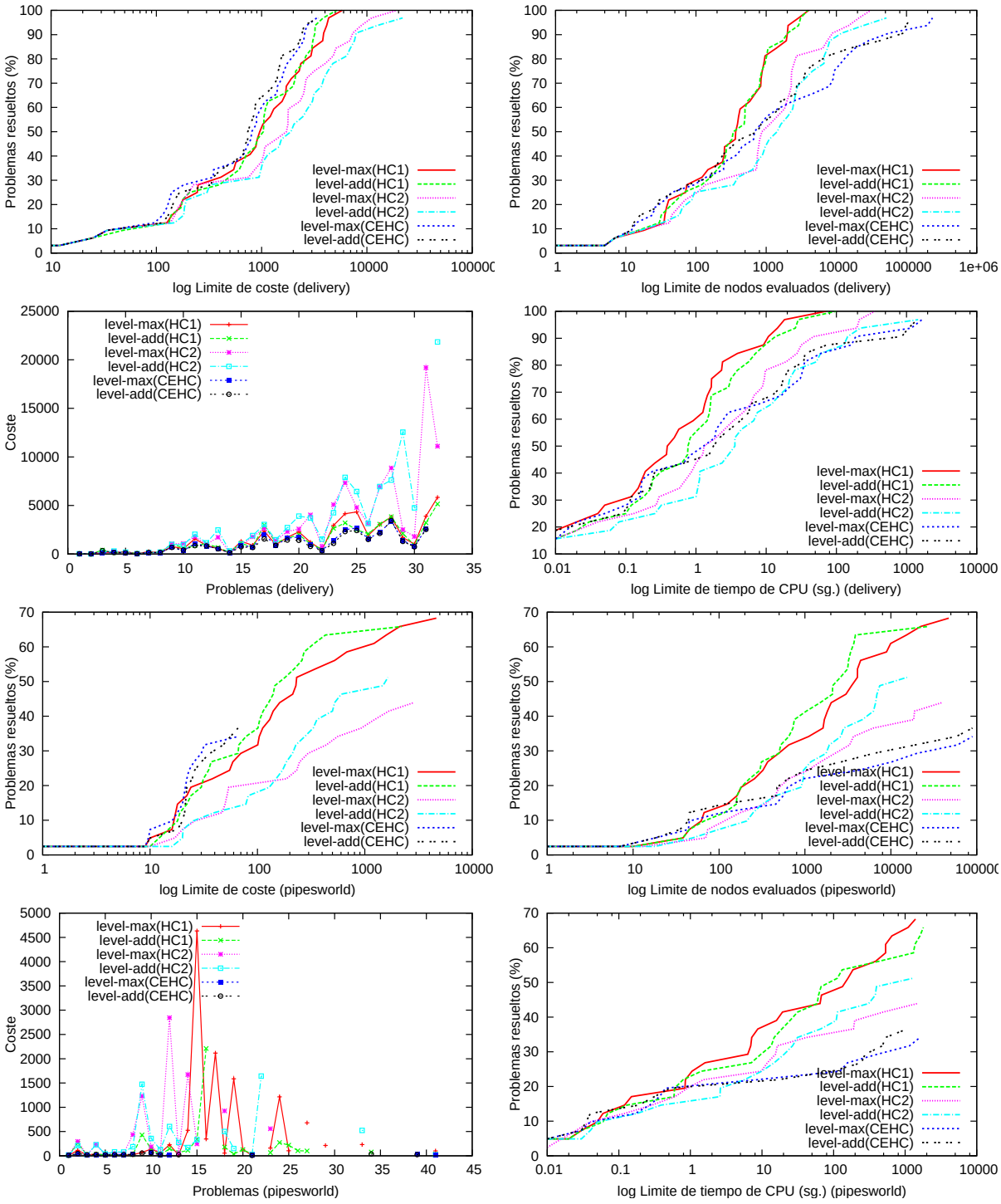

Figura A.8: Experimento 6: comparación entre HC y CEHC (IV).

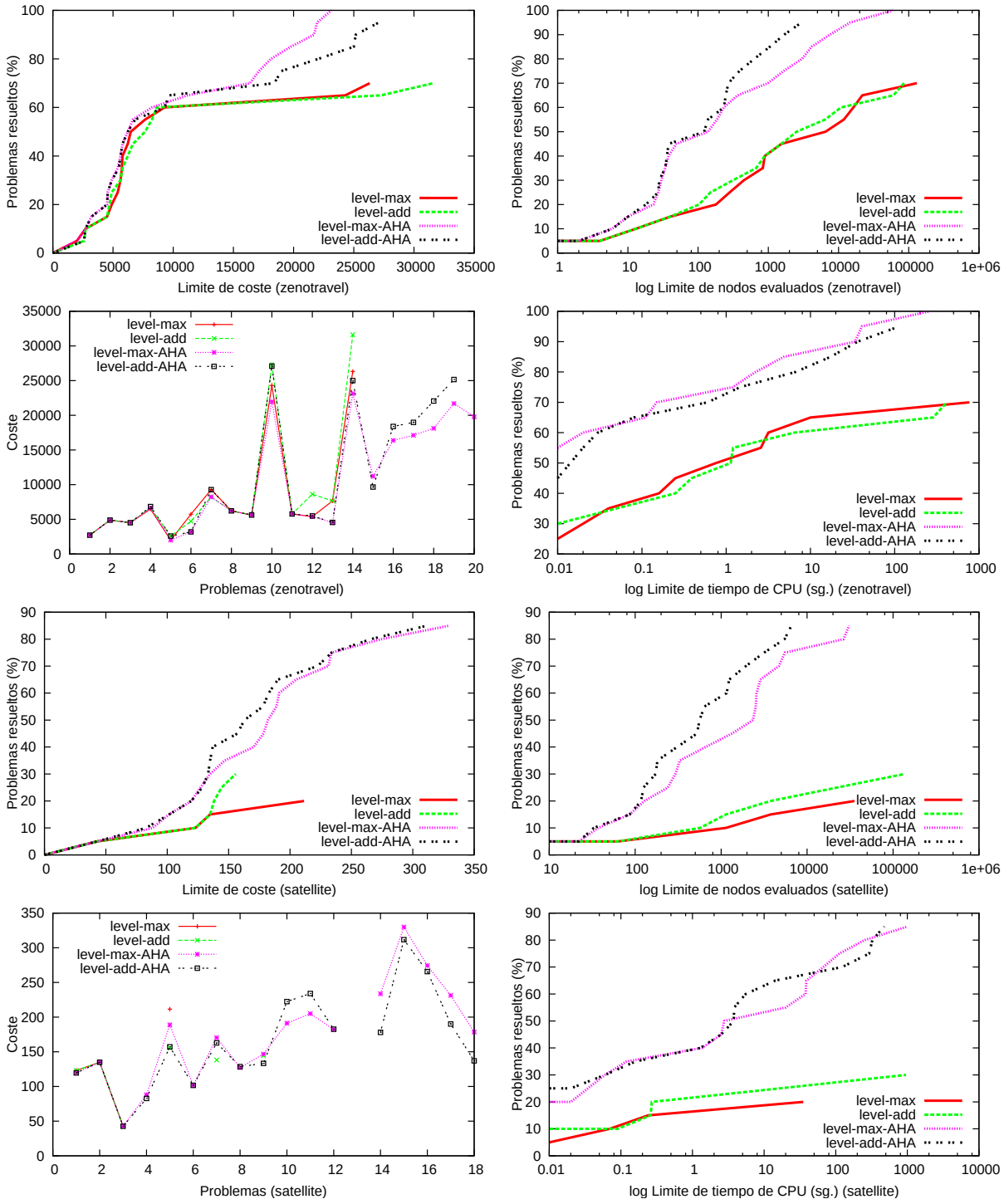

Figura A.9: Experimento 8: comparación entre BFS y BFS-AHA (I).

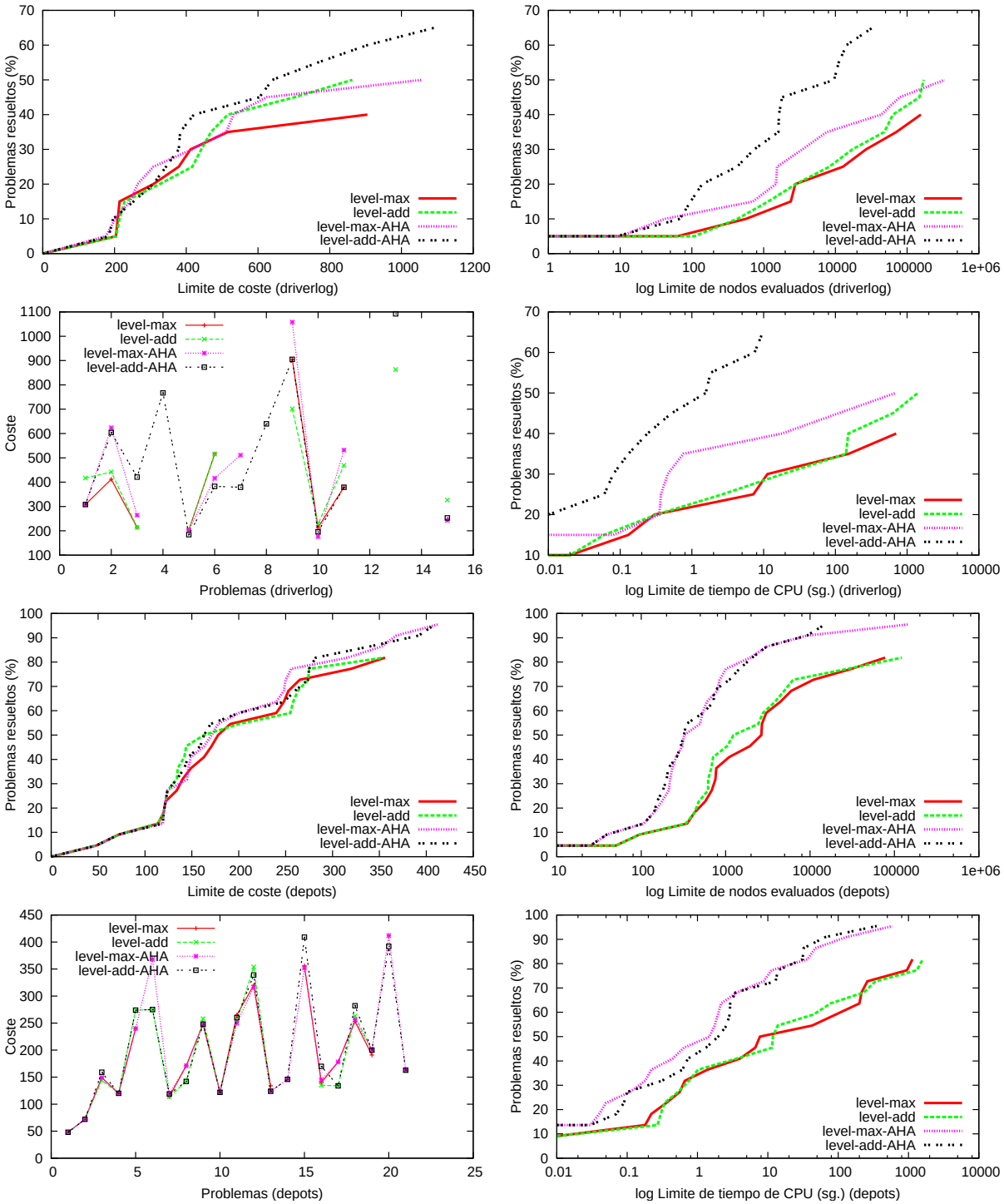

Figura A.10: Experimento 8: comparación entre BFS y BFS-AHA (II).

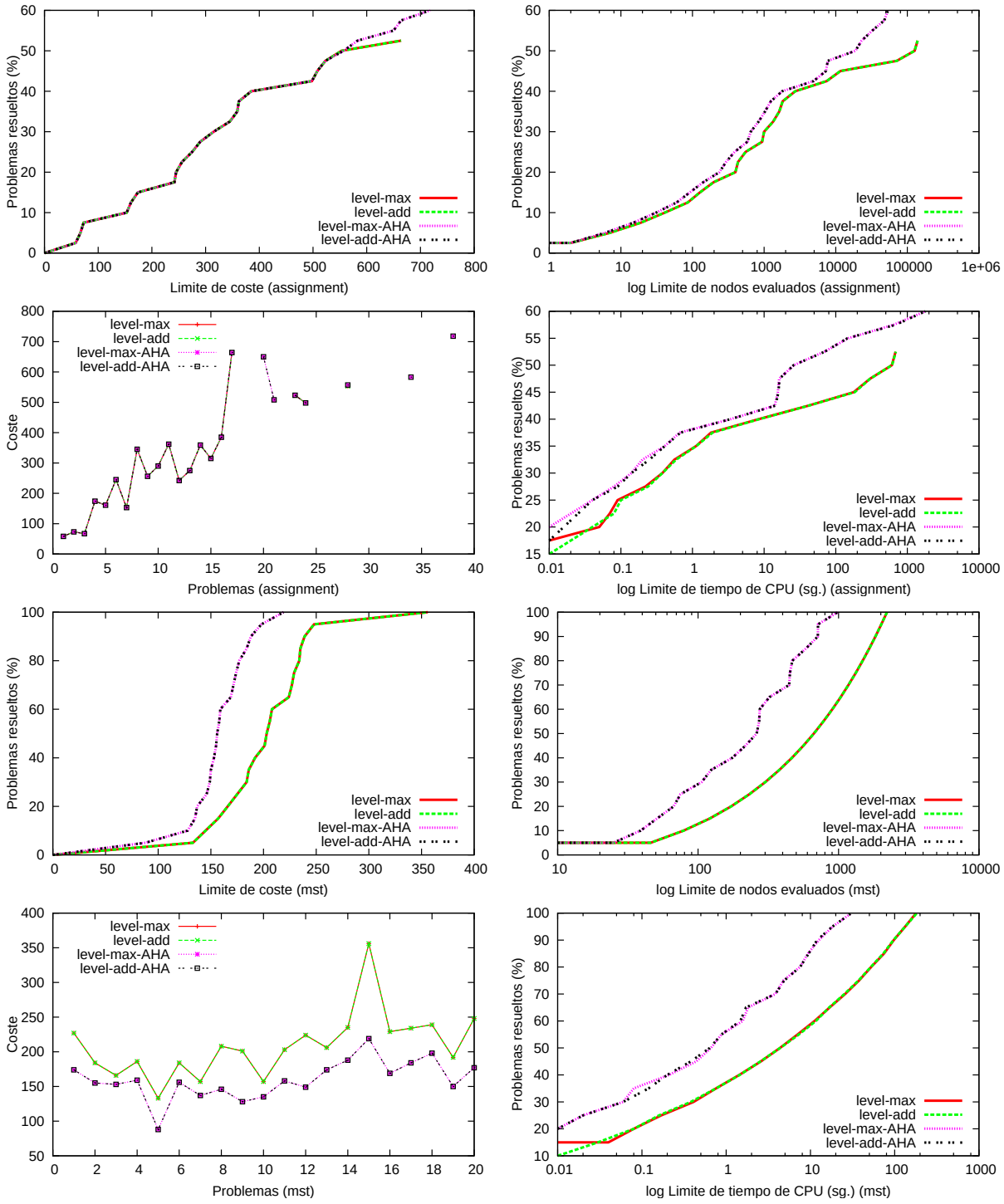

Figura A.11: Experimento 8: comparación entre BFS y BFS-AHA (III).

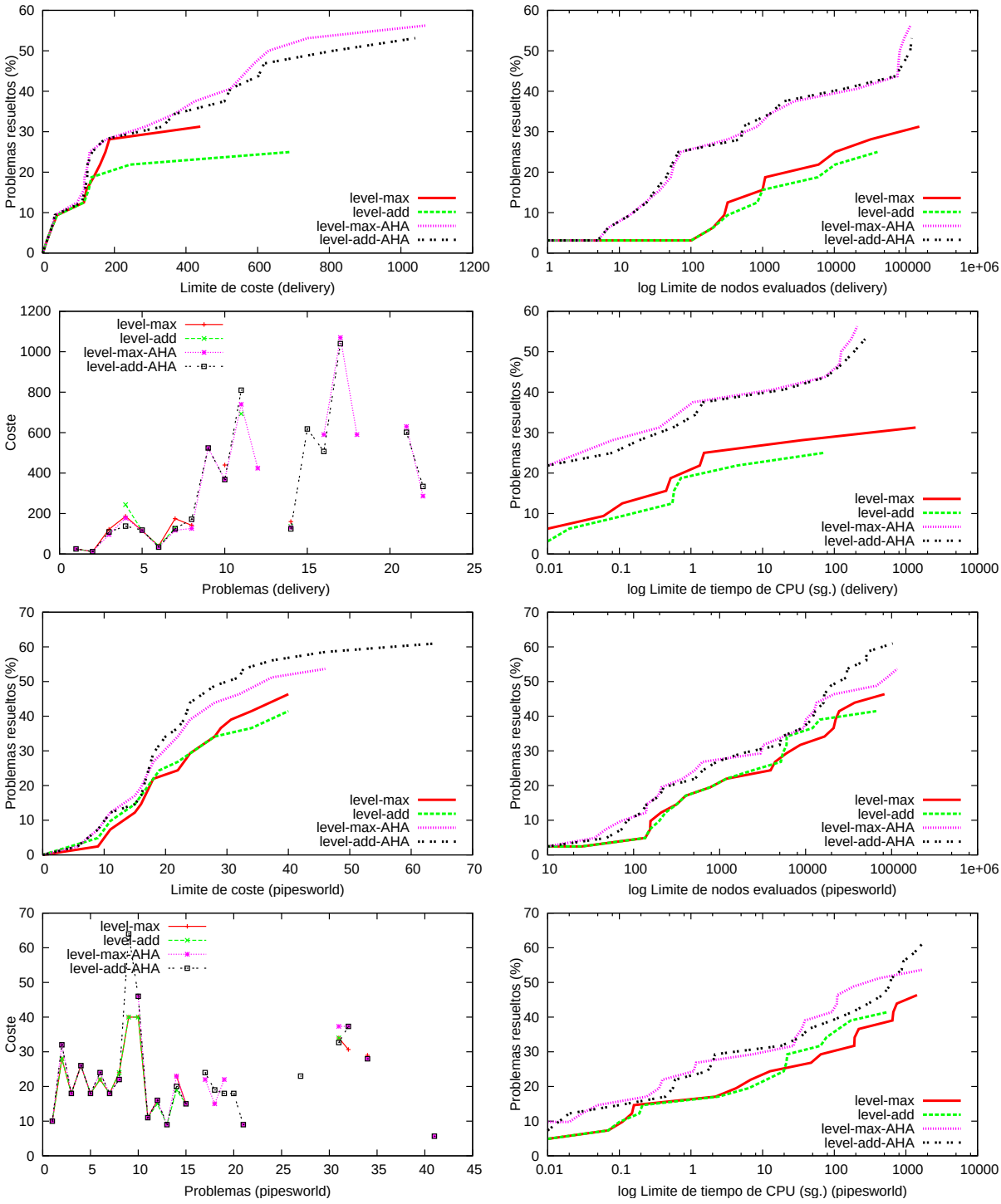

Figura A.12: Experimento 8: comparación entre BFS y BFS-AHA (IV).

Bibliografía

Adam, F. and Sammon, D. (2004). *The Enterprise Resource Planning Decade: lessons Learned and Issues for the Future*. Idea Group Publishing, Hershey, PA, USA.

Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence (AI)*, 116(1-2):123–191.

Backstrom, C. (1992). *Computational Complexity of Reasoning about Plans*. PhD thesis, Linkoping University, Linkoping, Sweden.

Baier, J. (2007). Improving relaxed-plan-based heuristics. In *Proceedings of the Workshop on Heuristics for Domain-Independent Planning. International Conference on Automated Planning and Scheduling (ICAPS)*.

Baier, J. A. and Botea, A. (2009). Improving planning performance using low-conflict relaxed plans. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, Thessaloniki, Greece.

Beek, P. V. and Chen, X. (1999). CPlan: a constraint programming approach to planning. In *Proceedings of the 16th American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 585–590.

Benton, J., van den Briel, M., and Kambhampati, S. (2007). A hibrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 34–41.

Blizard, W. D. (1989). Multiset theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66.

Blum, A. L. and Furst, M. L. (1995). Fast planning through planning graph analysis. In Mellish, C. S., editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, pages 1636–1642, Montréal, Canada. Morgan Kaufmann.

Bonet, B. and Geffner, H. (1999). Planning as heuristic search: new results. In *Proceedings of the European Conference on Planning (ECP)*.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence (AI)*, 129(1-2):5–33.

Bonet, B., Loerincs, G., and Geffner, H. (1997). A robust and fast action selection mechanism for planning. In *Proceedings of the American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 714–719. MIT Press.

Borrajo, D., Fernández, S., Fuentetaja, R., Arias, J. D., and Veloso, M. (2005). Tool for automatically acquiring control knowledge for planning. In *International Competition in Knowledge Engineering for Planning (ICKEPS). In the 15th International Conferenceon Automated Planning and Scheduling.*

Borrajo, D. and Veloso, M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 11(1-5):371–405. Also in the book Lazy Learning, David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

Bryce, D. and Kambhampati, S. (2006). How to skin a planning graph for fun and profit: a tutorial on planning graph based reachability heuristics. Technical report, Arizona State University.

Bryce, D. and Kambhampati, S. (2007). How to skin a planning graph for fun and profit: a tutorial on planning graph based reachability heuristics. *AI Magazine*, 28 No. 1:47–83.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence (AI)*, 69(1-2):165–204.

Bäckström, C. and Nebel, B. (1995). Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655.

Castillo, L., Fdez.-Olivares, J., García-Pérez, O., and Palao, F. (2006). Bringing users and planning technology together. Experiences in SIADEX. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS).*

Chen, Y., Hsu, C., and Wah, B. (2006). Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)*, 26:323–369.

Coles, A. I., Fox, M., Long, D., and Smith, A. J. (2008). A hybrid relaxed planning graph-LP heuristic for numeric planning domains. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS).*

Currie, K. and Tate, A. (1991). O-Plan: the open planning architecture. *Artificial Intelligence (AI)*, 52(1):49–86.

Davis, H., Bramanti-Gregor, A., and Wang, J. (1988). The advantages of using depth and breadth components in heuristic search. *Methodologies for Intelligent Systems*, 3:19–28.

de la Rosa, T., Jiménez, S., and Borrajo, D. (2008). Learning relational decision trees for guiding heuristic planning. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS).*

Do, M. B. and Kambhampati, S. (2000). Solving planning-graph by compiling it into CSP. In *Artificial Intelligence Planning Systems*, pages 82–91.

Do, M. B. and Kambhampati, S. (2003). Sapa: a scalable multi-objective heuristic metric temporal planner. *Journal of Artificial Intelligence Research (JAIR)*, 20:155–194.

Edelkamp, S. (2001). Planning with pattern databases. In *Proceedings of the European Conference on Planning (ECP)*.

Edelkamp, S. (2003). Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124.

Edelkamp, S. and Helmert, M. (2000). On the implementation of MIPS. In *AIPS Workshop on Model-Theoretic Approaches to Planning*, pages 18–25.

Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence (AI)*, 62(2):255–301.

Fernández, S., Borrajo, D., Fuentetaja, R., Arias, J. D., and Veloso, M. (2007). PLTOOL: a knowledge engineering tool for planning and learning. *The Knowledge Engineering Review*, 22(2):153–184.

Fikes, R. and Nilsson, N. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence (AI)*, 2:189–208.

Fox, M. and Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research (JAIR)*, 9:367–421.

Fox, M. and Long, D. (1999). The detection and exploitation of symmetry in planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 956–961.

Fox, M. and Long, D. (2001). STAN4: a hybrid planning strategy based on subproblem abstraction. *AI Magazine*, 22(3):102–111.

Fox, M. and Long, D. (2003). PDDL2.1: an extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124. ISSN 11076-9757.

Fuentetaja, R., Borrajo, D., and Linares, C. (2006). Improving relaxed planning graph heuristics for metric optimization. In *Workshop on Heuristic Search, Memory Based Heuristics and its Applications. American Association for the Advancement of Artificial Intelligence Conference (AAAI)*. AAAI Press.

Fuentetaja, R., Borrajo, D., and Linares, C. (2008). A new approach to heuristic estimations for cost-based planning. In *Proceedings of the 21st International FLAIRS Conference*, pages 543–548.

Fuentetaja, R., Borrajo, D., and Linares, C. (2009a). A look-ahead B&B search for cost-based planning. In *Proceedings of the 13th Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, pages 105–114.

Fuentetaja, R., Borrajo, D., and Linares, C. (2009b). A unified view of cost-based heuristics. In *Workshop on Heuristics for Domain-independent Planning. Workshop of the International Conference on Automated Planning and Scheduling. ICAPS 2009.*

Geffner, H. (2002). Perspectives on Artificial Intelligence planning. In *American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 1013–1023. AAAI/MIT Press. Invited Talk.

Geffner, H. (2007). The causal graph heuristic is the additive heuristic plus context. In *Proceedings of the Workshop on Heuristics for Domain-Independent Planning. International Conference on Automated Planning and Scheduling (ICAPS).*

Gerevini, A., Saetti, A., and Serina, I. (2004). Planning with numerical expressions in LPG. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 667–671.

Gerevini, A., Saetti, A., and Serina, I. (2008). An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence (AI)*, 172:899–944.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning - Theory and Practice.* Morgan Kaufmann Publishers, San Francisco, CA 94111.

Green, C. (1969). Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 219–239, San Mateo, CA. Morgan Kaufmann.

Hansen, E. A. and Zhou, R. (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 28:267–297.

Haslum, P., Bonet, B., and Geffner, H. (2005). New admissible heuristics for domain-independent planning. In *Proceedings of the 20th American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 1163–1168, Pittsburgh, USA.

Haslum, P. and Geffner, H. (2000). Admissible heuristics for optimal planning. In *Artificial Intelligence Planning Systems*, pages 140–149.

Haslum, P. and Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning (ECP).*

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 161–170.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246.

Helmert, M. and Geffner, H. (2008). Unifying the causal graph and additive heuristics. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS).*

Hoffmann, J. (2003). The METRIC-FF planning system: translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)*, 20:291–341.

Hoffmann, J. (2005). Where 'ignoring delete lists' works: local search topology in planning benchmarks. *Journal of Artificial Intelligence Research (JAIR)*, 24:685–758.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302.

Holte, R. C., Newton, J., Felner, A., Meshulam, R., and Furcy, D. (2004). Multiple pattern databases. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 122–131.

Joslin, D., Frank, J., Jónsson, A. K., and Smith, D. E. (2005). Simulation-based planning for planetary rover experiments. In *Proceedings of the 37th Conference on Winter Simulation*, pages 1049–1058. Winter Simulation Conference.

Kautz, H. and Selman, B. (1999). Unifying SAT-based and graph-based planning. In Minker, J., editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland. Computer Science Department, University of Maryland.

Kautz, H. A. and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, pages 359–363.

Keyder, E. and Geffner, H. (2007). Heuristics for planning with action costs. In *Proceedings of the 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, pages 140–149. Springer.

Keyder, E. and Geffner, H. (2008). Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, pages 588–592.

Knoblock, C. A. (1994). Generating parallel execution plans with a partial-order planner. In *Proceedings of the 2nd Conference on Artificial Intelligence Planning Systems (AIPS)*, San Mateo, California. Morgan Kaufmann.

Korf, R. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence (AI)*, 27(1):97–109.

Liu, Y., Koenig, S., and Furcy, D. (2002). Speeding up the calculation of heuristics for heuristic search-based planning. In *Proceedings of the American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 484–491.

McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the 9th American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, volume 2, pages 634–639, Anaheim, California, USA. AAAI Press/MIT Press.

Mccarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of Artificial Intelligence. *Machine Inteligence*, 4:463–502.

Mcdermott, D. (1991). Using regression-match graphs to control search in planning. *Artificial Intelligence (AI)*, 109((1-2)):111–159.

McDermott, D. (1996). A heuristic estimator for means-ends analysis in planning. In *Proceedings of the 3rd Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 142–149. AAAI Press.

Minton, S. (1988). *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA.

Mirkis, V. and Domshlak, C. (2007). Cost-sharing approximations for h^+. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 240–247.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Mur, J. W., and Wu, D. (2003). SHOP2: an HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404.

Nau, D. S. (2007). Current trends in AI planning. *AI Magazine*, 28(4):47–58.

Newell, A. and Simon, H. A. (1995). GPS, a program that simulates human thought. *Computers & thought*, pages 279–293.

Nguyen, X. and Kambhampati, S. (2000). Extracting effective and admissible heuristics from the planning graph. In *Proceedings of the American Association for the Advancement of Artificial Intelligence Conference (AAAI)*.

Nguyen, X. and Kambhampati, S. (2001). Reviving partial order planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 459–466.

Nigenda, R. S., Nguyen, X., and Kambhampati, S. (2000). AltAlt: combining the advantages of graphplan and heuristic state search. Technical report, Arizona State University.

Onaindia, E., Sapena, O., Sebastia, L., and Marzal, E. (2001). SimPlanner: an execution-monitoring system for replanning in dynamic worlds. In *Proceedings of EPIA*, volume 2258, pages 393–400.

Pednault, E. P. D. (1994). ADL and the state-transition model of action. *Journal of Logic and Computation*, 4(5):467–512.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: a sound, complete, partial order planner for ADL. In Nebel, B., Rich, C., and Swartout, W., editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 103–114, San Mateo, California. Morgan Kaufmann.

Polh, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 12–17.

Porteous, J., Sebastia, L., and Hoffmann, J. (2001). On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP)*, pages 37–48, Toledo, Spain.

Pérez, A. (1996). Representing and learning quality-improving search control knowledge. *Machine Learning*.

Refanidis, I. and Vlahavas, I. (2001). The GRT planning system: backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research (JAIR)*, 15:115–161.

Refanidis, I. and Vlahavas, I. (2003). Multiobjective heuristic state-space planning. *Artificial Intelligence Journal (AIJ)*, 145/1-2:1–32.

Richter, S., Helmert, M., and Westphal, M. (2008). Landmarks revisited. In *Proceedings of the 23rd American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 975–982.

Richter, S. and Westphal, M. (2008). The LAMA planner. Using landmarks counting in heuristic search. In *Short paper for the International Planning Competition*.

Rintanen, J. (2006). Unified definition of heuristics for classical planning. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*, pages 600–604. IOS Press.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: a Modern Approach (2nd edition)*. Prentice Hall.

Sandewall, E. and Rönnquist, R. (1986). A representation of action structures. In Kaufmann, M., editor, *Proceedings of the 5th American Association for the Advancement of Artificial Intelligence Conference (AAAI)*, pages 89–97.

Sapena, O., Onaindia, E., Mellado, M., Correcher, C., and Vendrell, E. (2004). Reactive planning simulation in dynamic environments with VirtualRobot. *Innovations in Applied Artificial Intelligence*, LNCS 3029:699–707.

Sapena, O. and Onaindía, E. (2004). Handling numeric criteria in relaxed planning graphs. In *Advances in Artificial Intelligence. IBERAMIA, LNAI 3315*, pages 114–123.

Smith, D., Frank, J., and Jonsson, A. (2000). Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15(1):61–94.

Smith, D. E. (2004). Choosing objectives in over-subscription planning. In Zilberstein, S., Koehler, J., and Koenig, S., editors, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 393–401.

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. Elsevier Science Inc, New York, USA.

Upal, M. A. (2003). What-if planning in military logistics. In Abraham, A., Frank, K., and Koppen, M., editors, *Proceedings of the Third International Conference on Intelligent Systems Design and Applications*, pages 149–158.

Upal, M. A. and Elio, R. (2000). Learning search control rules versus rewrite rules to improve plan quality. In *Proceedings of the Thirteenth Canadian Conference on Artificial Intelligence*, pages 240–253, New York. Springer-Verlag.

Vancza, J. and Markus, A. (2001). A constraint engine for manufacturing process planning. In *Principles and Practice of Constraint Programming*, pages 745–759.

Vapnyarskii, I. B. (2001). *Encyclopaedia of Mathematics*. Kluwer Academic Publishers.

Veloso, M. and Carbonell, J. (1993). Derivational analogy in PRODIGY: automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278.

Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., and Blythe, J. (1995). Integrating planning and learning: the PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81–120.

Vidal, V. (2004). A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 150–159.

Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4):27–61.

Yoon, S., Fern, A., and Givan, R. (2006). Learning heuristic functions from relaxed plans. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Yoon, S., Fern, A., and Givan, R. (2008). Learning control knowledge for forward search planning. *Machine Learning*, 9:683–718.

Younes, H. and Simmons, R. (2003). VHPOP: versatile heuristic partial order planner. *Journal of Artificial Intelligence Research (JAIR)*.