UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

Razonamiento Basado en Casos Aplicado a la Planificación Heurística

TESIS DOCTORAL

Tomás de la Rosa Turbides

Madrid, 2009

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

Departamento de Informática

Razonamiento Basado en Casos Aplicado a la Planificación Heurística

Tesis para la obtención del grado de DOCTOR en el programa de doctorado de CIENCIA Y TECNOLOGÍA INFORMÁTICA

Autor: Tomás de la Rosa Turbides

Directores: Dr. Daniel Borrajo Millán
Dr. Angel García Olaya

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día ..... de ..... de 2009.

Presidente: D. .....

Vocal: D. .....

Vocal: D. .....

Vocal: D. .....

Secretario: D. .....

Realizado el acto de defensa y lectura de la Tesis el día ..... de ..... de 2009 en .....

Calificación: .....

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A mis padres, Tomás y Josefina

RECUERDOS DE MI MEMORIA

Consejos de siempre y clavado en la conciencia
si acaso no lo ves, levántate, camina y aprende
lecciones de confianza a quien de niño no comprende
que para ser gran sabio hay que tener vasta paciencia
un poco de juicio y lo demás lo da la experiencia
sueños engrandecidos de una juventud risueña
que por haber soñado a buenas horas se empeña
en reclamar como reto el querer saberlo todo
sin saber que en este mundo no puede ser de otro modo
siempre hay mucho que aprender y es la vida quien enseña

Esfuerzos de largos días de puño y letra escribo
cuentan sin florituras el detalle de esta historia
y al final terminan siendo los recuerdos de mi memoria
los que buscan significado a las luchas que a veces vivo
a la hora de recordar tiene que haber algún motivo
latidos del corazón o un tímido sentimiento
y no la experiencia parecida, vacío conocimiento
es por la grata sensación de al luchar sentirse apoyado
por lágrimas o el dolor de los errores del pasado
recordad que el siguiente trabajo no es sólo razonamiento

Coincidiría el lector en que estoy en mi derecho
de recordar a mi manera las piedras del camino
varios años han pasado y ahora al ver que termino
me alegro con orgullo por muy bien haberlo hecho
y si mis propias escrituras un día me son de provecho
sabrás mi sensación de por vivir haber aprendido
fácilmente entenderás el camino que he elegido
hay momentos para recordar lo dicho de forma honrosa
si de mí te vas a acordar acuérdate también de mi prosa
cuando ya no esté por aquí y esté el verso en el olvido

Tomás de la Rosa

Agradecimientos

Emprender grandes proyectos involucra en ciertas ocasiones afrontar sacrificios importantes en la vida. La idea de realizar un doctorado en un país extranjero se antoja de primeras como una ilusión de juventud en la que no se vislumbran los cambios que afectarán a la propia vida. Por eso quiero agradecer muy especialmente a mis padres Tomás y Josefina, a mi hermana Judit y al resto de mi familia por el apoyo incondicional que me han ofrecido para este proyecto. Por su comprensión y por haberme transmitido esa esperanza de éxito que me ha dado fuerza para todos estos años. Agradezco también a todos mis amigos de República Dominicana, en especial a Andrés y Héctor, porque fueron los primeros que me mostraron con acciones y con consejos porqué las oportunidades de la vida hay que aprovecharlas.

Quiero agradecer a mis tutores Daniel y Angel, por los conocimientos, la dedicación y la motivación que me han ofrecido durante el desarrollo de la tesis. Además, a todos los compañeros del grupo PLG, por el agradable ambiente de trabajo que entre todos logramos mantener, en especial a mi compañero Sergio por la mutua motivación bajo el mismo objetivo, y a Raquel por nuestros interminables debates. También quiero agradecer a los nuevos amigos que he conseguido en la universidad, especialmente a Beatriz, a los demás compañeros del grupo CAOS y a Agustín, que con nuestras conversaciones de inverno me ha dado un apoyo especial en esta recta final.

Para terminar, quiero agradecer también a los amigos que como yo, han decidido emprender proyectos similares que involucran sacrificio, pero permiten obtener grandes gratificaciones. En especial a mis más allegados, Tania, Rafael, Nadia, Maylex, Ramón, Jonathan, Alina y Sofía, porque todos han sabido luchar hacia una misma dirección.

Sinopsis

La Planificación Automática es una rama de la Inteligencia Artificial que estudia la construcción de conjuntos o secuencias de acciones, llamadas planes, que permiten transformar el estado de un entorno, con el objetivo de alcanzar las metas de un problema planteado. La planificación heurística es un paradigma dentro de la planificación automática que resuelve los problemas utilizando algoritmos de búsqueda que son guiados por una función de evaluación llamada heurística. Este paradigma ha dado grandes frutos en los últimos años gracias al desarrollo de funciones heurísticas que se pueden construir de forma independiente al dominio de planificación. Los inconvenientes que presentan estas heurísticas son que, por un lado tienen un alto coste computacional, dificultando la resolución de problemas grandes dentro de un tiempo razonable. y por otro lado, la poca información en ciertos tipos de dominios, provocando que en ocasiones los algoritmos busquen infructuosamente una solución.

Por esto, surge la idea de retomar técnicas de aprendizaje automático que en años pasados fueron utilizadas sobre otros paradigmas de planificación, con la idea de mejorar la eficiencia de los planificadores. El objetivo de esta tesis doctoral es desarrollar un sistema de razonamiento basado en casos que sirva para complementar la búsqueda de un planificador heurístico. Se estudia el uso del conocimiento de los casos en diferentes algoritmos de búsqueda, y se evalúa experimentalmente sobre un conjunto de dominios, que por su diversidad, permite validar la técnica. Adicionalmente, se valora el conocimiento aprendido en los casos para establecer relaciones entre la información que puede almacenarse y las mejoras que se pueden obtener en el proceso de planificación.

Abstract

Automated Planning is an Artificial Intelligence field that studies how to build sequences of actions totally or partially ordered. These sequences, called plans, transform the state of the environment with the aim of achieving a given set of goals. Heuristic Planning is a recent planning paradigm that solves problems with search algorithms guided by an evaluation function called heuristic. Heuristic planning is still nowadays one of the top approaches mainly because we can build domain-independent heuristics with an automated procedure. These heuristics have two drawbacks. The first one is related to their computational cost, which imposes size restrictions to the problem being solved. The second one is the poor guidance heuristics give to the algorithms in some domains.

Thereby, part of the research community focuses on applying machine learning techniques used in the past within other planning paradigms. The objective of this dissertation consists of building a case-based reasoning system that supports the search process of a heuristic planner. We will integrate the knowledge given by domain case bases as a search control. The empirical evaluation shows the benefits of the approach. We also analyze the learned knowledge in order to find relations between domain-specific information being gathered and the improvements obtained by the planners in terms of time or plan quality.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

La Inteligencia Artificial (IA) ha buscado desde sus inicios desarrollar sistemas que resuelvan problemas automáticamente. Este objetivo se contrapone a la informática tradicional, que desarrolla programas específicos para los problemas que se le plantean. La IA por su parte, intenta que el esfuerzo que requiere desarrollar un programa de propósito general sea compensado con la posibilidad de plantear los problemas a alto nivel de una forma rápida para que que el sistema pueda resolverlos. La Planificación Automática (PA) crea sistemas de este tipo con el objetivo de hacer razonamiento deliberativo sobre un modelo de las posibles acciones ejecutables en un entorno. Este razonamiento permite sintetizar un conjunto de esas acciones que transforman el entorno desde una situación inicial hasta un nuevo estado en el que se han conseguido las metas que plantea el problema. La PA se ha utilizado exitosamente en aplicaciones reales tales como la planificación de vehículos (*rovers*) de exploración en Marte (Bresina et al., 2005), la extinción de incendios (Castillo et al., 2006) y el manejo de vehículos de exploración submarinos (McGann et al., 2008).

Dentro del campo de la PA, la planificación heurística es uno de los paradigmas más exitosos en los últimos años. Los problemas de planificación se resuelven como problemas de búsqueda, utilizando algoritmos guiados por una función de evaluación llamada heurística. El éxito de la planificación heurística reside en el desarrollo de heurísticas independientes del dominio, que permiten solucionar problemas en una gran variedad de dominios, que hasta entonces, no era posible con otros paradigmas de planificación. El inconveniente de las heurísticas independientes del dominio es el alto coste computacional que éstas tienen, lo que dificulta la resolución de problemas grandes. La función heurística se calcula al evaluar cada nodo del árbol de búsqueda, lo que implica de forma generalizada un problema de escalabilidad. Adicionalmente, en ciertos tipos de dominios, las heurísticas aportan poca información al proceso de búsqueda, lo que conlleva no encontrar una solución en un tiempo razonable.

Una de las alternativas para solucionar las limitaciones de la planificación heurística es apoyar el proceso de búsqueda con conocimiento de control dependi-

ente del dominio, de forma que el proceso no dependa únicamente de la eficiencia de la heurística. Otros paradigmas, como la planificación jerárquica (Nau et al., 2003), o la planificación basada en lógica temporal (Bacchus and Kabanza, 2000), han mostrado su buen rendimiento frente a otros planificadores, gracias al uso de conocimiento de control dependiente del dominio. No obstante, este conocimiento debe ser descrito por un experto, siendo en muchas ocasiones una tarea complicada. Una forma de resolver esta dificultad es adquirir automáticamente este conocimiento mediante un proceso de aprendizaje.

La adquisición de conocimiento de control por medio de técnicas de Aprendizaje Automático (AA) se ha utilizado en planificación en muchos contextos, especialmente en los años previos a la planificación heurística, cuando no se podían resolver problemas de cierta dificultad y se buscaba mejorar la eficiencia de los planificadores. La integración de las técnicas de AA y PA, además de ser una solución de ingeniería, tiene un trasfondo sobre la razón de ser de la IA. Un sistema capaz de razonar sobre distintos dominios no parece presentar mucha "inteligencia" si cada vez que recibe el mismo problema lo resuelve de forma idéntica y tarda el mismo tiempo. Por lo tanto, la PA ha integrado en sus sistemas diferentes técnicas de aprendizaje para además de poder resolver problemas con independencia del dominio, poder también mejorar su comportamiento en función de la experiencia.

En el contexto de la planificación heurística es necesario aplicar técnicas que, por un lado puedan recoger de forma abstracta una gran cantidad de episodios de planificación durante una etapa de aprendizaje o entrenamiento, y por otro lado puedan ser integradas de forma flexible en el proceso de búsqueda, para que sean útiles en los casos en los que la heurística no aporte información al proceso o que su coste computacional compense evitar evaluaciones de nodos de la búsqueda. El Razonamiento Basado en Casos es una alternativa atractiva porque podría almacenar el conocimiento particular de los dominios sin la necesidad de construir un modelo completo de comportamiento que sería difícil adquirir por la infinidad de planes posibles que existen en los dominios de planificación.

El Razonamiento Basado en Casos o CBR (por sus siglas en inglés: *Case-Based Reasoning*) es una técnica que intenta resolver los problemas utilizando el conocimiento de experiencias pasadas. El CBR se basa en que problemas parecidos suelen tener soluciones semejantes. Esta idea se refuerza al observar que los humanos nos enfrentamos a nuevos retos buscando por analogía, cómo hemos actuado en el pasado y qué hemos aprendido de las experiencias pasadas. El CBR fue uno de los paradigmas iniciales dentro del desarrollo de sistemas expertos. Se utiliza como base para muchos sistemas de recomendación, de diagnosis médica, diseño, configuración y planificación. Como disciplina, el CBR tiene sus orígenes en los trabajos de investigación que a principios de los años 80 realizaban las ciencias cognitivas. Se considera uno de los pilares principales de CBR el trabajo de Schank (Schank, 1982) sobre el modelo de memoria dinámica y sobre los MOPs *(memory organization packets)*, unidad que utilizaba la memoria para representar situaciones estereotipadas que fueron conocidas en el pasado. El CBR saca de esta idea su modelo, en el que retener sucesos o casos del pasado sirve para aprender y

enfrentarse a problemas que se presenten en el futuro.

El CBR ha sido aplicado en el pasado a diferentes técnicas de planificación. Aunque parece viable aplicarlo a la planificación heurística, plantea ciertas dudas, por la necesidad de una representación adecuada que permita abstraer correctamente las soluciones en una diversidad de dominios que anteriormente no existían, y por la necesidad de una técnica flexible que integre el conocimiento de control aprendido al proceso de búsqueda, a la vez que aprovecha la información aportada por las funciones heurísticas. Esto plantea el objetivo principal de esta tesis que consiste en desarrollar un sistema CBR que logre mejorar la eficiencia de un planificador heurístico. Las mejoras deberían obtenerse al utilizar el conocimiento aprendido para solventar las limitaciones que presenta la planificación heurística, principalmente el alto coste computacional de calcular la función heurística.

Esta tesis está estructurada de la siguiente forma. El segundo capítulo es una revisión al estado del arte, con una discusión final en la que se analizan las limitaciones de la planificación heurística y las posibles oportunidades de mejorar la eficiencia de sus técnicas. De este análisis se derivan los objetivos de la tesis. El cuarto capítulo recoge las diferentes metodologías de evaluación en planificación y presenta los dominios que se utilizarán para la fase experimental. A continuación dos capítulos, uno para describir el planificador heurístico desarrollado, y el otro para describir el sistema de razonamiento basado en casos, cada uno con sus respectivos apartados de evaluación experimental. Después un capítulo con análisis a diferentes aspectos del sistema CBR y finalmente las conclusiones y líneas futuras de investigación.

Capítulo 2

Estado del Arte

Este capítulo presenta una introducción a la Planificación Automática y un resumen de las principales investigaciones en el campo, de modo que sirvan de contexto para el trabajo presentado en esta tesis. Se incluyen las técnicas de Aprendizaje Automático que se han utilizado recientemente en planificación. Adicionalmente se presentan los fundamentos del Razonamiento Basado en Casos y se hace un repaso a los diferentes sistemas CBR que se han desarrollado para complementar distintos planificadores.

2.1. Introducción a la Planificación

La Planificación Automática busca resolver problemas descritos a alto nivel mediante la selección y organización de acciones que cambien el estado de un entorno. Desde una perspectiva general (Geffner, 2002) que englobe todos los enfoques, en la PA se pueden identificar tres elementos.

1. *El Modelo Conceptual* que describe el sistema.

2. *El Lenguaje de Representación* utilizado para describir los problemas a resolver.

3. *Los algoritmos* utilizados para resolver los problemas.

Dependiendo de las características del modelo conceptual se definen diferentes formas de planificación. Así, la **Planificación Determinista**, que describiremos en más detalle, tiene un pleno conocimiento del mundo. La **Planificación Basada en Costes** centra su atención en que las acciones tienen asociado un coste que influye en la organización final de las acciones de la solución. La **Planificación con Incertidumbre** que trabaja con modelos incompletos del mundo o la **Planificación Temporal** donde las acciones tienen distinta duración, lo que influye en la organización de las mismas.

2.2. Planificación Clásica

La planificación clásica, llamada también planificación determinista, es el modelo básico para la Planificación Automática. Aunque es el modelo menos realista, es en el que se han realizado la mayoría de las investigaciones en el campo. Esto se debe a la claridad del modelo y la facilidad de transformar problemas de otros modelos a éste. Este modelo trabaja sobre un sistema que describe el mundo en forma de estados y los cambios en el mundo se realizan a través de transiciones de estado que se consiguen al aplicar las acciones que permite el entorno. En este modelo (Ghallab et al., 2004) se asumen una serie de criterios:

- *Espacio de estados finito*: El sistema consta de un conjunto finito de estados.
- *Determinismo*: Una acción representa la transición de un estado a otro único estado.
- *Mundo estático*: No hay factores externos que cambien el mundo. Sólo la aplicación de una nueva acción puede generar un estado diferente.
- *Mundo completamente observable*: El sistema tiene un pleno conocimiento del estado actual.
- *Metas restrictivas*: El objetivo de los problemas es encontrar una secuencia de acciones que lleven a un estado donde se cumplan todas las metas.
- *Tiempo implícito*: Las acciones no tienen duración y se considera las transiciones de estado instantáneas.

Dentro de este modelo, el resolver un problema de planificación consiste en encontrar una secuencia de acciones que transformen un estado inicial del mundo en un estado en el que se cumplan las metas del problema.

2.2.1. Lenguaje de Representación de la Planificación Clásica

Los problemas de planificación se representan con un lenguaje que utilice lógica de predicados. STRIPS (Fikes and Nilsson, 1971) es uno de los lenguajes más utilizados para representar el modelo de la planificación determinista. Un problema en STRIPS es una tupla $P = (L, A, I, G)$ donde:

- L: representa el conjunto de literales en el espacio de estados.
- A: representa el conjunto de operadores.
- $I \subseteq A$: representa el subconjunto de literales que son ciertos en el estado inicial.
- $G \subseteq A$: representa el subconjunto de literales que deben ser ciertos en el estado meta.

La representación de una acción $a \in A$ consiste en 3 listas de literales en L. Estas listas son:

1. Precondiciones $Pre(a) \subseteq L$: Es el subconjunto de literales que deben ser ciertos para que una acción pueda ser aplicada.

2. Añadidos $Add(a) \subseteq L$: Es el subconjunto de literales que pasan a ser ciertos en el estado tras aplicar la acción

3. Borrados $Del(a) \subseteq L$: Es el subconjunto de literales que dejan de ser ciertos en el estado al aplicar la acción.

En la actualidad se utiliza el PDDL (Fox and Long, 2003) como lenguaje de representación para los problemas de planificación. El PDDL surgió como un intento de estandarizar los lenguajes de representación existentes. Tener un lenguaje común para los planificadores permitió que se realizara la Competición Internacional de Planificación (IPC), que desde el 1998 se efectúa cada dos años junto a la Conferencia Internacional de Planificación y Scheduling (ICAPS). El PDDL incluye la representación STRIPS, pero además incluye otras extensiones como ADL que permite una representación más rica a través de predicados negados, cuantificadores y efectos condicionales. Para diferenciar las diferentes extensiones, PDDL incluye en la descripción del dominio una definición de los requisitos *:requirements* necesarios que debe tener el planificador para poder manejar la descripción del dominio.

Los lenguajes de representación y en particular el PDDL definen un problema de planificación en dos partes.

1. Dominio: Describe el espacio de estados y las acciones que se pueden realizar. El espacio de estados se describe a través de tipos, constantes y un esquema o plantilla de predicados. Las acciones están descritas a través de un esquema de operadores que contienen sus respectivos esquemas de precondiciones, añadidos y borrados. La Figura 2.1 muestra un ejemplo de una acción en el dominio de los *Depots* [1].

2. Problema: Describe los objetos, el estado inicial y las metas del problema. La Figura 2.2 muestra un ejemplo de definición de problema para el dominio de los *Depots*.

Aunque en PDDL se les llama acciones a los operadores, por claridad, llamaremos operadores al esquema (definición con parámetros variables) de las acciones, y llamaremos acciones a las instancias particulares en donde los parámetros del operador son sustituidos por objetos reales del problema.

[1] El dominio de los *Depots* fue parte de la IPC3 en 2002: Unos camiones transportan un conjunto de contenedores entre almacenes y distribuidores, y los contenedores deben colocarse sobre palés o sobre otros contenedores en sus respectivos destinos.

```
(:action Lift
   :parameters (?x - hoist ?y - crate ?z - surface ?p - place)
   :precondition (and (at ?x ?p) (available ?x)
                      (at ?y ?p) (on ?y ?z) (clear ?y))
   :effect (and (not (at ?y ?p)) (lifting ?x ?y)
                (not (clear ?y)) (not (available ?x))
                (clear ?z) (not (on ?y ?z))))
```

Figura 2.1: Ejemplo de una acción en un dominio PDDL.

```
(define (problem depotprob0) (:domain Depot)
(:objects depot0 depot1 - Depot
          truck0 - Truck
          pallet0 pallet1 - Pallet
          crate0 - Crate
          hoist0 hoist1 - Hoist)
(:init (at pallet0 depot0)
       (at pallet1 depot1)
       (clear crate0) (clear pallet1)
       (at truck0 depot1)
       (at hoist0 depot0) (available hoist0)
       (at hoist1 depot1) (available hoist1)
       (at crate0 depot0)
       (on crate0 pallet0))
(:goal (and (on crate0 pallet1))))
```

Figura 2.2: Ejemplo de un problema en PDDL.

2.2.2. Los Algoritmos de Planificación

Los algoritmos utilizados para resolver los problemas de planificación suelen estar ligados a los planificadores o en su defecto a un paradigma de planificación en concreto. En general, estos algoritmos de búsqueda exploran un grafo y su objetivo es encontrar un camino en el grafo que vaya desde un nodo que representa a la situación inicial hasta un nodo que representa a una posible situación final. Para clasificar los algoritmos de planificación se deben tener en cuenta los siguientes aspectos.

- *El espacio de búsqueda*: Se refiere a qué representa el grafo de búsqueda, y puede ser búsqueda en el espacio de estados, espacio de planes o espacio de grafos de planificación.

- *La dirección de la búsqueda*: Puede ser búsqueda hacia adelante según las acciones aplicables, hacia atrás desde las metas o un enfoque mixto con una búsqueda bidireccional.

- *El algoritmo de búsqueda*: El algoritmo que decide la generación, ordenación y selección de los nodos en el grafo de búsqueda. Por ejemplo: Profundidad, Mejor Primero, Hill-Climbing, Enforced Hill-Climbing, A*, etc.

- *Las heurísticas*: Las técnicas utilizadas para guiar la búsqueda y evitar la exploración completa del grafo. Éstas pueden ser dependientes del dominio, utilizadas como conocimiento de control en la construcción de los planes, o independientes del dominio, expresadas como funciones que dado un estado estiman la distancia hasta la solución.

- *Las técnicas de poda*: Muy relacionadas con las heurísticas, pero en su caso no sirven de guía directamente, sino que eliminan partes del grafo de búsqueda para hacer el proceso más rápido.

Históricamente, los algoritmos de planificación han ido evolucionando para resolver problemas cada vez más complejos a la vez que iban surgiendo nuevos paradigmas dentro de la PA. STRIPS (Fikes and Nilsson, 1971) fue un planificador de los años 70 que hacía búsqueda no informada y que podía resolver problemas muy sencillos. Su algoritmo consistía en una búsqueda dentro del espacio de estados con un encadenamiento hacia atrás a partir de las metas. Utilizaba una pila de sub-metas tratando siempre de resolver la meta encima de la pila. El resultado era una secuencia de acciones de orden total. Presentaba el inconveniente de la no linealidad de las metas, resuelto años después por el planificador PRODIGY (Carbonell et al., 1992). PRODIGY permitía además la incorporación de heurísticas dependientes del dominio a través de reglas de control. Otra de sus aportaciones fue la ampliación del lenguaje de representación a modelos más expresivos, cercanos a las últimas versiones de PDDL. Fue utilizado ampliamente para investigaciones en planificación y aprendizaje como en PRODIGY/EBL (Minton, 1988), ANALOGY (Veloso and Carbonell, 1993) y en HAMLET (Borrajo and Veloso, 1997).

Por otro lado UCPOP (Penberthy and Weld, 1992) fue un planificador de los años 90 que realizaba la búsqueda en el espacio de planes. Esta característica también le permitía manejar la no linealidad de las metas. Utilizaba ADL como lenguaje de representación y su salida era un plan de orden parcial. Otro enfoque diferente era el utilizado por SATPLAN (Kautz and Selman, 1992), un planificador que transformaba los problemas de planificación en problemas de satisfacción lógica (SAT). Estos problemas se resolvían con un sistema de resolución SAT y la solución se transformaba en una secuencia de acciones.

A mediados de los años 90, la PA consigue grandes avances con la aparición de GRAPHPLAN (Blum and Furst, 1995). Este planificador construía un grafo de planificación instanciando todos los planes paralelos posibles. Después el algoritmo busca una secuencia de acciones hacia atrás dentro del grafo de planificación. Aunque esta técnica representaba un aumento considerable en el consumo de memoria para los planificadores, permitió resolver problemas de mayor complejidad.

Por otro lado SHOP2 (Nau et al., 2003) es un ejemplo de un planificador que trabaja sobre el paradigma de planificación jerárquica. Este tipo de planificación se basa en la descomposición de una tarea en subtareas más sencillas, hasta que se obtengan tareas primitivas que formen el plan. El modelado del dominio debe contener métodos que especifiquen cómo descomponer las tareas en subtareas.

2.2.3. Planificación Heurística

El surgimiento de los planificadores heurísticos fue el siguiente avance de importancia después de la aparición de GRAPHPLAN. Se entiende por planificación heurística a la resolución de problemas de planificación mediante un algoritmo de búsqueda guiado por una función heurística independiente del dominio. Una alternativa para construir estas heurísticas es resolver una versión simplificada del problema, llamada también problema relajado. La relajación propuesta por Drew McDermott (McDermott, 1996) consistió en ignorar la lista de borrados de las acciones del dominio, por lo que el problema relajado consistía en el mismo estado inicial y las mismas metas, pero la aplicación de las acciones sólo añadían nuevos literales al estado. La manera de resolver este problema relajado da lugar a diferentes heurísticas utilizadas por algunos planificadores. La solución óptima al problema relajado es una heurística admisible (h^+), lo que permitiría encontrar soluciones óptimas al problema normal con algoritmos como el A*. Sin embargo, el cálculo de esta heurística es NP-duro (Bylander, 1994), por lo que se utilizan aproximaciones para resolverlo en un tiempo razonable. Por ejemplo, HSP (Bonet and Geffner, 2001) calculaba su heurística h^{add} sumando los costes de todas las precondiciones de un conjunto de acciones necesarias para conseguir las metas. Esta heurística aunque estaba bien informada, no tomaba en cuenta la dependencia entre las metas y sobrestimaba en muchas ocasiones el coste total. h^{max}, la otra heurística en HSP, escoge el máximo coste de alcanzar una meta, calculado como la suma de las precondiciones más caras de las acciones que consiguen esa meta. Esta heurística es admisible, pero al referirse sólo a la meta "más cara", pierde mucha información y no es práctica para resolver problemas en muchos dominios.

Más tarde surge FF (Hoffmann and Nebel, 2001), uno de los planificadores más utilizados en investigación gracias a su éxito en dos competiciones de la IPC. Su aportación fue resolver el problema relajado (obteniendo un plan relajado) con un algoritmo tipo GRAPHPLAN en donde el número de acciones del plan relajado se toma como la distancia estimada al estado meta. Aunque esto resolvía parcialmente el problema de la independencia de las metas de h^{add}, la heurística propuesta por FF seguía siendo no admisible. FF utiliza el *Enforced Hill-Climbing* (EHC), un algoritmo que realiza iterativamente búsqueda local en amplitud hasta encontrar un nodo intermedio que mejore el valor heurístico. Incluye además una técnica de poda donde sólo se consideran las *helpful actions*, acciones que pertenecen al plan relajado o agregan alguna de las precondiciones necesarias para alcanzar alguna de las acciones del plan relajado. El planificador YAHSP (Vidal, 2004) utiliza las acciones de los planes relajados para construir estados *lookahead*. Estos estados se construyen al aplicar consecutivamente las acciones del plan relajado hasta que no se puedan aplicar más. Los estados *lookahead* permiten avanzar la búsqueda en un algoritmo mejor primero gracias al ahorro de evaluaciones a la función heurística.

Otra alternativa para la construcción de heurísticas independientes del dominio es la descomposición del problema de planificación en grafos causales, utilizados por el planificador FAST-DOWNWARD (Helmert, 2006). Su idea consiste en

cambiar la representación de lógica de predicados del PDDL a la representación SAS+ (Backstrom and Nebel, 1995), que define los estados con una serie de variables a los que se le asignan objetos del problema. Esta representación alternativa codifica implícitamente restricciones del problema que se pueden extraer en los grafos causales. La heurística es la suma de las transiciones en los grafos que son necesarias para conseguir la asignación final de las variables.

2.3. Aprendizaje Aplicado a la Planificación

Debido a que la PA siempre ha sido una tarea computacionalmente difícil, desde sus inicios se ha complementado por técnicas de Aprendizaje Automático (AA). La idea básica del AA es mejorar el rendimiento de un sistema mediante la adquisición de conocimiento por medio de la experiencia (Mitchell, 1997). Tradicionalmente los enfoques seguidos para el aprendizaje se pueden dividir en:

- **Aprendizaje Inductivo**: A partir de un conjunto de ejemplos se aprende una función que permite identificar futuras instancias. En este enfoque es importante la obtención de un conjunto de ejemplos representativo del dominio en cuestión.
- **Aprendizaje Analítico**: Aplica razonamientos deductivos sobre el modelo del dominio para obtener un conocimiento que generalice y explique uno o pocos ejemplos de entrenamiento.
- **Aprendizaje Analítico-Inductivo**: Es un enfoque mixto en el que el conocimiento obtenido por razonamiento deductivo es refinado mediante un conjunto de ejemplos.

Desde el punto de vista de la planificación, la adquisición de conocimiento plantea la pregunta de qué aprender. En este sentido podemos plantear que el objetivo del aprendizaje en planificación se centra en aprender el modelo del dominio para tener una mejor representación, o aprender conocimiento de control de los algoritmos para tener un mejor proceso de búsqueda.

2.3.1. Aprendizaje del Modelo de Dominio

Con el aprendizaje del modelo del dominio se intenta enriquecer el conocimiento de un dominio específico a través de un análisis previo a la planificación. Por ejemplo, TIM (Fox and Long, 1998) es una herramienta de análisis de dominio que soportaba al planificador STAN. Esta herramienta descubría automáticamente invariantes de estado, o lo que es equivalente, máquinas de estado finito que representan los estados por los que puede pasar una instancia en un problema de planificación. Con esta técnica se simplificaban los esquemas de instanciación de los operadores y permitía, a través del descubrimiento de simetrías, podar el árbol de búsqueda del planificador que estaba basado en GRAPHPLAN. Otra simplificación

para instanciar los operadores se encuentra en RIFO (Nebel et al., 1997) mediante el descubrimiento de predicados irrelevantes en los estados.

Otros enfoques intentan aprender la definición de los operadores del dominio o intentan refinarlos. Por ejemplo, dentro del modelo de la planificación clásica, Wang (Wang, 1994) aprendía operadores para PRODIGY utilizando trazas de ejecución de planes. Después, ARMS (Yang et al., 2007) aprendió los operadores PDDL utilizando el estado inicial, el plan observado y el estado final de un problema. Así mismo, (Shahaf and Amir, 2006) proponen un algoritmo para aprender la definición de los operadores en entornos parcialmente observables.

2.3.2. Aprendizaje de Conocimiento de Control

El aprendizaje de conocimiento de control consiste en adquirir conocimiento, por lo general dependiente del dominio, que complemente el proceso de búsqueda, con el objetivo de hacer el proceso más eficiente o de encontrar mejores soluciones. El proceso de aprendizaje puede realizarse durante la planificación o después de la obtención de un plan válido. Independientemente del enfoque de aprendizaje que se utilice, se debe identificar primero cuáles son los objetos u oportunidades para aprender y segundo cómo representar el conocimiento aprendido. En el conocimiento de control se consideran objetos de aprendizaje a:

- *Puntos de Decisión*: Dependiendo del paradigma de planificación y del algoritmo utilizado se puede aprender de diferentes puntos de decisión en el árbol de búsqueda. Así en la búsqueda del espacio de estados se puede decidir sobre qué meta trabajar, qué operador o instancia de operador elegir. En la búsqueda en el espacio de planes se puede elegir entre qué meta trabajar y qué amenazas solucionar. En la planificación jerárquica se puede aprender qué método elegir.

- *Puntos de Retroceso*: Según los nodos de fallo que se encuentren, el algoritmo empleado puede aprender los puntos de retroceso o *backtracking*.

- *Funciones de Evaluación*: En los algoritmos heurísticos se pueden aprender las funciones de evaluación o un refinamiento de las mismas para luego mejorar el ordenamiento de los nodos.

- *Episodios de Planificación*: Se intenta aprender total o parcialmente el proceso de planificación en conjunto. Dependiendo de la forma de representar el conocimiento esto puede ser el propio plan (solución completa), secuencia de operadores (macro-operadores), transiciones de estado y secuencias a partir del árbol de búsqueda, por lo general con anotaciones de los puntos anteriores.

El otro aspecto del aprendizaje de conocimiento de control es la representación de dicho conocimiento. Este va muy ligado a la forma de reutilizar el conocimiento durante el proceso de búsqueda. Los principales son:

Macro-operadores

Los macro-operadores fueron utilizados por primera vez para mejorar el rendimiento del planificador STRIPS (Fikes et al., 1972). Los macro-operadores son secuencias de operadores que se añaden a los operadores del dominio. De esta forma se reduce la profundidad del árbol de búsqueda y se avanza más rápido hacia las metas. Su principal inconveniente es que aumentan el factor de ramificación, por lo que parte del trabajo de investigación es determinar un balance entre el número de macro-operadores y su utilidad para el dominio. Esta técnica se ha utilizado con las últimas técnicas de planificación como es el caso de MACRO-FF (Botea et al., 2005) que aprende macro-operadores en FF. Estos mismos macro-operadores pueden utilizarse iterativamente para acelerar aún más el avance hacia las metas (Botea et al., 2007). Los macro-operadores pueden generarse automáticamente combinando los operadores del dominio, pero a fin de evitar la generación de macro-operadores inútiles, los enfoques modernos como MACRO-FF, los generan a través de ejemplos de planes resueltos. Una alternativa es aprenderlos con una estrategia en línea durante la propia planificación. MARVIN (Coles and Smith, 2007) utiliza macro-operadores en línea generando secuencias de operadores que descubre durante la planificación y que intenta aplicar dentro del mismo problema. Otra opción es generar los macro-operadores a partir de planes resueltos por diferentes planificadores fuentes y combinarlos con algoritmos genéticos (Newton et al., 2007).

Reglas de Control

Las reglas de control son sentencias del tipo *Si...entonces*, que se utilizan para guiar la exploración del árbol de búsqueda. Este conocimiento puede utilizarse mediante la poda del árbol de búsqueda o mediante la ordenación de los nodos para su exploración. El sistema PRODIGY/EBL (Minton, 1988) aprendía reglas de control mediante aprendizaje basado en la explicación (EBL). GRASSHOPER (Leckie and Zukerman, 1991) aprendía reglas de control con un enfoque inductivo utilizando técnicas de ILP (*Inductive Learning Programming*), para guiar al planificador PRODIGY. STATIC (Etzioni, 1993) generaba reglas de control con un enfoque analítico utilizando técnicas de EBL (*Explanation Based Learning*). HAMLET (Borrajo and Veloso, 1997) utilizaba un enfoque mixto. Primero aprendía reglas con EBL y luego mediante técnicas inductivas generalizaba y refinaba estas reglas.

Casos

En planificación, los casos representan episodios de planificación que se almacenan en una base de conocimiento para después ser reutilizados en nuevas situaciones. Dado que esta técnica es el tema central de este trabajo, ésta se explicará más detalladamente en la Sección 2.4.

Funciones Heurísticas

Ya hemos mencionado que los planificadores heurísticos realizan la búsqueda con un algoritmo guiado por una función heurística. La heurística de FF, siendo la más utilizada, es computacionalmente costosa y para varios dominios no suele estar muy informada. En este sentido, el aprender funciones heurísticas, o al menos corregir su valor debe considerarse también como aprendizaje para el control de la búsqueda.

Una reciente aproximación (Yoon et al., 2006) propone, con una técnica inductiva, aprender un suplemento al valor heurístico que permita corregir las desviaciones del coste real. Mediante la observación de problemas resueltos se intenta hacer una regresión lineal que represente esta diferencia, que luego será el suplemento al valor heurístico. Por otro lado, (Xu and Fern, 2007) aprenden funciones lineales que permiten localmente hacer un ranking de los estados para mantener los caminos prometedores dentro de la lista abierta de una búsqueda en haz.

Otra opción es aprender las funciones heurísticas utilizando bases de datos de patrones o PDBs (*Pattern DataBases*). La idea consiste en conseguir la abstracción de un problema que sirva de patrón y que pueda ser resuelto de forma óptima mediante búsqueda exhaustiva. Después, en una tabla se memoriza el valor heurístico para cada estado, que por la simplificación del problema suele ser una heurística admisible. (Edelkamp, 2001) es un primer trabajo utilizando las PDBs. Posteriores como (Holte et al., 2004) intentaban mejorar las estimaciones escogiendo patrones más precisos. (Helmert et al., 2007) desarrollaron una estrategia para generar las abstracciones más adecuadas para los tipos de representación utilizados en planificación.

Políticas

Una política es una función que explícitamente indica cuál acción elegir para cada posible estado del mundo. Tiene su inspiración en el aprendizaje por refuerzo, en donde se aprenden las recompensas que se pretenden obtener con el tiempo al aplicar cada acción en cada estado específico. Dzeroski (Dzeroski et al., 1998) desarrolló la técnica del aprendizaje por refuerzo relacional, en donde se puede aprender políticas utilizando un lenguaje de representación relacional como el utilizado comúnmente en la planificación de tareas. El problema que presentaban las políticas era encontrar una forma de generalizarlas, porque en planificación resulta inviable la representación explícita de los estados. Por eso era necesario construir políticas generales, que son un conjunto resumido de reglas que cubren todos los posibles estados del mundo y pueden decir qué acción aplicar. (Khardon, 1999) y (Martin and Geffner, 2004) consiguieron aprender estas políticas generales para algunos dominios de planificación. En general, resulta muy difícil aprender políticas eficientes en una gran variedad dominios con un método automático. Por eso, (Yoon et al., 2008) presentan un enfoque en el que las políticas están integradas dentro del algoritmo de búsqueda. Introduciendo como nodos de la búsqueda es-

tados en los que se ha aplicado la política en un número de pasos limitado, consiguen un algoritmo robusto que aprovecha la política cuando ésta ofrece buen conocimiento y si no, se apoya en la función heurística utilizada normalmente. Este trabajo obtuvo buenos resultados en la sección de aprendizaje de la IPC-6.

Otra posibilidad es aprender las políticas en forma de árboles de decisión relacionales como en (Cocora et al., 2006), en donde se determinaba qué acción ejecutar para el campo de navegación de robots. Dentro del campo de la planificación, (De la Rosa et al., 2008) consiguen aprender políticas en forma de árboles de decisión relacionales utilizando la herramienta TILDE (Blockeel and De Raedt, 1998).

Programas Específicos de Planificación

Un programa específico de planificación es una forma particular de política general que, en lugar de ser un conjunto de reglas, es un programa que permite abstraer los posibles estados del problema. Con este programa se puede decidir la acción a tomar con la simple ejecución del código. A través de bifurcaciones y ciclos se codifica sin marcha atrás qué acción se debe ejecutar. DISTILL (Winner and Veloso, 2003) genera automáticamente el código de los planificadores específicos en función de ejemplos de planes y logra una generalización parcial al mezclar los códigos de planificadores que son subconjuntos de otros o tienen solapamiento de código.

2.4. Razonamiento Basado en Casos

El Razonamiento Basado en Casos es una técnica de resolución de problemas que se apoya en la premisa de que problemas parecidos tendrán soluciones semejantes. Tomando este principio como base, su idea consiste en, dado un problema a resolver, recuperar de una memoria las experiencias pasadas más parecidas, para que las soluciones utilizadas anteriormente sirvan de guía para encontrar las nuevas soluciones. El CBR se centra en los casos, como unidad básica de razonamiento, y en el ciclo CBR, como el proceso iterativo de aprendizaje y reutilización de los casos. En esta sección explicaremos brevemente estos conceptos para luego presentar los principales sistemas basados en casos que se han desarrollado en el ámbito de la planificación.

2.4.1. Representación de Casos

Se entiende por caso a un episodio o experiencia pasada que está almacenado en la memoria y que se puede utilizar para razonar sobre nuevos problemas. De forma general un caso se puede ver como una tupla $Caso = \{$Problema, Solución, Efectos$\}$ donde

- **Problema**: Es la descripción de la situación pasada.

- **Solución**: Es la solución al problema o los pasos necesarios para obtener dicha solución.

- **Efectos**: Describe la situación después de aplicar la solución al problema. Sirve para identificar contextos en los que se aprende de experiencias de éxito y de fallo.

Este esquema genérico para representar un caso es válido tanto en situaciones sencillas como complejas. Una caso sencillo puede representar un ejemplo de clasificación donde el problema es un conjunto de variables atributo-valor, la solución es la clase a la que pertenece el ejemplo y la memoria sólo almacena casos positivos. Un caso complejo puede representar un estado inicial en un modelo, la solución una serie de pasos aplicados en la experiencia pasada y los efectos cuál fue el resultado de aplicar dichos pasos. De este modo Watson (Watson, 1997) propone clasificar los sistemas CBR por el tipo de tarea que desempeñan en:

1. *Clasificación*: La solución almacenada en el caso es la clase a la que pertenece la situación descrita en el problema. Suele ser útil para problemas con información incompleta. La clase del caso recuperado se utilizada como solución. En esta categoría entran los sistemas de diagnosis, predicción, control de procesos y evaluación.

2. *Síntesis*: La solución para el nuevo problema debe ser construida en función de los casos recuperados, por lo general después de un proceso de adaptación. Aquí entran los sistemas de diseño, planificación y configuración.

2.4.2. El Ciclo del Razonamiento Basado en Casos

El CBR tiene como una de sus justificaciones la analogía con el mundo real, donde un experto tiene más conocimiento a medida que tiene más experiencia. En este sentido en el CBR se entiende el aprendizaje como un ciclo continuo de recordar y memorizar nuevas situaciones para ir ganando experiencia.

Kolodner (Kolodner, 1993) propone una etapa inicial que involucra los procesos de recuperación y proposición de una nueva solución. Después, una etapa en la que se adapta o se justifica la solución propuesta. Finalmente, un proceso de evaluación y almacenamiento. Una visión parecida (Aamodt and Plaza, 1994) propone un ciclo conocido popularmente como las cuatro Rs del CBR y que es actualmente el más citado en la comunidad. Consta de cuatro procesos que son: recuperación, reutilización, revisión y retención. En líneas generales se entiende que las diferencias entre estos ciclos, o con otros propuestos, dependen del sistema CBR y de cómo dicho sistema utilice y mantenga la base de casos. Se puede decir que por regla general todo sistema basado en casos ejecuta su ciclo en tres grandes procesos mostrados en la Figura 2.3 y que se detallan a continuación.

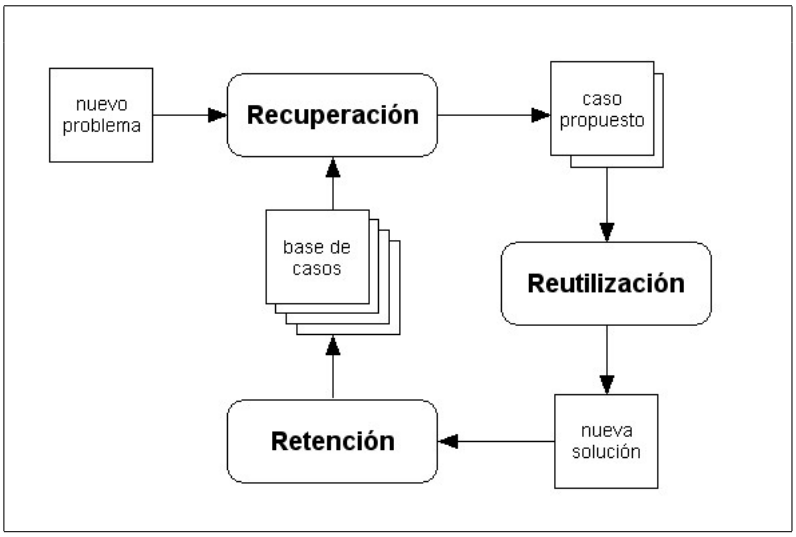


Figura 2.3: El Ciclo de Razonamiento Basado en Casos.

Recuperación

El proceso de recuperación se encarga de extraer de la base de casos el caso o los casos más parecidos a la situación actual. Centra su objetivo en encontrar una medida de similitud efectiva que permita una buena recuperación de casos. Dependiendo de la representación de los casos y del número de casos recuperados puede tener las siguientes fases (Kolodner, 1993):

- **Identificación de características**: Por lo general los casos no están representados como los problemas, por lo que hay que identificar en el problema nuevo las características que se van a utilizar para la recuperación. Un ejemplo práctico es identificar los elementos por los que están indexados los casos.

- **Comparación**: También llamada *matching*, que es donde se compara el nuevo problema con los anteriores. Esta comparación se hace equiparando las características que forman los índices de los casos. Es su ausencia la comparación se hace con todos los casos en la base de casos.

- **Ranking**: Se le asigna a cada caso un valor en función de la medida de similitud, para determinar cuál de todos los casos se parece más al problema actual y qué grado de semejanza tienen el resto de casos, por si se utilizan varias alternativas.

- **Selección**: Se escogen los n mejores casos del *ranking* según se necesiten en el proceso de reutilización.

Reutilización

La reutilización se encarga de resolver la nueva situación. En los sistemas CBR de clasificación se propone la solución del caso almacenado, pero en los sistemas de síntesis este proceso implica a su vez un proceso de adaptación. La adaptación en CBR sigue dos enfoques básicos (Kolodner and Leake, 1996).

- **Transformación**: Consiste en cambiar estructuralmente la solución del caso propuesto, conservando las partes útiles y modificando las partes que no se adapten a la nueva situación. Esta transformación puede realizarse por sustitución, reglas de transformación, re-instanciación o por ajuste de parámetros.
- **Derivación**: Consiste en generar una nueva solución utilizando las mismas inferencias o las mismas reglas que se utilizaron en el problema anterior. La idea se basa en transferir por analogía las líneas de razonamiento para ser utilizadas en el nuevo problema.

Estas dos formas de adaptación no son contrarias. De hecho, existen enfoques híbridos, que por ejemplo hacen una re-instanciación del caso y luego utilizan derivación. En ocasiones el proceso de reutilización puede contener una fase de revisión que intenta verificar si después de la adaptación el caso es útil, con el objetivo de utilizarlo o intentar nuevas modificaciones. También puede valer para indicar que es necesario recuperar un caso diferente porque el recuperado no ha sido útil para resolver el problema.

Retención

El almacenamiento se encarga del mantenimiento de la base de casos y de almacenar las nuevas situaciones una vez resueltas. Dependiendo del sistema CBR puede contener las siguientes fases:

- **Evaluación**: Determina si es necesario aprender la nueva situación. Esto puede ser porque la solución final era muy diferente de la del caso recuperado o porque se está en una fase inicial de entrenamiento donde se almacenan todas las experiencias.
- **Mezcla**: Realiza la unión de dos o más casos en uno mediante alguna técnica de generalización. Puede decidirse en la fase de evaluación al ver que varios casos son muy parecidos y no merece la pena hacer más grande la base de casos.
- **Indexación**: Coloca físicamente el nuevo caso en el lugar correspondiente en la base de casos, sea por el índice o por algún tipo de organización que tenga la memoria.
- **Olvido**: Realiza labores de mantenimiento y determina si algún caso ya no será válido en el futuro y tiene sentido eliminarlo de la base de casos.

2.4.3. Planificación Basada en Casos

Se entiende por Planificación Basada en Casos como la planificación que utiliza técnicas de CBR para la construcción de planes, en lugar de planificar desde cero. Los casos suelen ser los planes (o abstracciones de ellos) que resolvieron problemas pasados, o secuencias de razonamiento que permiten la reconstrucción de los planes. Se le atribuye a CHEF (Hammond, 1990) ser el primer planificador basado en casos. Utilizaba el dominio de las recetas de cocina, recibía como entradas las metas en forma de características que debían tener los platos y su salida eran las recetas, con secuencias de pasos para construirlas. CHEF recuperaba de la memoria de casos, el caso más parecido en función de las características dadas en las metas y luego adaptaba la receta a la nueva situación, primero instanciando el plan con los nuevos objetos, y luego reparando el plan en función de anotaciones que se incluían en el caso para anticipar los fallos.

Ya que el CBR es utilizado en muchos áreas dentro de la inteligencia artificial, en ocasiones la integración de CBR y planificación se le atribuye a sistemas que no son tradicionalmente de PA. Así, una revisión exhaustiva del campo puede incluir indistintamente a los sistemas que involucran planificación automática y a los que no (Spalazzi, 2001). Por ejemplo, un sistema de diagnosis basado en casos puede incluir en su representación un atributo que sea el plan de tratamiento para una enfermedad, y esto no representa en sí un plan entendido desde el punto de vista de la PA. Otros sistemas, aunque sí incluyan planes reales con secuencias de acciones, entran dentro del contexto de sistemas desarrollados para problemas específicos. Por ejemplo, CHEF del que ya hemos hablado y CHARADE (Perini and Ricci, 1996) que utiliza técnicas de CBR para producir planes jerárquicos dentro del campo de la extinción de incendios. Este sistema, aunque incluía un ciclo completo de CBR, no poseía un planificador generativo como tal, sino que se centraba en recuperar planes pasados y en adaptarlos en función de una nueva asignación de recursos para la situación de un incendio particular.

En este sentido sólo consideramos planificadores basados en casos a los sistemas de planificación que buscan secuencias de acciones aplicables, que cambian el mundo dentro del contexto de un modelo, y que a su vez estén complementados por técnicas de CBR. Una visión un poco más cercana se encuentra en la revisión del 2005 de los últimos avances en el campo (Cox et al., 2005). A continuación se muestra un resumen de los principales sistemas CBR aplicados a la planificación, clasificados según el paradigma de planificación.

2.4.4. CBR en Planificadores en Espacio de Estados

PRODIGY/ANALOGY (Veloso and Carbonell, 1993) fue el primer sistema que integró un ciclo completo de CBR aplicado a la planificación, y en particular a la planificación que hace búsqueda en el espacio de estados. Fue construido sobre la arquitectura del planificador PRODIGY (Carbonell et al., 1992). La integración del CBR está basada en la derivación, una técnica de reconstrucción que transfiere y

adapta las líneas de razonamiento utilizadas en los problemas pasados. Un caso consistía en la traza de planificación acompañada de una serie de justificaciones que podían ser:

1. Enlaces entre alternativas de decisión dentro del árbol de búsqueda, que capturaban las dependencias entre metas.

2. Registro de las alternativas de fallo que han sido exploradas.

3. Punteros a heurísticas externas.

El sistema extraía una huella (*foot-print*) de las precondiciones mínimas para alcanzar un conjunto de metas e indexaba los casos por este criterio. Cuando se presentaba un nuevo problema el proceso de recuperación buscaba la similitud con los casos almacenados comparando sus índices con el estado inicial y la huella generada del problema nuevo. Su utilidad fue probada en los dominios STRIPS utilizados para la época. Más adelante, este trabajo sirvió como base para futuras investigaciones. Por ejemplo en un enfoque de iniciativa mixta (Cox and Veloso, 1997), que combinaba la planificación basada casos con planificación humana integradas en una interfaz común.

En el contexto de la planificación en el espacio de estados también está PARIS (Bergmann and Wilke, 1996), un sistema de planificación basado en casos que introduce técnicas de abstracción dentro de los procesos de CBR. Los casos concretos en la memoria eran abstraídos a diferentes niveles, produciendo un conjunto de casos abstractos que también eran guardados en la memoria durante la fase de almacenamiento. Cuando un nuevo problema encajaba con un caso abstracto, éste era recuperado y los detalles ausentes eran completados en una fase de refinamiento. Este refinamiento se llevaba a cabo por el planificador, que hacía búsqueda hacia adelante en el espacio de estados. PARIS utilizaba un enfoque de generalización durante el aprendizaje y de especialización durante la resolución de problemas, lo que le permitía ser más flexible en el proceso de reutilización. Dado que en el proceso de recuperación podían estar como alternativa casos abstractos a diferentes niveles, siempre se escogía el de nivel más bajo para realizar menos esfuerzo de refinamiento. El proceso de almacenamiento organizaba la memoria de tal manera que los casos en un nivel de abstracción servían de índices para los casos de las mismas características pero con más nivel de detalle en niveles inferiores. De esta manera se formaba una memoria jerárquica adecuada para el enfoque de recuperación utilizado.

2.4.5. CBR en Planificadores en Espacio de Planes

Poco después de que surgiera CBR aplicado a la planificación en el espacio de estados, surge también su integración con planificadores en el espacio de planes. CAPLAN/CBC (Muñoz-Avila et al., 1994) fue un sistema CBR construido sobre el planificador CAPLAN (Weberskirch, 1995), un planificador que hace búsqueda

en el espacio de planes. Mostró su utilidad en dominios técnicos de manufactura de piezas. En este tipo de dominios un problema esta caracterizado por una situación inicial, unas metas y unas restricciones de orden llamadas dependencias. La recuperación de casos está guiada por las dependencias en lugar de las metas. Esta estrategia es más adecuada para estos dominios en los que se puede predeterminar un orden parcial entre metas. El proceso de reutilización consiste en una reconstrucción de un grafo de metas (la forma de razonar de CAPLAN). Este proceso permite interacción con el usuario que puede podar algunas opciones del caso recuperado. La reutilización permite además que el planificador evite las opciones de fallo reconocidas por la situación pasada.

Otro sistema dentro de este contexto fue DERSNLP (Ihrig and Kambhampati, 1996), que integraba CBR junto con el planificador de orden parcial SNLP (McAllester and Rosenblitt, 1991). El sistema utilizaba derivación en el proceso de reutilización de los casos. Otro detalle novedoso es que incluía aprendizaje con técnicas de EBL para anotar características de los nuevos problemas cuando el caso utilizado no podía ser reproducido con éxito en la fase de reutilización. Los casos estaban orientados a resolver una meta en particular, y sólo se referían a múltiples metas cuando no era posible mezclar sub-planes individuales para metas dependientes. SPA (Hanks and Weld, 1995) por *Systematic Plan Adaptor*, es un algoritmo de planificación basado en casos que sigue la filosofía del mínimo compromiso de los planificadores de orden parcial que hacen búsqueda en el espacio de planes. En el proceso de adaptación se buscaban las acciones inconsistentes para ser eliminadas. Como una extensión a SPA surgió MPA (Ram and Francis, 1996) (*Multiple Plan Adaptor*) el cual podía mezclar varios sub-planes recuperados de la memoria durante la construcción del plan de orden parcial final.

2.4.6. CBR en Planificación Jerárquica

La utilización de CBR dentro de la planificación jerárquica se remonta a la aparición de PRIAR (Kambhampati and Hendler, 1992). Aunque no debe considerarse un sistema CBR como tal, por no memorizar el conocimiento generado, sí fue muy revelador en cuanto a la etapa de reutilización del ciclo CBR. Este sistema utilizaba una adaptación transformacional para adaptar un plan inicial en caso de que se necesitara replanificación. Este plan inicial era generado por un planificador jerárquico no lineal. Después, el sistema identificaba con un proceso de validación qué partes del plan podían utilizarse, y cuáles necesitaban ser reparadas. Después, se buscaban los mínimos cambios posibles para que el plan fuera válido para la nueva situación.

El uso de CBR dentro de la planificación HTN se ha centrado en general en enriquecer el conocimiento del modelo del dominio. La idea básica consiste en utilizar el caso más parecido cuando no haya un método que indique la descomposición de una tarea. En este enfoque se encuentra SIN (Muñoz-Avila et al., 2001), que integraba el planificador SHOP (Nau et al., 2003) con una técnica conversacional de recuperación de casos. Un caso era un método instanciado junto con

un conjunto de pares {pregunta, respuesta} que ayudaban a la recuperación. Otro sistema de filosofía similar fue PROCHIP (Macedo and Cardoso, 2004). Este planificador también utilizaba los casos de la librería cuando no existían métodos para la descomposición de tareas. Su principal diferencia es que mantenía un árbol con tareas de probabilidad condicional para expresar las diferentes posibilidades de descomposición. Más recientemente fue desarrollado REPAIRSHOP (Warfield et al., 2007), que es una variante del planificador SHOP (Nau et al., 2003). Se diferencia de anteriores enfoques en que utiliza CBR para integrar la capacidad de replanificación dentro del planificador. REPAIRSHOP anota un registro de los caminos de fallo durante la planificación, para utilizar dicha información cuando está reparando el plan.

2.4.7. CBR en Otros Paradigmas de Planificación

Existen otros sistemas que aunque no son de dominio específico, no entran dentro de los paradigmas conocidos de la planificación. Entre ellos podemos citar a CAPER (Kettler et al., 1994) y a MRL (Koehler, 1996). CAPER fue un sistema CBR aplicado a la planificación que fue construido sobre la arquitectura de los computadores paralelos *Connection Machine* (Hillis, 1985) desarrollados en el MIT a mediados de los años 80. Tratando de aprovechar los beneficios de la computación paralela, CAPER centraba su enfoque en un proceso de recuperación de casos rápido sobre una base de casos de gran tamaño. La base de casos estaba estructurada como una red semántica y los casos eran recuperados en función de características de los objetos que aparecían en las metas y el estado inicial.

MRL (Koehler, 1996) fue un sistema de planificación basada en casos que utiliza un enfoque de planificación deductiva (Manna and Waldinger, 1987). La planificación deductiva es una alternativa a la planificación generativa, en la que se construye el plan por medio de probar la existencia de un estado que cumple las especificaciones del problema. Las secuencias de inferencias sobre una lógica formal daban lugar a un plan. Estas secuencias pueden tener a su vez estructuras de control como en el código de los planificadores específicos. MRL recuperaba planes de su librería y los planteaba como planes hipotéticos que luego modificaba y ajustaba para que cumplieran las nuevas especificaciones del problema que se quería resolver.

2.5. Discusión

Existe un consenso casi general de que la aparición de GRAPHPLAN es un punto de inflexión en el campo de la planificación automática, a partir del que se han sucedido una serie de avances significativos. Como visión personal y a manera de resumen, en el contexto de la planificación clásica, podemos citar los importantes en:

- El PDDL y sus diferentes evoluciones, como estándares de lenguajes de rep-

resentación para la planificación clásica y para el resto de modelos de planificación.

- El desarrollo de las heurísticas independientes del dominio basadas en la relajación de la lista de borrados de las acciones.
- La transformación de los problemas de planificación a problemas SAT.
- El desarrollo de los grafos causales y otras abstracciones basadas en la representación SAS+.

Estos avances se pueden entender dentro del contexto de la IPC, que sirve como referente para muchas de las investigaciones que se realizan actualmente en el campo. Por ejemplo, el PDDL puede considerarse como el primer gran logro de la IPC. Futuras evoluciones del lenguaje (ultima versión PDDL 3.1) han permitido que se puedan representar problemas más cercanos a la realidad. Otra aportación significativa dentro del contexto de la IPC es la creación de una colección de alrededor de 30 dominios, que varían en dificultad, estructura y objetivos. Especialmente, los de las últimas competiciones son ya modelos simplificados de problemas reales y empiezan a dejarse de parecer a los clásicos dominios de juguete.[2]

En el ámbito de las heurísticas, podemos indicar que la heurística de FF (Hoffmann and Nebel, 2001) ha sido la más influyente en los últimos años. Muchos de los planificadores que compitieron en la IPC en posteriores ediciones a la del año 2000 estaban basados en FF o al menos utilizaban su heurística. También se han desarrollado versiones de FF para otros modelos de planificación como: METRIC-FF (Hoffmann, 2003) para planificación basada en costes, CONFORMANT-FF (Hoffmann and Brafman, 2006) para planificación conforme o FF-REPLAN (Yoon et al., 2007) para planificación probabilística.

Consideramos que así como se ha trabajado mucho sobre las heurísticas independientes del dominio, bastante menos se ha trabajado sobre los algoritmos para planificación. Muchos algoritmos del mundo de la búsqueda heurística ni siquiera se han probado en algún planificador. Entre los algoritmos destacan el EHC, por la influencia que ha tenido FF, y el WBFS (*Weighted Best-First Search*) que se ha utilizado como alternativa a la búsqueda local de EHC. Por otro lado encontramos a LPG (Gerevini et al., 2004) que realiza búsqueda estocástica en el espacio de grafos de planificación, y a IDENTIDEM (Coles et al., 2006) que hace búsqueda estocástica en la regiones del espacio de estados donde EHC se queda estancado.

El aprendizaje automático tampoco tiene mucha presencia dentro del contexto de la IPC. Después de cinco ediciones se decidió organizar una sección de aprendizaje para la sexta edición en 2008. Anteriormente, sólo las macro-acciones de MACRO-FF (Botea et al., 2005) y de MARVIN (Coles and Smith, 2007) habían participado como técnicas de aprendizaje aplicadas a planificadores heurísticos. El

[2] Algunos dominios son llamados de juguete por la simplicidad del modelo que representan y no por su dificultad. La complejidad para producir planes es ya particular para cada dominio, y algunos como el mundo de los bloques (blocksworld) sigue presentando retos para la obtención de planes óptimos.

resto de técnicas entraron en la IPC ya en la sección de aprendizaje. El interés de aplicar técnicas de aprendizaje a planificadores heurísticos ha surgido porque estos planificadores, a pesar de tener un buen rendimiento en una considerable variedad de dominios, presentan limitaciones que comprometen su eficiencia. En particular, los planificadores que utilizan heurísticas basadas en la relajación de los borrados de las acciones presentan las siguientes limitaciones:

- Las heurísticas utilizadas tienen un alto coste computacional, dando lugar a que la mayor parte del tiempo de planificación se invierta en las evaluaciones a la función heurística, a requerimiento de los algoritmos de búsqueda, que para cada estado deben determinar la estimación del coste para alcanzar las metas.

- Las heurísticas realizan malas estimaciones en una serie de dominios, provocando que los algoritmos entren en regiones del espacio de estados que, o tienen el mismo valor heurístico o son mínimos locales. Las malas estimaciones se ven agravadas por el coste computacional, lo que provoca que en estos dominios los planificadores puedan resolver sólo problemas de poca dificultad.

- Las heurísticas que son admisibles (como $h^{\max}$, utilizada por HSP (Bonet and Geffner, 2001)) son muy poco informadas y sólo permiten resolver problemas pequeños dentro de un tiempo razonable.

La tercera limitación la estudian de forma particular los investigadores que trabajan con planificación óptima. Para enfrentarse a las dos primeras limitaciones se pueden plantear diferentes enfoques:

- Desarrollo de heurísticas más precisas, ya sea combinando varias de las existentes o aprendiendo el error de la estimación que éstas cometen.

- Introducir conocimiento de control a los algoritmos de búsqueda para que puedan reducir el número de evaluaciones de la función heurística. Se asume que una reducción en el tiempo invertido en evaluaciones se traduce en una mejora en la eficiencia general del algoritmo.

Ambos enfoques dan una oportunidad a las técnicas de aprendizaje. Tanto el error en las estimaciones como el conocimiento de control son dependientes del dominio y el aprendizaje permite adquirir de forma automática este conocimiento que de otro modo tendría que ser suplido por un experto.

En la revisión al estado del arte se vio que las macro-acciones (Botea et al., 2005; Coles and Smith, 2007; Newton et al., 2007) y las políticas (Yoon et al., 2008) se han utilizado en planificadores heurísticos. Ambas técnicas se pueden ver como estrategias de reducción de evaluaciones en el sentido que permiten conseguir estados a una mayor profundidad en el árbol de búsqueda que estén en el

camino de la solución. Sin embargo, técnicas probadas en el pasado, como el razonamiento basado en casos o las reglas de control no se han re-implementado sobre planificadores heurísticos. Los últimos avances en la PA dejaron obsoletos los planificadores que se apoyaban en sistemas de aprendizaje. Estos sistemas de aprendizaje eran fuertemente dependientes del planificador que acompañaban, por lo que es necesario adecuarlos y re-implementarlos para que sus ideas puedan ser utilizadas nuevamente. Esta es una de las razones por las que el AA ha estado fuera de la IPC hasta el 2008. Ahora que la planificación heurística es un paradigma aceptado y maduro, resurge el interés por aplicar técnicas que en su mayoría ya fueron probadas en el pasado. Los que son más críticos con la idea de retomar el aprendizaje, plantean que estas técnicas fueron probadas en pocos dominios y que su aplicación es particular al algoritmo que utiliza el planificador.

Desde nuestro punto de vista, el razonamiento basado en casos podría ser una solución alternativa al enfoque de introducir conocimiento de control a los algoritmos de búsqueda, de modo que se pueda reducir el número de evaluaciones de la función heurística. Este supuesto plantea el objetivo principal de este trabajo. Demostrar su validez implicaría que la eficiencia de la técnica se refleja en una variedad de dominios y que además, el conocimiento aprendido es aplicable sobre diferentes algoritmos de los utilizados en planificación.

Capítulo 3

Objetivos de la Tesis Doctoral

El objetivo principal de la tesis doctoral es desarrollar técnicas de razonamiento basado en casos para ser aplicadas a las técnicas de planificación heurística, de manera que se pueda mejorar el tiempo en el proceso de planificación y la calidad de los planes obtenidos.

Como idea general se plantea la creación de un sistema de aprendizaje que utilice el paradigma del razonamiento basado en casos para adquirir conocimiento dependiente del dominio a partir de problemas resueltos. Este conocimiento deberá integrarse dentro de la búsqueda de un planificador heurístico de modo que sirva como heurística complementaria que permita reducir el número de estados evaluados o para que aporte información en las regiones del espacio de estados donde la heurística esté desinformada. Este sistema de aprendizaje estará enmarcado dentro del modelo de planificación clásica. Los objetivos específicos de la tesis se pueden detallar como:

1. Desarrollar un planificador heurístico que sirva como base para el desarrollo del sistema de aprendizaje basado en casos. El planificador deberá adoptar las técnicas más recientes relacionadas con la búsqueda heurística, tanto en algoritmos como en funciones heurísticas.

2. Diseñar un modelo de representación para los episodios de planificación que permita almacenar en una base de casos el conocimiento necesario para complementar el proceso de búsqueda en un algoritmo heurístico.

3. Desarrollar un sistema de aprendizaje que integre el ciclo completo de CBR para ayudar a la resolución de problemas en el planificador desarrollado. El sistema implica la implementación de una base de casos que utilice la representación escogida y los programas que lleven a cabo las etapas de almacenamiento, recuperación y reutilización de los casos.

4. Evaluar el efecto sobre los algoritmos de planificación que produce la utilización del conocimiento dependiente del dominio proporcionado por la

base de casos. Explorar además, el efecto de combinar el conocimiento proporcionado por la base de casos junto con la información suministrada por las heurísticas independientes del dominio. Las evaluaciones deberán hacerse sobre un conjunto de dominios de diferente tipología y dificultad.

5. Analizar el conocimiento aprendido en las bases de casos para establecer relaciones entre la información almacenada y las posibles mejoras obtenidas en el proceso de planificación.

Capítulo 4

Metodologías de Evaluación

Este capítulo recoge un resumen de las principales metodologías utilizadas para la evaluación de sistemas de planificación, tanto los tradicionales como los que utilizan aprendizaje. La idea consiste en justificar la metodología utilizada para el estudio de los algoritmos planteados en esta tesis. Se describirán los dominios de planificación seleccionados para los experimentos junto con las características comunes de los experimentos diseñados para la evaluación.

4.1. Evaluación de Planificadores

Tradicionalmente, para comparar sistemas de planificación se toma la técnica o algoritmo que se está investigando y se compara con uno o varios planificadores que sean recientes en el estado del arte. Si el trabajo planteado es una modificación al planificador, heurística o algoritmo, se toma como base la técnica original y se compara con la técnica planteada.

En estas evaluaciones se toman como variables independientes:

1. El dominio de planificación.
2. El conjunto de problemas del dominio. En la mayoría de los casos con dificultad incremental.
3. El límite de tiempo de CPU para la resolución del problema.

Después, se analizan los resultados tomando en cuenta las variables dependientes que son:

1. El número de problemas resueltos.
2. El tiempo de procesador empleado para resolver los problemas.
3. La calidad de la solución.
4. El número de nodos generados y evaluados.

El número de problemas resueltos dentro de un límite de tiempo muestra una evidencia clara de que una técnica resulta mejor que otra. Para el resto de variables dependientes, éstas se suelen presentar en gráficas que muestran el tiempo de ejecución o la calidad de la solución por cada problema resuelto. Resulta difícil sacar conclusiones de este tipo de gráficas, debido a que en muchas ocasiones una técnica no domina a la otra en todos los problemas. De hecho, resulta aún más complicado cuando hay una relación entre la diferencia de dos variables dependientes, por ejemplo que en un problema se emplee más tiempo, pero se consiga mejor calidad en la solución.

Una alternativa al mostrar los resultados es presentar los valores acumulados o los valores medios de las variables dependientes, en especial el tiempo total de ejecución de los problemas. El problema de este análisis es que, para hacerlo justo, se incluyen en los acumulados sólo los valores de los problemas que hayan resuelto todas las técnicas. El inconveniente de este tipo análisis es que mucha información queda fuera de los resultados y en ocasiones resulta interesante ver precisamente cómo se ha comportado una técnica en los problemas que han sido resueltos sólo por esa técnica. La información también puede resultar engañosa por ejemplo en los casos en que dos técnicas resuelvan la misma cantidad de problemas pero el número de problemas resueltos por ambos es diferente, porque problemas que resuelve una técnica no se resuelven con la otra y viceversa.

Otra posibilidad para la evaluación de planificadores es establecer una métrica de evaluación que permita decidir por un valor global cuál técnica es mejor. Este tipo de evaluaciones se ha dado dentro de las IPCs para simplificar la elección de un ganador. En la edición del 2008, (IPC-6) tanto la sección determinista como la sección de aprendizaje utilizaron el siguiente esquema de evaluación

- Para cada problema se determina el valor V^* como el mejor valor (calidad o tiempo) que se haya visto en cualquier planificador.
- Para cada planificador, cada problema puntúa un ratio de V^*/V, siendo V el valor obtenido por ese planificador (calidad o tiempo).
- Puntúa 0 si no ha resuelto el problema.

De esta forma, cada planificador recibe por cada problema una puntuación entre 0 y 1, dependiendo de lo bien que lo haya hecho respecto a las demás técnicas. Ya sea por dominio o en el conjunto de la competición, la suma de todos los puntos permite hacer una comparación razonablemente justa entre diferentes técnicas. El inconveniente de esta métrica de evaluación es que el valor final es muy dependiente del conjunto de problemas que se haya escogido para evaluar. Sobre todo en la valoración del tiempo, un planificador puede perder muchos puntos por el ratio de evaluación, aunque en percepción para los humanos se esté comportando de forma cercana a otro planificador. En este aspecto, la implementación de los algoritmos juega un papel importante, porque las comparaciones entre planificadores no toman en cuenta el lenguaje de programación o las optimizaciones particulares

que se hacen sobre los algoritmos, que pueden influir en el ratio que se obtiene como puntuación de tiempo. El ratio en la evaluación de la calidad está más compensado, por lo que la puntuación total puede considerarse como un parámetro válido y justo de comparación.

Una alternativa adicional es presentar los resultados como distribuciones acumuladas de las variables dependientes. Esta idea proviene del campo de la búsqueda local estocástica en la que se realizan varias ejecuciones de los problemas y se determina la probabilidad de resolver un problema en un tiempo dado. En la planificación heurística, si los algoritmos utilizados son deterministas sólo se necesita una ejecución, pero la idea original puede mantenerse, representando el porcentaje de problemas resueltos como una función de distribución acumulada de la variable dependiente. Por ejemplo, si se quiere analizar el tiempo de ejecución del algoritmo $\mathcal{A}$, se puede saber qué porcentaje de problemas resueltos se ha alcanzado en un tiempo t, sabiendo el valor de la función de distribución acumulada para el tiempo de ejecución $P(T_{\mathcal{A}} \leq t)$.

Esta forma de comparación es válida para las otras variables dependientes tales como la longitud de la solución o el número de nodos evaluados. Este enfoque para presentar los resultados tiene la ventaja de mostrar si un algoritmo domina a otro de forma global mediante los valores acumulados, pero sin perder la información de los problemas que hayan sido resuelto sólo por alguno de los algoritmos.

4.2. Evaluación de Sistemas de Aprendizaje

Los planificadores que utilizan conocimiento aprendido para mejorar su rendimiento son generalmente comparados con un sistema base que suele ser el mismo planificador sin conocimiento. Más que demostrar que una técnica o algoritmo es mejor que otro, en este caso, se intenta mostrar que el conocimiento dependiente del dominio, aprendido de forma automática, sirve para mejorar el sistema de base. Se utilizan las mismas variables independientes, a las que se le añade un conjunto de problemas diferente llamado problemas de entrenamiento, que son utilizados para generar el conocimiento mediante alguna técnica de aprendizaje automático. Las variables dependientes son las mismas que al evaluar dos sistemas de planificación. En trabajos más recientes en los que se intenta aprender para múltiples planificadores, sí se incluyen varios planificadores en la evaluación. Por ejemplo, (Newton et al., 2007) incluyeron todos los planificadores que utilizaron para obtener los macro-operadores. La utilidad del conocimiento aprendido puede validarse con técnicas tradicionales del aprendizaje automático como la validación cruzada, pero en el campo de la PA, no se le suele dar mucha importancia a este tipo de evaluación, posiblemente por el tiempo que implica realizar los experimentos.

Debido a la dificultad de reproducir los algoritmos de aprendizaje de un sistema en otro, no se suele comprobar que un sistema de aprendizaje para planificación sea mejor que otro. A esto se le añade que en muchos casos la posibilidad de implementar otras técnicas es nula, ya que muchas de estas técnicas de aprendizaje

son específicas del planificador base y son difícilmente transferibles a otros planificadores. El primer intento de comparar distintos sistemas de aprendizaje para planificación se realizó en la sección de aprendizaje la IPC-6, en donde participaron diferentes enfoques de aprendizaje de conocimiento de control. El sistema de evaluación de esta competición fue igual a la sección determinista, en el que la puntuación por problema era V^*/V, según lo explicado en la sección anterior. La puntuación global permite decidir el mejor sistema de planificación, pero no el sistema de aprendizaje más efectivo, por lo que queda abierta en la comunidad la pregunta de cómo evaluar estos sistemas para determinar el sistema que más o que mejor aprende. Los resultados de la competición mostraron que es mucho más relevante el planificador de base, que el propio conocimiento aprendido, que en algunos casos hasta podía degradar el rendimiento de algunos planificadores en ciertos dominios. No obstante, esto no indica que el aprendizaje no pueda aportar mejoras importantes a la escalabilidad de los planificadores, pero sí pone de manifiesto que la elección de una buena técnica o planificador de base es un aspecto importante a considerar a la hora de desarrollar un planificador que utilice aprendizaje.

4.3. Metodología de Evaluación de la Tesis

En esta sección se describe en concreto la forma de evaluación que se va a utilizar en esta tesis y se justificarán las diferentes decisiones para el diseño de los experimentos. Como el objetivo del desarrollo de esta tesis es mostrar las posibles mejoras en rendimiento y escalabilidad de un planificador heurístico, se ha decidido para el diseño de experimentos utilizar las mismas variables independientes para todos los algoritmos heurísticos en los que se integre alguna técnica de razonamiento basado en casos. Éste permite a su vez, poder analizar los algoritmos entre sí, no solamente el algoritmo base y su mejora. A continuación se detallan la elección de las diferentes variables independientes.

4.3.1. Dominios de Evaluación

En la planificación automática, la selección de dominios para los experimentos es un aspecto crucial, porque siempre se intenta demostrar que la técnica es válida en cualquier dominio o en al menos una clase de ellos. La variedad se intenta conseguir eligiendo diferentes dominios según su complejidad computacional, su pertenencia a una clase en alguna clasificación o topología definida.

La selección para los experimentos de esta tesis se basa primero en que los dominios pertenezcan a diferentes IPCs, que resulta también una característica de interés. Y segundo por la topología definida por (Hoffmann, 2005) en la que se intenta caracterizar los dominios según el comportamiento de la función heurística del planificador FF (Hoffmann and Nebel, 2001). También se incluirán dos dominios de la sección de aprendizaje de la competición del 2008, que resulta de interés

dado que futuros trabajos relacionados con aprendizaje harán referencia a sistemas y resultados obtenidos en esa competición.

En la topología definida por Hoffmann, los dominios se clasifican por dos características. La primera, por la descripción de las acciones que puedan llevar a un camino sin salida o *dead-end*[1]. Según esta característica los dominios pueden ser:

- **No dirigidos**: Todas las acciones del dominio son reversibles, por lo que no hay estados de camino sin salida.
- **Inofensivos**: Las acciones o son reversibles, o añaden predicados que no se pueden borrar y borran hechos que no se necesitan en ninguna precondición. Tampoco hay estados de camino sin salida.
- **Reconocibles**: Las acciones pueden generar un camino sin salida, pero son reconocibles por la función heurística.
- **No reconocibles**: Las acciones pueden generar un camino sin salida, y la función heurística no puede reconocerlo al evaluar un estado.

La segunda característica para clasificar los dominios según la topología de Hoffmann es la existencia o no de *plateaus* acotados con independencia del tamaño de los problemas. Un *plateau* es una región conectada del espacio de búsqueda en la que todos los estados tienen un valor heurístico igual o peor que el valor heurístico del estado de entrada. Se considera que un nodo que mejora el valor heurístico de la región es un nodo de escape del *plateau*. Para determinar la existencia de *plateaus* acotados se define una medida máxima de escape como la profundidad máxima que puede tener un *plateau* en un dominio. Si la medida máxima de escape en un dominio es una constante, sin importar lo grande que sea el problema de planificación, se dice que ese dominio tiene *plateaus* acotados.

Según esta topología, los dominios van incrementando su dificultad según sean no dirigidos, inofensivos, reconocibles o no reconocibles. Tomando en cuenta la segunda característica, los dominios con *plateaus* acotados son más fáciles que los que no los tienen. Así, los dominios más difíciles son los no reconocibles, que de hecho implica tener *plateaus* no acotados.

Tomando en cuenta una diversidad en términos históricos de las IPCs (Mcdermott, 2000; Bacchus, 2001; Long and Fox, 2003; Gerevini et al., 2009) y una diversidad en pertenencia a la topología de Hoffmann, se han escogido ocho dominios de evaluación que pueden agruparse según el tipo de tarea del dominio de la siguiente forma:

- **Construcción**: Involucra un conjunto de objetos que deben estar en cierta configuración para formar un todo. Los dominios se diferencian por los

[1] Un estado de camino sin salida es un estado en el que no es posible encontrar una solución a partir de ninguno de sus sucesores.

tipos de objetos y por las restricciones a la hora de ir construyendo la configuración de los objetos. En este tipo están el *Blocksworld*, el *Matching Blocksworld* y el *Parking*.

- **Transporte**: Formado por un conjunto de sitios, de objetos y de medios de transporte. Las tareas en el dominio consisten en llevar los objetos de un sitio a otro. Los dominios se diferencian por las restricciones de conectividad de las localidades, las capacidades de los medios de transporte y por el coste (o combustible) asociado a mover los medios de transporte. En este tipo están el *Logistics*, el *Mystery'* y el *Portcrane*.

- **Ejecución de Tareas**: Involucra un conjunto de agentes y una serie de tareas que realizan en una serie de pasos en los que se pueden compartir recursos. En esta categoría los dominios puede ser muy diversos, pero por regla general un dominio consiste en configurar los agentes en ciertos estados para que puedan cumplir las tareas. En esta clasificación están el *Satellite* y el *Rovers*.

La Tabla 4.1 muestra un resumen de las principales características de los dominios elegidos. Los valores que se desconocen en la clase de *plateaus* se debe a que esos dominios no fueron analizados por el estudio de topología de dominios de Hoffmann y deducir sus características necesitaría un estudio adicional. Los valores de la clase de camino sin salida sí son deducibles analizando las acciones del dominio.

Dominio	Número operadores	Número predicados	Número Tipos	IPCs	Clase camino sin salida	Clase plateaus
Blocksworld	4	5	1	2000	no dirigido	no acotado
Matching-bw	10	10	2	2008	no reconocible	no acotado
Parking	4	5	2	2008	no dirigido	-
Logistics	6	3	6	1998, 2000	no dirigido	acotado
Mystery'	4	8	5	1998	no reconocible	no acotado
Portcrane	12	13	5	-	inofensivo	-
Satellite	5	8	4	2002, 2004	inofensivo	acotado
Rovers	9	25	7	2002, 2006	reconocible	no acotado

Tabla 4.1: Características de los dominios de evaluación.

Dominio Blocksworld

El dominio *Blocksworld* o mundo de bloques es uno de los primeros dominios creados para la investigación en PA. Consiste en una mesa abstracta de tamaño infinito sobre la que hay colocados un número arbitrario de bloques, que pueden estar sobre la mesa o sobre otros bloques. La tarea involucra levantar o dejar los bloques sobre la mesa o sobre otros bloques hasta conseguir la configuración deseada. Aunque a simple vista resulta un dominio de juguete, al día de hoy sigue siendo uno de los dominios más complicados por su fuerte dependencia entre las metas.

Dominio Matching Blocksworld

El dominio *Matching Blocksworld* (*matching-bw* en forma abreviada) fue uno de los dominios utilizados en la sección de aprendizaje de la IPC-6 en 2008. El dominio es una variación del *Blocksworld* en la que los bloques tienen una polaridad asociada y además hay 2 brazos, uno con cada polaridad. La tarea consiste en poner los bloques en una configuración específica, pero con la salvedad de que los bloques se dañan al cogerlos con el brazo equivocado. Sobre un bloque dañado no se puede poner otro bloque, por lo que una elección incorrecta de los brazos puede llevar a estados de camino sin salida.

Parking

El dominio *Parking* fue también uno de los dominios utilizados en la sección de aprendizaje de la IPC-6. El dominio consiste en aparcar coches en una serie de bordillos de la calle con la restricción de que pueden estar aparcados como máximo dos coches por bordillo, uno detrás de otro. Las metas en el dominio son diferentes configuraciones de los coches aparcados, lo que implica conducir los coches de unos bordillos a otros para colocarlos correctamente. Este dominio se puede ver como un dominio isomorfo a un *Blocksworld* hipotético en el que hay varias mesas y en las que sólo se pueden construir torres de dos bloques. El dominio es no dirigido porque siempre se puede devolver un coche a su posición anterior. Por otro lado, no puede determinarse sin una evaluación empírica si el dominio posee *plateaus* acotados.

Dominio Logistics

Este dominio fue desarrollado en su versión original por Manuela Veloso como uno de los dominios de prueba para ANALOGY (Veloso and Carbonell, 1993). Fue adoptado por la comunidad de la IPC y ha sido utilizado en las IPC-1 e IPC-2. El *Logistics* es un dominio de transportación que consiste en llevar un conjunto de paquetes a su destino correspondiente, que puede ser un aeropuerto o una oficina postal. Estas localidades pueden estar a su vez en diferentes ciudades. Para transportar los paquetes es necesario utilizar los medios de transporte que son los camiones y los aviones. Los camiones mueven paquetes dentro de una ciudad y los aviones entre ciudades. La versión de la IPC define en su jerarquía seis tipos reales y contiene seis acciones: Cargar y descargar un avión, cargar y descargar un camión, volar un avión y conducir un camión. El dominio es completamente reversible y tiene una distancia de salida de *plateau* acotada.

Dominio Mystery Prime

El dominio *Mystery'*, también identificado como *mprime*, es un dominio de transporte creado para la IPC-1 en 1998. Consiste en mover objetos de una localidad a otra en un grafo. Los objetos se pueden montar y desmontar de los vehículos,

siempre sin exceder su capacidad. Los vehículos pueden moverse entre localidades gastando una unidad de combustible al cambiar de localidad. El dominio se diferencia de la versión original del *Mystery* en que en la nueva versión el combustible puede moverse de una localidad a otra. La dificultad principal del dominio radica en la presencia de caminos sin salida durante la búsqueda producidos por el gasto irreparable de combustible.

Dominio Portcrane

El dominio *Portcrane* es un dominio desarrollado para esta tesis, con el objetivo de mostrar que ciertas características de problemáticas reales son idóneas para la representación de dominios en los que una técnica de aprendizaje como CBR puede ser muy útil a la hora de buscar soluciones a los problemas. La tarea del dominio consiste en desmontar de un barco una serie de contenedores para colocarlos primero en una zona de tránsito y luego en una zona del puerto que se especifica como meta. Cada contenedor según su tipo debe conservar ciertos atributos como el estar "congelado" o "asegurado". Los contenedores se desmontan con grúas *portainers* y después en el puerto se mueven con grúas *transtainers*. Ambos tipos de grúas pueden levantar los contenedores de diferentes formas según el tipo de contenedor. La dificultad del dominio reside en identificar la acción correcta según el tipo de contenedor, porque de lo contrario habría que incluir en el plan acciones de reparación por las equivocaciones, representadas en las acciones de congelar o asegurar un contenedor. En el Apéndice B se muestra el dominio en PDDL y un ejemplo de un problema. El dominio se clasifica como inofensivo porque no es posible volver a montar un contenedor en el barco, pero esto nunca es una meta del problema. Por otro lado, sería necesario un estudio adicional para determinar si el dominio tiene o no *plateaus* acotados.

Dominio Satellite

El dominio *Satellite* es una simplificación del proceso que realizan los satélites de observación espacial. Fue utilizado en las competiciones IPC-3 e IPC-4. Consiste en capturar una serie de imágenes utilizando un conjunto de satélites que pueden tener diferentes instrumentos con diferentes modos de uso, que a su vez necesitan ser calibrados antes de tomar imágenes en un modo particular. Las acciones involucradas en el dominio son: encender un instrumento de un satélite, apagarlo, girar un satélite hacia una dirección, calibrar un instrumento y capturar una imagen. El dominio no es reversible, pero las acciones que no tienen una acción inversa, no borran hechos que sean necesarios en alguna precondición de otra acción.

Dominio Rovers

El dominio *Rovers* es una simplificación de las actividades que realizan los vehículos autónomos de exploración sobre la superficie de Marte. Fue utilizado

en las competiciones IPC-3 e IPC-5. El dominio consiste en navegar el *rover* por diferentes puntos, tomando muestras de suelo o de rocas. Además, se pueden tomar imágenes de diferentes objetivos, primero calibrando el equipo de tomar imágenes. La información de las muestras o de las imágenes debe ser enviada a un dispositivo de transmisión que está fijo en algún punto de la superficie. Se definen en total nueve acciones: navegar el rover, tomar muestras de suelo o roca, calibrar el equipo, tomar imagen, vaciar el contenedor de muestras y enviar la información recogida para muestras, rocas o imágenes. El dominio no es reversible, pero de existir algún camino sin salida en la búsqueda, éste sería reconocido por la función heurística del plan relajado.

4.3.2. Problemas de Evaluación

Los problemas de evaluación son un conjunto de problemas de prueba que se tiene por cada dominio. Para este trabajo se han generado conjuntos de cuarenta problemas de dificultad incremental utilizando los generadores aleatorios de problemas que están disponibles en las webs de las IPCs y que son utilizados habitualmente por la comunidad. El generador del *Portcrane* se desarrolló con una filosofía similar a los demás. El aumento de la dificultad por problema es difícil de determinar debido a que cada generador tiene una serie de parámetros particulares a cada dominio. En dominios como el *Blocksworld*, la dificultad es fácil de incrementar porque es proporcional al número de bloques y a la altura media de las torres en los estados inicial y final. En el resto de dominios donde hay varios tipos de objetos y varias clases de metas, no hay una forma clara de definir sistemáticamente el incremento de dificultad. En general, este incremento se ha definido aumentando de forma gradual el número de objetos por tipo, de manera similar a como se ha hecho para generar los conjuntos de prueba de las IPCs. Como información ilustrativa se muestran en la Tabla 4.2 los valores mínimos y máximos entre los que oscilan los problemas en términos de objetos, metas y literales del estado inicial.

Dominio	Objetos		Literales		Metas	
	min.	max.	min.	max.	min.	max.
Blocksworld	6	25	8	32	3	20
Matching-bw	7	26	20	82	3	21
Parking	9	52	12	70	5	34
Logistics	15	115	13	107	6	80
Mystery'	18	71	29	116	5	30
Portcrane	10	89	25	228	6	118
Satellite	14	160	5	407	1	87
Rovers	20	147	92	3695	4	46

Tabla 4.2: Características de los problemas de evaluación.

Se decidió generar conjuntos de prueba nuevos y no utilizar directamente los conjuntos de las IPCs, porque estos conjuntos, al ser muchos de competiciones

diferentes, tienen un número diferente de problemas. Dado que los conjuntos que se han generado tienen todos 40 problemas, esto permite una mayor facilidad para la interpretación de los resultados en conjunto. Adicionalmente, el sistema de evaluación por puntos utilizado en la última IPC necesita esta característica para realizar una evaluación justa.

4.3.3. Otras Variables de Evaluación

De las variables independientes queda comentar el tiempo límite de ejecución. En este sentido, la comunidad ha ido incrementando paulatinamente este límite y en la actualidad se suelen reportar los experimentos a 1800 segundos. No hay una justificación aparente para la elección de este tiempo, ya que este valor tiene que ir en consonancia con el conjunto de problemas de prueba. Una posible explicación radica en dar a los planificadores un tiempo considerable para después sólo evaluar la calidad de las soluciones. Para la evaluación en este trabajo se decidió dar un límite de tiempo de 900 segundos por dos razones. La primera, se considera que ese tiempo es suficiente para mostrar el comportamiento de diferentes técnicas sin el perjuicio de invertir demasiado tiempo durante la experimentación. La segunda razón es que éste fue el límite utilizado para la sección de aprendizaje de la última IPC y puede que se utilice como referente para las futuras investigaciones en planificación y aprendizaje.

El resto de detalles de la evaluación no son consideradas variables independientes como tal, pero se suelen especificar en la descripción de los experimentos porque el resultado final de las mediciones tiene dependencia sobre estos detalles. Se utilizará un límite de memoria RAM de 1GB para cada problema y los experimentos se ejecutarán en un PC con un procesador Intel Pentium 4 a 3.0Ghz. Se utilizará la implementación LISP de SBCL en su versión 1.0.21 sobre un sistema operativo Linux. Al final de cada capítulo subsiguiente se reportarán los resultados correspondientes a las técnicas presentadas, siempre utilizando los detalles expuestos en este capítulo en lo referente a los dominios, problemas y otros detalles de evaluación.

Capítulo 5

Sistema de Planificación

El desarrollo de un sistema de Razonamiento Basado en Casos aplicado a la planificación heurística implica que el desarrollo del sistema de aprendizaje se realizará sobre un planificador heurístico. Para este trabajo se optó por la opción de desarrollar el planificador heurístico re-implementando las principales técnicas y algoritmos del estado del arte, con una filosofía que permita una fácil integración con el sistema CBR, u otros sistemas de aprendizaje que se deseen incluir posteriormente. En este capítulo se presentan los elementos de SAYPHI, el planificador heurístico que se ha desarrollado para que sirva como base al sistema CBR. Al final del capítulo se muestra una evaluación del planificador utilizando las heurísticas que posee.

5.1. El Planificador Heurístico

Un planificador heurístico es un sistema de planificación que resuelve sus tareas mediante algoritmos de búsqueda tradicionales utilizando como guía funciones heurísticas desarrolladas para el campo de la planificación. En este sentido un planificador heurístico consta de tres partes principales:

1. **Intérprete**: Traduce la tarea de planificación descrita en los ficheros de dominio y problema a estructuras instanciadas que utilizarán los algoritmos de búsqueda.

2. **Heurísticas**: Es el conjunto de funciones de evaluación, independientes o dependientes del dominio que guiarán la búsqueda de los algoritmos.

3. **Algoritmos**: Implementaciones de los algoritmos de búsqueda que resuelven el problema de planificación.

En las siguientes secciones se explicarán las tareas de planificación desde un punto de vista formal y cómo esto se corresponde con la representación de PDDL. Después se explicarán el funcionamiento de las heurísticas, de los algoritmos de búsqueda y algunos detalles de implementación en SAYPHI.

5.2. Intérprete de PDDL

El lenguaje de representación para el planificador es el PDDL, que es el lenguaje estándar para los planificadores actuales. Se utiliza el requisito `:typing`, que es una extensión del PDDL sobre el lenguaje STRIPS, que permite definir una jerarquía de tipos para los objetos del mundo. Adicionalmente se puede utilizar el requisito `:fluents` para indicar otra extensión de PDDL en el que se pueden incluir variables numéricas para el modelo de planificación basada en costes [1].

Al igual que se describió en la Sección 2.2, se ha definido un problema de planificación como la tupla (L, A, I, G) en la que L es el conjunto de todos los literales posibles con los que se definen los estados del mundo, A es el conjunto de acciones, $I \subseteq L$ es el estado inicial del problema y $G \subseteq L$ es el conjunto de literales que definen las metas del problema. Una acción $a \in A$ se define como $(pre(a), add(a), del(a))$, en donde $pre(a) \subseteq L$ es el conjunto de literales que son precondición para que la acción sea aplicable, $add(a) \subseteq L$ es el conjunto de literales que pasan a ser ciertos tras aplicar la acción y $del(a) \subseteq L$ el conjunto de literales que se eliminan del estado al aplicar la acción.

De esta forma las transiciones de estado se definen como una función que aplica una acción a a un estado S.

$$S_{res} = aplicar(a, S) = S/del(a) \cup add(a)$$

Así, el estado S_{res} es el estado resultado de eliminar la lista de literales en los borrados $del(a)$ y de incluir la lista de añadidos $add(a)$. El objetivo de resolver un problema de planificación es encontrar un plan $\mathcal{P} = \{a_1, a_2, \ldots, a_n\}$, como el conjunto de acciones que tras aplicarlas consecutivamente se obtiene un estado en el que son ciertos los literales de G.

El conjunto de literales L definidos para la tarea de planificación viene descrito de forma compacta en los ficheros del dominio y del problema. En el dominio, por la jerarquía de tipos y por un esquema o plantilla tipificada de predicados (las relaciones que existen entre los objetos del mundo). En el problema, por el conjunto de objetos definidos para esa tarea de planificación. Igualmente, el conjunto de acciones A está definido por el conjunto de acciones del dominio que son también esquemas tipificados de las listas de precondiciones añadidos y borrados. SAYPHI, al igual que el resto de planificadores que hacen búsqueda hacia adelante, instancia los esquemas para eliminar la carga computacional de las unificaciones de fórmulas, necesarias en lógica de predicados. Como resultado, el lenguaje se transforma a uno de lógica proposicional, y los conjuntos L y A están representados explícitamente en las estructuras que utiliza el planificador.

Por eficiencia, el espacio de estados se representa con vectores de bits, uno asociado a cada predicado. Cada posición del vector indica si el literal asociado es cierto o falso en el estado. Se relaciona un literal a una posición calculando la

[1] SAYPHI soporta el modelo de planificación basada en costes. Los detalles de ese modelo no serán tratados aquí porque no son necesarios para el trabajo que se quiere abordar con el sistema CBR.

combinación correspondiente al permutar los objetos, primero en el orden en el que aparecen en la descripción del problema, y segundo por el orden de las variables del predicado. Por ejemplo, si el dominio define un predicado `(en ?x - paquete ?y - lugar)` y el problema define los objetos `A`, `B` y `C` como tipo `paquete`, y los objetos `L1` y `L2` como tipo `lugar`, se puede ver las posiciones asociadas a su vector de bits en la tabla de la Figura 5.1.

Número	Literal	Posición en el vector de bits
0	(en A L1)	100000
1	(en B L1)	010000
2	(en C L1)	001000
3	(en A L2)	000100
4	(en B L2)	000010
5	(en C L2)	000001

Figura 5.1: Ejemplo de codificación de literales de estado en SAYPHI.

Por tanto, el vector tendrá tantos bits como posibles instanciaciones de predicados haya, y tendrá bits en 1, los literales que sean ciertos en el estado que se esté representando. Por ejemplo la Figura 5.2 muestra un posible estado utilizando el predicado del ejemplo anterior. Los tres literales se representan con bits a 1 en sus posiciones correspondientes.

Estado	Vector de bits
{(en A L1) (en B L1) (en C L2)}	110001

Figura 5.2: Ejemplo de un estado y su correspondiente vector de bits en SAYPHI.

Por otro lado, las acciones instanciadas tienen asociado un número entero que se calcula siguiendo el mismo procedimiento de permutación mostrado para los literales. La parte de pre-proceso realizada por el intérprete termina con la construcción de un grafo de conectividad que será utilizado para realizar un cálculo más eficiente de las funciones heurísticas. Este grafo consiste en tener una estructura redundante que almacena la información inversa contenida en las listas de precondiciones, añadidos y borrados. En este caso se determina para cada literal en L:

1. $Prec\_of(l) \subseteq A$: Es el subconjunto de acciones instanciadas en las que el literal $l \in L$ está como precondición de la acción.

2. $Added\_by(l) \subseteq A$: Es el subconjunto de acciones instanciadas que añaden el literal.

3. $Deleted\_by(l) \subseteq A$: Es el subconjunto de acciones instanciadas que borran el literal.

5.3. Heurísticas

Como se describió en la Sección 2.2.3, el éxito de la planificación heurística radica en la posibilidad de construir funciones heurísticas independientes del dominio de forma automática. La idea original de (McDermott, 1996) consiste en plantear una tarea de planificación simplificada en la que no se consideran los borrados de las acciones del dominio.

De esta forma, a partir de un problema de planificación $P = (L, A, I, G)$ se puede definir un problema relajado $P' = (L, A^+, I, G)$ en donde A^+ es el conjunto de acciones relajadas a^+ y

$$a^+ = (pre(a), add(a)) \text{ para cada } a \in A$$

Dado que el cálculo de las funciones heurísticas se realiza para cada nodo en el proceso de búsqueda, se asume que el estado I pasa a ser el estado actual de la búsqueda, en el sentido de que el problema relajado se resuelve partiendo del estado actual hasta conseguir las metas. Por eso, se redefine el problema relajado $P' = (L, A^+, S, G)$ con S como estado actual de la búsqueda. Las diferentes aproximaciones para resolver este problema relajado dan como resultado diferentes heurísticas. En la siguientes secciones se explicarán las implementadas en SAYPHI.

5.3.1. Heurística del Plan Relajado

La heurística del plan relajado fue introducida en el planificador FF (Hoffmann and Nebel, 2001). La heurística consiste en resolver el problema relajado P' con el algoritmo del planificador GRAPHPLAN (Blum and Furst, 1995) y tomar como estimación heurística la longitud del plan encontrado. El algoritmo de GRAPHPLAN queda muy simplificado a la hora de aplicarse al problema relajado porque no tiene que considerar los conflictos en el grafo de planificación, llamados *mutex*. Como en este trabajo interesa el GRAPHPLAN como estimador heurístico, sólo se describirá la versión simplificada del algoritmo.

El algoritmo para obtener el plan relajado consta de dos fases. La primera es la expansión del grafo relajado de planificación y la segunda la extracción del plan relajado a partir del grafo. El grafo de planificación se representa mediante una secuencia P_0, A_0, ...,P_{t-1}, A_{t-1}, P_t en donde cualquier capa i del grafo esta formada por P_i, un conjunto de proposiciones y por A_i, un conjunto de acciones. Cada capa se construye de forma incremental empezando por $P_0 = S$ como la primera capa de proposiciones, y A_0 como la capa que contiene las acciones aplicables en el estado S. Después, de forma iterativa, se inserta una nueva capa con los añadidos de todas las acciones de la capa anterior y el conjunto de acciones que pasan a ser aplicables. El algoritmo falla si en un determinado punto antes de alcanzar la metas no se pueden insertar más proposiciones. En este caso no existe solución al problema relajado y la estimación heurística se indica como ∞. La Figura 5.3 muestra el algoritmo para la expansión del grafo de planificación.

Expansion_RPG (G,S,A): *RPG*

G: Metas
S: Estado actual
A: Acciones

$t = 0$; $P_0 = S$
while $G \nsubseteq P_t$ **do**
  $A_t = \{a \in A \mid prec(a) \subseteq P_t\}$
  $P_{t+1} = P_t \cup add(a), \forall a \in A_t$
  if $P_{t+1} = P_t$ **then**
    return "fallo"
  $t = t + 1$
return $[P_0, A_0, \ldots, A_{t-1}, P_t]$

Figura 5.3: Algoritmo para la expansión del grafo de planificación relajado.

Extraccion-plan-relajado (G,RPG): *plan-relajado*

G: Metas
RPG: Grafo de planificación relajado

$PR \leftarrow []$, $m \leftarrow$ *numero-capas*(RPG)
for $i = 1 \ldots m$ **do**
  $G_i = \{g \in G |$ *capa*$(g) = i\}$
for $i = m \ldots 1$ **do**
  for all $g \in G_i$ **do**
    $a \leftarrow$ accion en RPG con $g \in add(a)$ y *diff*(a) mínima
    for all $f \in pre(a)$, *capa*$(f) \neq 0$ **do**
      $G_{capa(f)} \leftarrow G_{capa(f)} + f$
    $G_i \leftarrow G_i - add(a)$
    $G_{i-1} \leftarrow G_{i-1} - add(a)$; **interacciones positivas**
    $PR \leftarrow PR + a$
return PR

Figura 5.4: Algoritmo para la extracción del plan relajado.

Para la extracción del plan relajado se sigue el algoritmo mostrado en la Figura 5.4. Primero, las metas en G se separan como conjuntos de metas G_i en donde cada meta g_i debe conseguirse en la capa P_i del grafo (P_i es la primera capa donde aparece g_i). El proceso comienza por la última capa y en cada capa i se selecciona por cada meta en G_i una acción de la capa A_{i-1} que consiga ese hecho. Si existe más de una acción que consigue un hecho, se selecciona la que tenga menor dificultad en las precondiciones de las acciones. Hoffmann explica esta selección como la heurística de la dificultad, que es la base para la función heurística explicada en la siguiente sección. Después, las precondiciones de la acción seleccionada se introducen en el conjunto de metas de su capa correspondiente. Cada vez que se selecciona una acción, sus añadidos se eliminan (en caso de que existan) de las metas en G_i e G_{i-1}, para que éstas ya no tengan que producir nuevas selecciones de acciones que las consigan. Este detalle permite tomar en cuenta las interacciones positivas entre las acciones, de modo que no haya más acciones de las necesarias en el plan relajado, sino el conjunto adecuado correctamente ordenado. Sin embargo, decidir la ordenación óptima es un problema NP-completo y aquí lo que interesa es obtener una solución rápida para calcular el valor heurístico. El orden arbitrario de la resolución de metas dentro de una misma capa y la heurística de la dificultad son estrategias para extraer rápidamente un plan relajado. (Hoffmann and Nebel, 2001) probaron que este algoritmo podía encontrar una solución al problema relajado sin un punto de retroceso (*backtracking*) en la búsqueda sobre el grafo relajado.

La heurística de la dificultad es el criterio para seleccionar una acción de entre varias posibilidades durante la extracción del plan relajado. Este caso se da si en el momento de resolver una meta o sub-meta g, existen al menos una a_1 y una a_2 tal que $g \in add(a_1)$ y $g \in add(a_2)$. La heurística consiste en preferir la acción que tenga la menor dificultad en las precondiciones. Esta dificultad se determina en función de la primera capa en la que aparece cada precondición, y se calcula con la fórmula:

$$\text{dificultad}(a) = \sum_{p \in prec(a)} \min\{\, i \mid p \in P_i \,\} \tag{5.1}$$

Después de la extracción del plan relajado se obtiene el valor para la función heurística del plan relajado, que resulta de contar el número de acciones que se encuentren en el plan relajado. Formalmente, considerando la secuencia de acciones $\mathcal{P}^+ = \{a_1, a_2, \ldots, a_n\}$ como el plan que resuelve del problema relajado $P' = (L, A^+, S, G)$ a partir de un estado S, la función heurística del plan relajado o bien la heurística de FF se obtiene por

$$h^{\text{FF}}(S) = |\mathcal{P}^+| \tag{5.2}$$

La longitud del plan óptimo para P' es una heurística admisible para el problema original, pero como ya se ha dicho, obtener el plan óptimo durante el proceso de extracción del plan es un problema NP-duro. La aproximación utilizada para extraer rápidamente un plan relajado hace que la heurística sea no admisible.

5.3.2. Heurística de la Dificultad del Plan Relajado

Como parte de este trabajo de tesis se ha desarrollado una heurística complementaria a la heurística del plan relajado, integrando al valor heurístico la dificultad de las precondiciones de las acciones. Esta heurística fue motivada por las siguientes razones:

- Los empates en la evaluación heurística con h^{FF} se producen cuando dos o más estados tienen igual longitud en el plan relajado. Sin embargo, estos planes relajados tienen frecuentemente acciones diferentes.

- El proceso de expansión del grafo relajado de planificación y extracción del plan relajado involucran muchos cálculos que se desechan, al sólo tomarse en cuenta la longitud del plan relajado y la heurística de las acciones útiles (explicada más adelante). Parte de la información generada en estos procesos debería utilizarse en favor de la búsqueda.

En este sentido, interesa definir una heurística que pueda diferenciar estados que tengan el mismo valor heurístico pero que tengan diferentes dificultades en las precondiciones de sus acciones. En el diseño de la nueva función heurística interesa preservar el criterio de que un estado S_1 es mejor que S_2 porque $h(S_1) < h(S_2)$ e interesa diferenciar sólo los casos en los que $h(S_1) = h(S_2)$. Para esto, el valor complementario que represente la dificultad de las acciones del plan relajado debe estar en el intervalo de $[0, 1)$ y debe sumarse al valor heurístico normal. La dificultad de las precondiciones de las acciones del plan relajado queda representada por la suma de las dificultades utilizadas en la extracción del plan relajado según la formula 5.1. De esta forma la heurística del *plan relajado* + *dificultad* o h^{diff} para un estado S se obtiene con la fórmula:

$$h^{\mathrm{diff}}(S) = h^{\mathrm{FF}}(S) + complemento\_h(S) \tag{5.3}$$

donde

$$complemento\_h(S) = \begin{cases} (1 - \frac{1}{\sum\limits_{i=1\ldots n} \mathrm{dificultad}(a_i)}) & \text{Si } \sum\limits_{i=1\ldots n} \mathrm{difficultad}(a_i) > 0 \\ 0 & \text{el resto de los casos} \end{cases}$$

y n es el número de acciones del plan relajado. En el caso de que la suma de las dificultades de un plan relajado sea igual a cero, todas las acciones son aplicables en la primera capa del grafo. En este caso las dificultades no permiten preferir entre dos planes relajados con acciones diferentes. En la sección de resultados se analizará el comportamiento de esta nueva heurística frente a h^{FF} utilizando diferentes algoritmos. Resultados previos sobre este trabajo se encuentran en (De la Rosa and Fuentetaja, 2006).

5.3.3. Heurística de Acciones Útiles

Las acciones útiles, o mejor conocidas como las *helpful actions*, es una heurística de control que sirve para diferenciar las acciones que parecen más prometedoras para la búsqueda, basándose en información generada durante la construcción del plan relajado. Una acción es marcada como *helpful* si añade alguna proposición que necesita alcanzar el plan relajado en la segunda capa de proposiciones. Formalmente, si el conjunto G_1 es el conjunto de metas (o proposiciones necesarias para otras acciones) que se deben alcanzar en la capa P_1 del grafo relajado de planificación, el conjunto de acciones *helpful* para cualquier estado S de la búsqueda esta determinado por:

$$helpful(S) = \{a \in A_0 \mid add(a) \cap G_1 \neq \emptyset\}$$

Esta definición implica que todas las acciones aplicables del plan relajado son también *helpful*, porque fueron escogidas precisamente para alcanzar alguna precondición en G_1. La heurística de las acciones útiles es utilizada durante la búsqueda como técnica de poda anticipada, de modo que directamente sólo se consideran como sucesores de un estado las acciones marcadas como *helpful*.

5.4. Algoritmos de Búsqueda

El sistema de planificación propuesto implementa diferentes algoritmos de búsqueda informada que pueden ser utilizados con las heurísticas explicadas en la sección anterior.

5.4.1. Escalada

El algoritmo de escalada elige en cada nivel del árbol de búsqueda el mejor nodo en función del valor heurístico de todos los nodos de ese nivel (Russell and Norvig, 2002). Fue utilizado por primera vez en planificación automática en HSP (Bonet and Geffner, 2001). Para planificación se suelen incluir algunas mejoras al algoritmo tradicional para hacer la búsqueda más eficiente. La primera es mantener en una estructura de acceso rápido (*tabla-hash*) los estados que pertenecen al camino actual para evitar nodos repetidos y evitar que la búsqueda entre en ciclos debido a la existencia de mínimos locales en la función heurística. SAYPHI incluye además, un *backtracking* cronológico, para que el algoritmo pueda explorar caminos alternativos cuando un nodo se quede sin sucesores. Esta situación puede darse por estados con evaluación ∞, estados repetidos o estados que han sido podados por no estar marcados como *helpful*.

5.4.2. Enforced Hill-Climbing

El algoritmo Enforced Hill-Climbing o EHC fue utilizado en planificación como alternativa al algoritmo de Escalada. EHC fue popularizado por el planificador

FF. El algoritmo consiste en hacer de forma iterativa búsquedas en amplitud a partir de un estado s, hasta encontrar un estado s' en el árbol que cumpla que $h(s') < h(s)$. Cuando este estado es encontrado se reinicia la búsqueda en amplitud a partir de ahí. El proceso se puede entender como la búsqueda progresiva de estados que van mejorando el valor heurístico. A profundidad uno implica que pueden evaluarse menos nodos de los generados. Sin embargo, a más profundidad implica un crecimiento exponencial en el número de nodos, razón por la cual los problemas con largos *plateaus* son de gran dificultad para el algoritmo.

Después que se encuentra un nodo que mejora la evaluación heurística, no se vuelve a considerar ninguna decisión por encima de ese nodo. Para evitar que el algoritmo falle por quedarse sin sucesores, SAYPHI re-expande el último nodo de la búsqueda en amplitud para incluir los nodos que no están marcados como *helpful* [2]. La comparación de estados repetidos sólo se realiza con los nodos de la búsqueda en amplitud actual. Esto se justifica porque por alguna elección pasada, puede que sea necesario visitar nuevamente algún estado para poder encontrar algún nodo de escape de *plateau*.

La propiedad avariciosa del EHC resulta interesante, al poder diferenciarse el momento de la generación de nodos del momento de su evaluación. Si en un nivel cualquiera de la búsqueda en amplitud se encuentra un sucesor que mejora la heurística, el resto de sucesores en el mismo nivel no es evaluado, lo que presenta una oportunidad para establecer heurísticas de control adicionales, en las que se determine cuál es el orden en que se deben evaluar los nodos. En problemas con factor de ramificación grande, este orden puede ser determinante. Sin embargo, heurísticas para ordenar las evaluaciones de los nodos no han sido estudiadas en planificación.

5.4.3. WBFS

El algoritmo WBFS (*Weighted Best-First Search*) es una generalización del algoritmo de mejor primero. En este algoritmo, en cada estado, se generan los sucesores, se evalúan con la función heurística y se introducen en la lista abierta. El siguiente nodo de la búsqueda es el primero de la lista después de ordenarla por una función de evaluación. En el WBFS la función de evaluación es $f(s) = w_g g(s) + w_h h(s)$. En planificación, típicamente la w_g se utiliza con valor 1 y la w_h suele variarse para impulsar la búsqueda en mayor o menor medida hacia la solución. Este sesgo con frecuencia compromete la calidad de la solución; a mayor w_h, peor calidad. Con $w_g = 1$ y $w_h = 1$ el WBFS utilizaría la función de evaluación del algoritmo A*. Según pruebas experimentales, (Bonet and Geffner, 2001) indican que para las heurísticas de planificación, los valores entre 2 y 10 para w_h no producen diferencias significativas. Otro ejemplo de utilización fue

[2] El fallo de EHC en FF implica cambiar al algoritmo WBFS. La versión posterior, METRIC-FF implementa una estrategia parecida para recuperar la búsqueda. En su caso la re-expansión con los estados no *helpful* se realiza desde el último nodo que mejora el valor heurístico.

METRIC-FF (Hoffmann, 2003), que utilizó $w_h = 5$ cuando fue presentado para la IPC del 2002.

SAYPHI permite utilizar el WBFS con o sin poda de las acciones útiles. Además incluye la optimización de no re-expandir un nodo repetido que se encuentre con una menor g. En este caso, se actualizan los caminos de todos los descendientes del nodo en la lista cerrada. Los caminos construidos previamente tienen después de la actualización, el camino más corto encontrado.

5.4.4. DFBnB

El algoritmo DFBnB (*Depth-First Branch and Bound*) es un algoritmo que, dada una cota superior de coste, hace una búsqueda de primero en profundidad podando las ramas siempre que se exceda el coste de la mejor solución encontrada hasta el momento. Para ser utilizado en planificación, el algoritmo presenta el inconveniente de encontrar el umbral superior de coste por lo que tiene sentido encontrar una primera solución con un algoritmo como EHC, e ir mejorando la calidad de las soluciones encontradas con la técnica de expansión y acotamiento (*branch and bound*). El algoritmo puede mejorarse considerablemente utilizando a su vez técnicas de ordenación de nodos, como por ejemplo la función de evaluación del A*. Los sucesores de un nodo se ordenan por $f(n)$ y se podan todos los que su evaluación exceda el coste total de la mejor solución encontrada. Para heurísticas admisibles estas técnicas de ordenación y poda garantizan encontrar la solución óptima. Para el caso de las heurísticas de planificación expuestas, la ordenación permite evaluar rápidamente grandes regiones del árbol de búsqueda, pero podrían podar ramas en las que la función heurística esté sobrestimando el coste real.

En SAYPHI, la implementación del DFBnB obtiene su cota superior de coste con la solución del algoritmo EHC y en caso de que éste falle, con un WBFS con la función de evaluación $f(n) = g(n) + 3h(n)$. El algoritmo permite además encontrar múltiples soluciones, útil para los casos en que una técnica de aprendizaje necesite evaluar varias soluciones del mismo problema.

5.4.5. Técnicas *lookahead*

El *lookahead* es una modificación a los algoritmos de búsqueda en el que se incluye en la expansión de los nodos, uno o varios nodos adicionales que intentan vislumbrar zonas del árbol de búsqueda más profundas, al crear estados en los que se han aplicado varias acciones. Esta técnica fue utilizada por primera vez en planificación heurística por el planificador YASHP (Vidal, 2004) cuando introdujo nodos *lookahead* a una búsqueda de mejor-primero, utilizando las acciones del plan relajado para construir los estados *lookahead*.

La forma estándar de crear los estados *lookahead* es mediante una función que toma las acciones de un plan propuesto (el plan relajado para estos casos) e intenta aplicarlas sucesivamente. Dado que tras la aplicación de algunas acciones, otras pueden convertirse en aplicables, este proceso se repite iterativamente hasta que no

queden acciones del plan propuesto, o ya no queden más acciones aplicables. En YAHSP se incluyen además, ciertas variaciones a esta construcción, por ejemplo el activar o desactivar ciertas acciones durante la construcción. Estas técnicas fueron introducidas para mejorar el resultado de los nodos *lookahead* en ciertos dominios y no responden como tal a una justificación clara de uso. En SAYPHI se ha implementado sólo la versión estándar de los *lookahead* para tener un enfoque sencillo y limpio a la hora de introducir conocimiento de control aprendido. La Figura 5.5 muestra el pseudo-código para la generación de un nodo *lookahead* en su versión sin mejoras.

Generacion_lookahead (s,RP): *nodo-lookahead*

---

s: Estado actual
RP: Plan relajado propuesto

$la\_estado = s$; $la\_plan = []$; $la\_activo \leftarrow$ **TRUE**
while la_activo **do**
  $la\_activo \leftarrow$ **FALSE**
  for each $a \in RP$ **do**
    if *aplicable*(a, la_estado) **then**
      $la\_estado \leftarrow$ *aplicar*(a, la_estado)
      sacar a de RP y añadirlo a la_plan
      $la\_activo \leftarrow$ **TRUE**
return *crear-nodo-lookahead*(la_estado, lh_plan)

Figura 5.5: Algoritmo para la generación de un nodo lookahead a partir de un plan propuesto.

5.5. Experimentación y Resultados

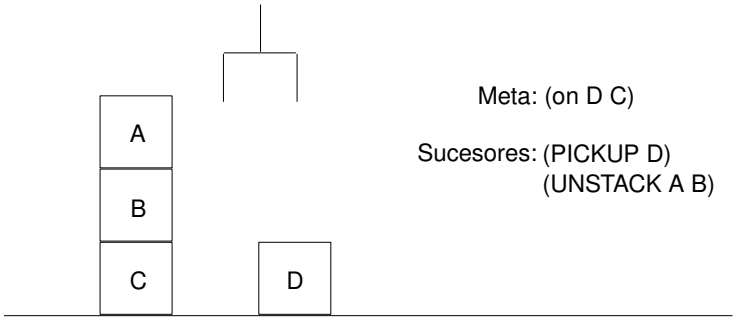
En esta sección se presentan los experimentos que se han realizado para analizar el comportamiento de la heurística h^{diff} en los algoritmos de Escalada, EHC y WBFS. En todos los casos se ha utilizado los dominios, problemas y demás detalles para evaluación explicados en la sección 4.3. Dentro de cada experimento se evaluará:

- **Algoritmo base**: Corresponde al algoritmo en su versión estándar utilizando la heurística h^{FF}.

- **Algoritmo con h^{diff}**: Corresponde al mismo algoritmo utilizando la nueva heurística.

5.5.1. Resultados en Escalada con h^{diff}

En el algoritmo de Escalada, la heurística h^{diff} servirá para discriminar entre los estados con la misma evaluación heurística pero con diferente dificultad en las precondiciones de las acciones del plan relajado. Dado que en este algoritmo se evalúan todos los candidatos tras la expansión de un nodo, se asegura que en cada selección siempre se escogerá el estado con menor valor heurístico, y de haber empate el estado con menor dificultad. La Figura 5.6 presenta un ejemplo en el *Blocksworld* que muestra este efecto.



(PICKUP D)		(UNSTACK A B)	
Acciones	Precondiciones	Acciones	Precondiciones
(PUTDOWN D) [0]	(holding D) [0]	(PUTDOWN A) [0]	(holding A) [0]
(UNSTACK A B) [1]	(on A B) [0] (clear A) [0] (handempty) [1]	(UNSTACK B C) [1]	(on B C) [0] (clear B) [0] (handempty) [1]
(UNSTACK B C) [2]	(on B C) [0] (clear B) [2] (handempty) [1]	(PICKUP D) [1]	(clear D) [0] (handempty) [1]
(STACK D C) [3]	(clear C) [3] (holding D) [0]	(STACK D C) [2]	(clear C) [2] (holding D) [2]

Figura 5.6: Ejemplo en el *Blocksworld* de planes relajados con diferente dificultad en las precondiciones de las acciones.

Partiendo del estado en la imagen, se desea alcanzar la meta `(on D C)`. Las únicas acciones que se pueden aplicar son `(PICKUP D)` y `(UNSTACK A B)`. Ambas acciones producen planes relajados de longitud cuatro, pero sin embargo tienen diferente suma en las dificultades. Las acciones y los literales tienen entre corchetes el nivel en que aparecen por primera vez en el grafo de planificación relajado. Según la fórmula 5.1 la acción `(PICKUP D)` tiene dificultad 7 y la acción `(UNSTACK A D)` tiene dificultad 6. Esto implica que `(UNSTACK A D)`

tendrá mejor valor heurístico que `(PICKUP D)` si se utiliza h^{diff}.

Este efecto permite al algoritmo de búsqueda reconocer que la acción `(PICKUP D)` no tiene sentido en este estado y de hecho, la presencia de `(PUTDOWN D)` en el plan relajado es sólo necesaria para liberar el brazo. Por el contrario `(PUTDOWN A)` aparece en el plan relajado del estado generado con `(UNSTACK A B)`, y es en efecto una acción del plan real, permitiendo así que la acción `(STACK D C)`, que consigue la meta, se pueda ejecutar en un nivel anterior del grafo. Este ejemplo muestra una particularidad en el *Blocksworld* de la que h^{diff} se puede aprovechar, pero de ningún modo esto implica que de forma general los estados con menor dificultad sean siempre la mejor elección de cara a encontrar una solución al problema. La heurística h^{diff} tiende a preferir estados con planes relajados más paralelos que estados con planes relajados más secuenciales. Intuitivamente, esta elección tiene sentido, porque supondría que hay más caminos para llegar hasta las metas. En muchos casos esto implica que será más fácil encontrar la solución, pero es un supuesto que no puede garantizarse en todos los casos.

La Tabla 5.1 muestra el número y porcentaje de problemas resueltos por cada técnica en los dominios de evaluación. h^{diff} ha mejorado el comportamiento del algoritmo de Escalada en cinco de los ocho dominios de evaluación. En el *Blocksworld* la situación expuesta en el ejemplo se repite con relativa frecuencia, pero no en todos los problemas. Las configuraciones de otras torres en los problemas más grandes pueden distorsionar el efecto durante el cálculo de la función heurística.

Dominio	Escalada		Escalada con h^{diff}	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	**34**	**85.0 %**	33	82.5 %
Matching-bw	11	27.5 %	**13**	**32.5 %**
Parking	**38**	**95.0 %**	37	92.5 %
Logistics	34	85.0 %	**37**	**92.5 %**
Mystery'	11	27.5 %	**12**	**30.0 %**
Portcrane	**37**	**92.5 %**	33	82.5 %
Satellite	38	95.0 %	**40**	**100.0 %**
Rovers	38	95.0 %	**39**	**97.5 %**

Tabla 5.1: Problemas resueltos para el experimento de Escalada con h^{diff}.

La Figura 5.7 muestra la distribución acumulada de tiempo en cuatro de los dominios de evaluación. En estas gráficas, el eje x representa el tiempo de CPU utilizado para resolver los problemas (expresado en escala logarítmica), y el eje y representa el porcentaje de problemas resueltos en un tiempo dado. Cada configuración evaluada produce una línea ascendente que llega hasta el porcentaje total de problemas resueltos justo en el mayor tiempo utilizado para resolver algún problema. Las líneas más a la izquierda indican que el algoritmo resuelve la misma cantidad de problemas en menos tiempo, por lo que puede considerarse que tiene un mejor rendimiento. Si las líneas no se cortan durante toda la gráfica se dice que el mejor algoritmo domina al otro.

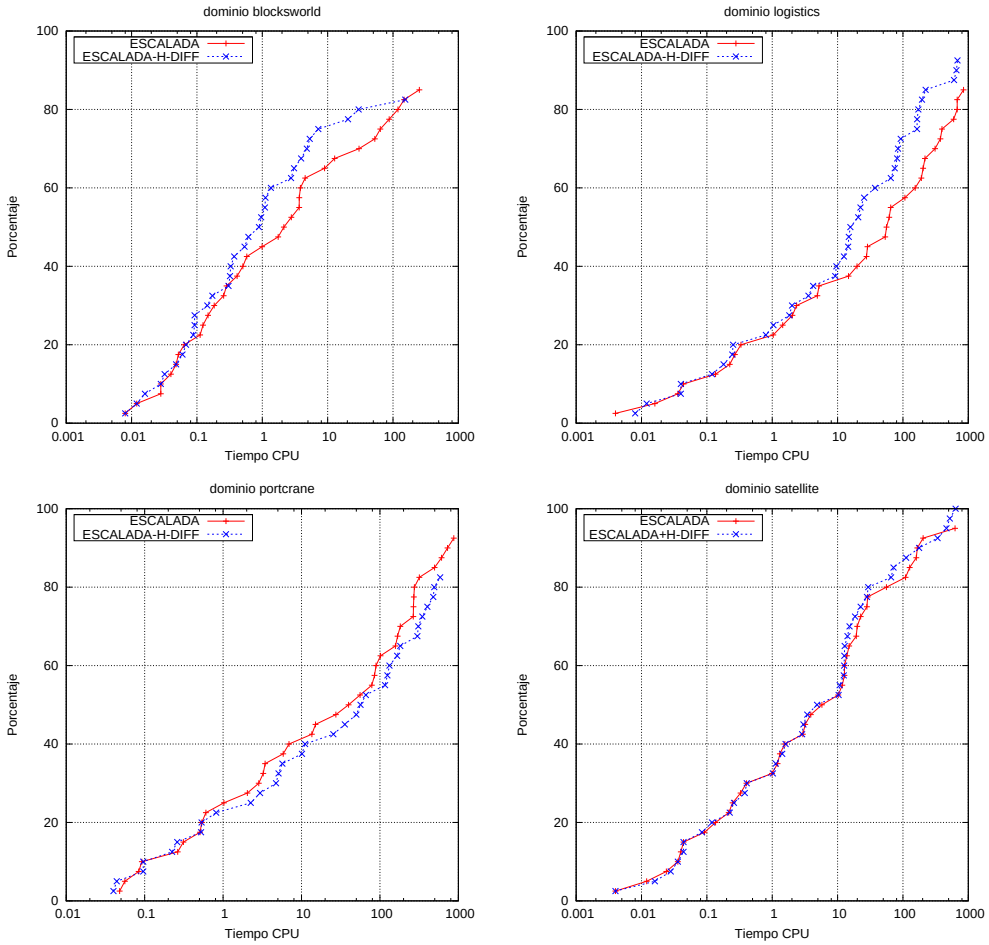


Figura 5.7: Distribución acumulada de tiempo de CPU para experimentos con Escalada y h^{diff}.

A pesar de que h^{FF} resuelve más problemas en el *Blocksworld*, se observa en la gráfica que durante la mayor parte h^{diff} domina la distribución acumulada de tiempo. También se presenta una mejora considerable en la longitud de los planes, apreciable en la distribución acumulada de longitud del plan mostrada en la Figura 5.8. En estas gráficas el eje x representa el número de acciones que tienen los planes y el eje y el porcentaje de problemas resueltos que se consiguen con la correspondiente longitud como límite. Estas gráficas se interpretan igual que las gráficas de tiempo.

De los 30 problemas del *Blocksworld* resueltos por ambas técnicas, h^{diff} obtiene un promedio de 198.7 acciones por plan, frente a las 652.8 de h^{FF}. Esta mejora es tan significativa porque en las soluciones con h^{diff} se evitan en muchas ocasiones las acciones sin sentido como las mostradas en el ejemplo de la Figura 5.6. A pesar de esto, los planes obtenidos son de mala calidad como se verá en

comparación con otros algoritmos, porque el espacio de búsqueda en este dominio tiene grandes zonas con mínimos locales en los que el algoritmo de Escalada transita de manera poco informada. Al final se consigue salir de estos *plateaus* porque el dominio es no dirigido y siempre se pueden deshacer las acciones erróneas.

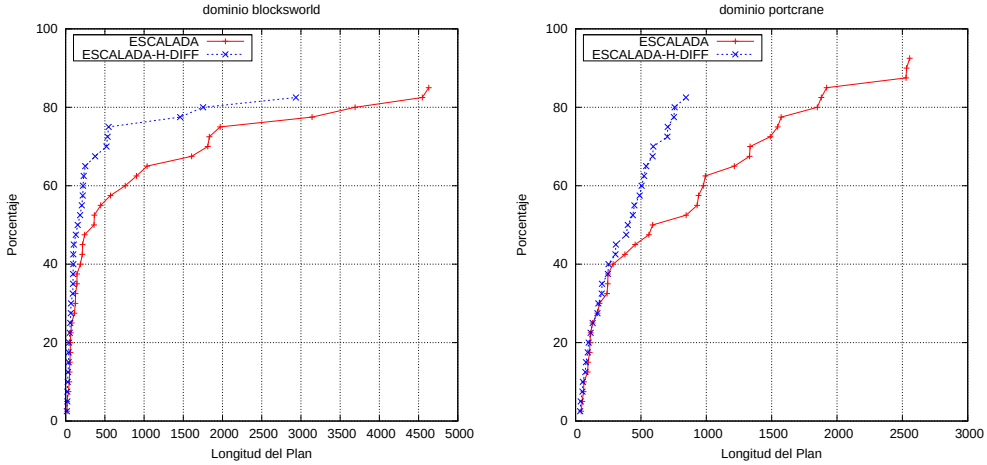


Figura 5.8: Distribución acumulada de longitud del plan en el *Blocksworld* y el *Portcrane* para experimentos con Escalada y h^{diff}.

En el *Logistics* se obtiene también una mejora generalizada en número de problemas resueltos, tiempo y longitud de los planes. En este caso, h^{diff} permite reconocer acciones sin sentido que llevan a cabo los medios de transporte (aviones y camiones). Todas las acciones que involucran mover un medio de transporte y que aparecen en el plan relajado son aplicables en el estado actual de la búsqueda (movimientos correctos e incorrectos). Sin embargo, un movimiento correcto permite realizar las acciones con los paquetes en capas más cercanas del grafo, lo que hace disminuir la suma total de las dificultades.

En el *Portcrane* se obtiene una mejora en la calidad de los planes (Figura 5.8) debido al reconocimiento de las acciones sin sentido de las grúas *transtainer*. Sin embargo, como se aprecia en la Figura 5.7, h^{FF} tiene un mejor comportamiento en tiempo. La explicación se encuentra en una particularidad del dominio. h^{diff} prefiere encadenar las acciones de un mismo contenedor porque permite conseguir las sub-metas en capas anteriores del grafo relajado de planificación. Por el contrario, h^{FF} prefiere levantar todos los contenedores que pueda, antes de moverlos. Esta diferencia hace que el árbol de búsqueda para h^{diff} tenga un mayor factor de ramificación, porque a cada paso tendrá siempre más grúas libres, multiplicado a su vez por las posibilidades que hay de levantar los contenedores de diferentes formas. En los problemas grandes esto se traduce en un aumento considerable en el número de nodos evaluados y por tanto más tiempo de CPU empleado. El beneficio en nodos que reporta la mejora de calidad no compensa el aumento en el factor de ramificación, y en resultados globales explica un menor número de problemas resueltos.

Esta particularidad del dominio no es por su tipología, sino más bien por el orden en que están declarados los operadores en el PDDL, que permite a h^{FF} elegir ocupar las grúas primero, porque se produce un empate con el resto de acciones aplicables.

Tanto en el *Satellite* como en el *Rovers* se identifican algunas acciones de movimiento que no son provechosas. En estos dominios, a diferencia de los de transporte, los empates en la función heurística que ocurren en estados que involucran movimiento de dispositivos de una localidad a otra, ocurren con mucho menos frecuencia. Por ejemplo, en el *Satellite*, si el instrumento no está calibrado, girar el satélite a la dirección de calibración o a la dirección de tomar una imagen producen planes relajados de igual longitud. h^{diff} reconoce el movimiento correcto (girar a la dirección para calibrar) porque las acciones en el plan relajado tienen menos dificultad, pero sin embargo, a partir de ahí, no se producirán más empates en el valor heurístico que involucren ese instrumento. Por este efecto, las acciones sin sentido son menos frecuentes, permitiendo a h^{FF} obtener planes de calidad razonable. En este sentido h^{diff} consigue mejoras de la misma forma que en los dominios de transporte pero el beneficio en tiempo o calidad es menos significativo. Por otro lado, a pesar de que con h^{diff} hay pequeñas mejoras en los dominios con caminos sin salida irreconocibles (*Mystery'* y *Matching Blocksworld*), el factor determinante para los malos resultados es la principal característica de la técnica de Escalada: tomar decisiones de forma local. Las mejoras respecto a h^{FF} residen en los situaciones que se producen por la semejanza a los dominios de su tipología (construcción y transporte). El resto de resultados para Escalada con h^{diff} se encuentran el la Figura C.1 del apéndice C.

En general, la mala calidad de los planes en algunos dominios, y la incapacidad de resolver problemas en los dominios con caminos sin salida, hacen del algoritmo de Escalada una alternativa de búsqueda poco considerada en la planificación heurística. Sólo la primera versión de HSP (Bonet and Geffner, 2001) utilizó el algoritmo de Escalada. A partir de entonces los planificadores han optado por otros algoritmos de búsqueda que permitan enfrentarse a estas deficiencias.

5.5.2. Resultados en EHC con h^{diff}

La heurística h^{diff}, al utilizarse en el algoritmo EHC, tiene un comportamiento diferente que en el algoritmo de Escalada. Con el EHC no todos los sucesores de un estado se evalúan si se encuentra antes un nodo con mejor valor heurístico. Si se da esa situación, h^{diff} se comportaría de forma idéntica que h^{FF}, porque también mejoraría el valor heurístico del nodo padre. También se comportarían de forma idéntica cuando los nodos sucesores tengan igual o peor valor heurístico que el padre, y cuando los sucesores con igual evaluación tengan a su vez igual o peor dificultad que la del nodo padre. De hecho, este último caso es el que se produce en el ejemplo analizado para Escalada con h^{diff} mostrado en la Figura 5.6. El estado actual tiene también un plan relajado de longitud cuatro pero la suma de las dificultades es cuatro, por lo que sus dos sucesores, `(PICKUP D)` y `(UNSTACK`

`A B)` tienen ambos peor evaluación para h^{diff}.

El aporte que puede hacer h^{diff} para el EHC es para los casos en los que tanto el nodo padre como alguno de sus sucesores tiene igual valor heurístico para h^{FF}, pero tienen diferente dificultad en las precondiciones de las acciones, lo que conlleva a un mejor valor heurístico para h^{diff}. La diferenciación entre un nodo padre y un sucesor puede permitir al algoritmo EHC evitar la entrada en *plateaus* y por tanto una posible reducción de nodos evaluados. No hay una relación implícita entre un estado con menor dificultad y el posible camino de salida del *plateau*, por lo que en varios casos la elección puede producir que el camino alternativo de salida tenga más pasos. Esto es porque el camino de salida, al ser resuelto con una búsqueda en amplitud, es el mejor camino desde el nodo en que se inicia hasta la salida del *plateau*. En este sentido se puede intuir que contrario a lo que sucedía con Escalada, en EHC, el uso de h^{diff} puede producir perdida de calidad, sin importar si se reducen o no el número de nodos evaluados o el tiempo de planificación.

Se ha evaluado el comportamiento de EHC con h^{diff} comparándolo con el uso de h^{FF}. La Tabla 5.2 muestra el número y porcentaje de problemas resueltos en los dominios de evaluación. h^{diff} ha mejorado el comportamiento de EHC respecto de h^{FF} en seis de los ocho dominios.

Dominio	EHC		EHC con h^{diff}	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	19	47.5 %	**23**	**57.5 %**
Matching-bw	4	10.0 %	**6**	**15.0 %**
Parking	29	72.5 %	**35**	**87.5 %**
Logistics	**40**	**100.0 %**	**40**	**100.0 %**
Mystery'	**26**	**65.0 %**	20	50.0 %
Portcrane	39	97.5 %	**40**	**100.0 %**
Satellite	38	95.0 %	**40**	**100.0 %**
Rovers	33	82.5 %	**37**	**92.5 %**

Tabla 5.2: Problemas resueltos para el experimento de EHC con h^{diff}.

En el *Blocksworld*, la heurística h^{FF} subestima considerablemente el coste real de alcanzar las metas, haciendo que EHC entre constantemente en *plateaus* de mucha profundidad, lo que multiplica exponencialmente el número de nodos evaluados. h^{diff} permite reducir esta explosión en las situaciones explicadas y es la razón por la que tiene un mejor comportamiento en tiempo, que se puede apreciar en la Figura 5.9. En los 20 problemas del *Blocksworld* resueltos por ambas técnicas, h^{FF} obtuvo una longitud acumulada de 31.2 acciones frente a las 34.4 que obtuvo h^{diff}. La calidad se ve afectada porque los desempates de los valores heurísticos de h^{FF} no guardan una relación con la dificultad de resolver los problemas, sino más bien ayudan a salir de los *plateaus*. Al igual que en el *Blocksworld*, en el *Parking* se obtienen por la misma razón mejoras en tiempo en perjuicio de la calidad. Por otro lado, el comportamiento deficiente de EHC en el *Matching Blocksworld* con ambas heurísticas es debido a que en el dominio, muchos estados que mejoran

el valor heurístico son caminos sin salida no reconocibles. Por ejemplo, levantar un bloque con algún brazo, sin importar la polaridad, reduce la longitud del plan relajado, pero como EHC no rectifica sus decisiones después de haber mejorado el valor heurístico, el algoritmo falla en encontrar una solución si en el nodo que mejora el valor heurístico se ha levantado el bloque con el brazo incorrecto.

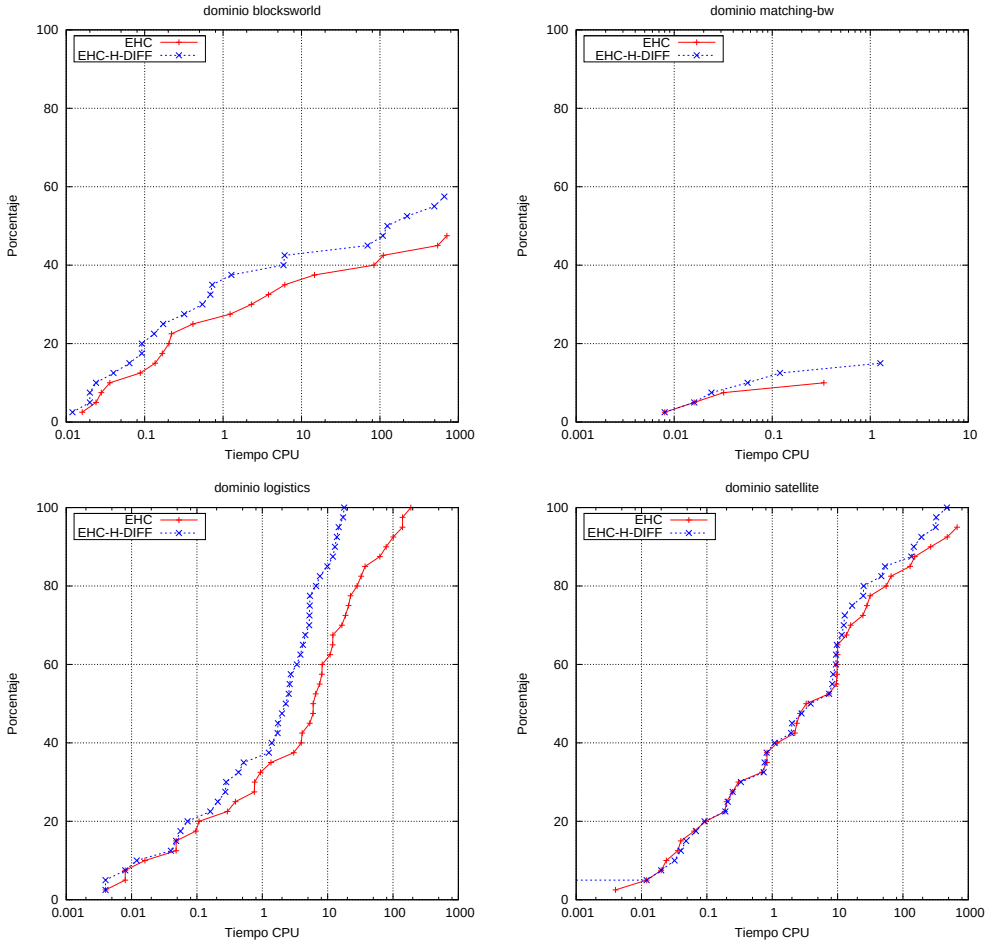


Figura 5.9: Distribución acumulada de tiempo de CPU para experimentos con EHC y h^{diff}.

La mejora en tiempo del *Logistics*, mostrado en la Figura 5.9, se produce manteniendo la calidad de los planes obtenidos con h^{FF}. En este dominio se da la particularidad de que los empates de valores heurísticos que se daban con h^{FF} para el algoritmo de Escalada, coinciden además con un empate en el valor heurístico del nodo padre. EHC con h^{FF} resuelve esta situación explorando los nodos en amplitud hasta profundidad dos (el *plateau* es acotado), pero EHC con h^{diff} puede resolver la situación desde que reconoce un estado con mejor valor heurístico. Esta particularidad se repite en el *Portcrane* que reporta un ligera mejora en tiempo

manteniendo la misma calidad. En el *Mystery'* sin embargo, hay una reducción en el número de problemas resueltos. La característica que se ha comentado en los dominios de transporte se refiere a los medios de transporte que en el plan real tienen que pasar dos veces por un sitio y en el plan relajado no. En el *Mystery'* esta característica no guarda ninguna relación con el consumo de combustible, y es lo que hace que h^{diff} no pueda aprovecharse de la particularidad. Por el contrario, la mayor exploración de nodos en ciertas zonas de la búsqueda permite a EHC con h^{FF} tener más probabilidad de no caer en caminos sin salida sin la posibilidad de recuperarse.

Los resultados en el *Satellite* y en el *Rovers* muestran una reducción de nodos evaluados utilizando h^{diff}, lo que se traduce en una mejora en tiempo y número de problemas resueltos. La calidad de los planes en ambos dominios se ve ligeramente empeorada, principalmente en los problemas grandes. La diferencia debe considerarse poco significativa porque por ejemplo utilizando la puntuación utilizada en la IPC-6 explicada en la sección 4.2, en el dominio *Satellite* h^{diff} consigue 38.51 puntos frente a los 38.00 de h^{FF}. Utilizando este criterio de evaluación la pequeña perdida de calidad se ve compensada por los puntos aportados por los 2 problemas adicionales que se han resuelto. Otras gráficas con la evaluación de tiempo se encuentran en el apéndice C.1.

5.5.3. Resultados en WBFS con h^{diff}

La heurística h^{diff} servirá para deshacer los empates en los nodos de la lista abierta que tengan igual evaluación para la función $f(n) = g(n) + w_h h(n)$. A diferencia de los algoritmos de Escalada y EHC estos empates pueden darse no sólo en la vecindad del nodo evaluado. Específicamente, los empates entre dos nodos se pueden dar en dos situaciones:

1. Tienen igual $h(n)$, en cuyo caso también tienen igual $g(n)$. Aquí, como en los algoritmos previos se prefiere el nodo que tenga menor dificultad en las acciones del plan relajado.

2. Tienen $h(n)$ diferente. El nodo de $h(n)$ menor tiene un plan relajado más corto, y típicamente esto significa menor valor para el $complemento\_h$, y por tanto un desempate en favor del nodo de $g(n)$ más alta y $h(n)$ más pequeña.

Por otro lado, el valor que tenga w_h en la función de evaluación influirá directamente en las oportunidades que tenga el algoritmo de aprovecharse de los desempates que permita h^{diff}. A valores más pequeños de w_h, h^{diff} será menos influyente. Por ejemplo, en la función de evaluación del A*, $f(n) = g(n) + h(n)$, no serviría de nada resolver los empates en los que $f(n)$ sea inferior al coste final del plan. Con una heurística consistente es fácil demostrar que todos esos nodos tendrían que ser expandidos antes de llegar a la solución. Aunque este no es el caso de las heurísticas que se están tratando, la no admisibilidad de las heurísticas no

implica que los desempates utilizando $w_h = 1$ tengan sentido. Se asume que el valor que se ha tomado hasta el momento $w_h = 3$, sí podrá mostrar los efectos de h^{diff}.

La Tabla 5.3 muestra los resultados del experimento que compara h^{diff} con h^{FF} en el algoritmo WBFS. La heurística h^{diff} arroja mejoras en el número de problemas resueltos en siete de los ocho dominios de evaluación. Las mejoras son debido a la reducción de nodos evaluados, que marcan la diferencia en los problemas grandes. No obstante, lo relevante de los resultados en el algoritmo WBFS es su diferencia con los resultados en los algoritmos de búsqueda local, independientemente de la heurística. Por ejemplo, en el *Matching Blocksworld* el mejor resultado de los algoritmos locales fue 13 problemas resueltos, frente a los 25 y 27 de WBFS con h^{FF} y h^{diff} respectivamente. Por el contrario, en el *Rovers*, WBFS apenas resolvió 15 problemas frente a los 33 de Escalada con h^{FF}, el peor resultado de los algoritmos de búsqueda local. La diferencia entre el WBFS y los algoritmos de búsqueda local se ve menos marcada en dominios como el *Blocksworld*, por lo que una conclusión sobre estos resultados no es suficiente para afirmar la influencia que tiene la elección del algoritmo según el tipo de dominio. A priori, se puede intuir que la presencia o no de camino sin salida en los dominios, es una característica que debiera influir más en la elección del algoritmo, que en la propia elección de la heurística.

Dominio	WBFS		WBFS con h^{diff}	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	24	60.0 %	**27**	**67.5 %**
Matching-bw	25	62.5 %	**27**	**67.5 %**
Parking	31	77.5 %	**32**	**80.0 %**
Logistics	19	47.5 %	**20**	**50.0 %**
Mystery'	23	57.5 %	**27**	**67.5 %**
Portcrane	12	30.0 %	**14**	**35.0 %**
Satellite	18	45.0 %	**23**	**57.5 %**
Rovers	**15**	**37.5 %**	13	32.5 %

Tabla 5.3: Problemas resueltos para el experimento de WBFS con h^{diff}.

Por otro lado, los resultados de calidad obtenidos en el experimento de WBFS con h^{diff} no muestran cambios significativos entre una u otra heurística. En la Tabla C.1 del Apéndice C se reportan los promedios de longitud de los planes para los problemas resueltos por ambas técnicas. Mas que diferencias globales, se aprecian variaciones entre unos problemas y otros, en los que cambia la longitud de los planes obtenidos en unas pocas acciones.

5.5.4. Discusión

Los resultados experimentales presentados en la pasadas secciones confirman la carga computacional que suponen para los algoritmos el cálculo de las heurísti-

cas independientes del dominio. En este sentido, la reducción en el número de nodos evaluados, por norma general, siempre se traducirá en un mejor comportamiento de los algoritmos de búsqueda. La heurística h^{diff}, que no supone un mayor coste computacional que h^{FF}, permite la reducción del número de evaluaciones bajo ciertas circunstancias, permitiendo así que globalmente produzca mejor comportamiento que h^{FF}. El estudio más detallado de h^{diff} queda fuera del alcance de esta tesis y quedan abiertas algunas interrogantes que valdría la pena investigar. Se pueden resumir en las siguientes ideas:

- La selección de las acciones a la hora incluirlas en el plan relajado se hace mediante la heurística de preferir la acción de menor dificultad. Habría que determinar qué relación guarda esta preferencia con los valores de h^{diff}.
- Se asume que un mejor comportamiento de una heurística frente a otra en el mismo algoritmo implica que la heurística es más informada. En este sentido, habría que investigar si las mejoras de h^{diff} residen sólo en su información adicional, o más bien en su capacidad de reducir el estancamiento de los algoritmos mediante la resolución de empates de los valores heurísticos. Por ejemplo, cabe la pregunta: ¿Produciría también buenos resultados una resolución aleatoria de estos empates?

Por otro lado cabe señalar, que el análisis detallado de las resoluciones de empates de valores heurísticos, como los mostrados en el ejemplo de la Figura 5.6, revelaron que la disminución de la dificultad se debía a que ciertas acciones se podían aplicar en capas previas del grafo de planificación. En este ejemplo, como en otros estudiados, existe menor dificultad cuando hay un menor número de capas del grafo de planificación. Esta afirmación no puede garantizarse para todos los casos y la implicación inversa tampoco es deducible. Se conoce que el número de capas del grafo de planificación es exactamente igual a la heurística $h^{\max}$ utilizada en el planificador HSP. En este sentido, queda como interrogante adicional determinar si la resolución de empates de valores heurísticos para h^{FF} utilizando $h^{\max}$, produciría los mismos efectos que h^{diff}. Si por el contrario, la suma de las dificultades no guarda una relación directa con el número de capas del grafo de planificación relajado, cabe la posibilidad de utilizar $h^{\max}$ como criterio adicional para deshacer empates porque tampoco supondría para la búsqueda un coste computacional adicional al que tiene h^{FF}.

Capítulo 6

Sistema de Razonamiento Basado en Casos

En este capítulo se presenta el sistema de razonamiento basado en casos que se ha desarrollado sobre el planificador SAYPHI con el objetivo de integrar técnicas de aprendizaje CBR con las técnicas de planificación heurística. Primero se describe cómo están representados los casos en secuencias de tipo y después se presenta de qué forma estas secuencias son utilizadas dentro de un ciclo completo de razonamiento basado en casos. Al final del capítulo se incluye la evaluación experimental que se ha realizado sobre los diferentes algoritmos de búsqueda.

6.1. Secuencias de Tipo

Como se mencionó en la Sección 2.4, la unidad básica de conocimiento para un sistema CBR es el caso. Un caso intenta recoger una experiencia que después será utilizada cuando se necesite resolver un nuevo problema similar al caso almacenado. Los problemas de planificación presentan el inconveniente de que sus soluciones, los planes, son en la práctica infinitos y no es viable almacenar cada experiencia como tal. Por eso, es necesario algún tipo de abstracción que generalice las experiencias y permita recopilar múltiples situaciones en un sólo caso. Una posibilidad es recoger los episodios de planificación de forma fragmentada, haciéndolo de forma particular para cada objeto del problema. Esta idea parte del principio de que en algunos dominios los objetos suelen tener un conjunto de transiciones posibles dentro de un plan. Por ejemplo, en un dominio de transporte se supone que en muchos casos, los objetos habrá que cargarlos a algún medio de transporte, y descargarlos cuando el medio de transporte esté en el lugar de destino del objeto. Si además se considera que estos comportamientos relativos a los objetos se relacionan directamente con el tipo del objeto, se obtiene la idea básica de las secuencias de tipo, que resultan de registrar de forma abstracta el comportamiento relacionado a algún tipo de objeto durante las transiciones generadas por la ejecución de un plan. En los apartados siguientes se describen cómo se repre-

sentan formalmente estas secuencias de tipo y cómo son generadas a partir de un problema de planificación y su plan solución.

6.2. Representación

Para la representación de las secuencias de tipo se van a definir primero una serie de conceptos que son necesarios para su construcción. En el problema de planificación (definido previamente como la tupla (L, A, I, G)), el conjunto de literales L y el conjunto de acciones A se deducen respectivamente del conjunto de predicados $\mathcal{L}$ y del conjunto de operadores $\mathcal{A}$, cuando éstos son instanciados utilizando el conjunto de objetos O declarado para el problema.

Así, un predicado $\ell \in \mathcal{L}$ de aridad n puede instanciar el literal l en la forma $\ell(o_1, \ldots, o_n)$ con $o_i \in O$. Además se definen las funciones

- $pred(l) = \ell$ es el predicado del literal l
- $arg(l)$ = $(o_1, \ldots, o_n)$ son los argumentos del literal l

Igualmente, un operador $op \in \mathcal{A}$, que tiene m parámetros, se puede instanciar en la acción a de la forma $op(o_1, \ldots, o_m)$ con $o_i \in O$. También se definen las funciones

- $op(a) = op$ es el operador de la acción a
- $argop(a)$ = $(o_1, \ldots, o_m)$ son los parámetros de la acción a

Definición 1 Un **sub-estado de un objeto** es el conjunto de literales de un estado en los que el objeto aparece en alguno de sus argumentos. Así, el sub-estado del objeto o para el estado s viene definido por

$$s_o = \{\, l \mid l \in s \ \wedge \ o \in arg(l) \,\} \tag{6.1}$$

Por ejemplo, si en el dominio del *Blocksworld* se tiene un estado $s =$ [`(on A B)` `(ontable B)` `(clear A)` `(holding C)` `(handempty)`], el sub-estado del bloque `A` es [`(on A B)` `(clear A)`] y el sub-estado del bloque `B` es [`(on A B)` `(ontable B)`].

Definición 2 La **propiedad de un objeto** referido a un literal l se define como el predicado de l con un subíndice que representa la posición en la que el objeto aparece como argumento dentro de l. Dado un literal l, la función que calcula la propiedad U para el objeto o se define como

$$U_o(l) = pred(l)_i \ \text{ tal que } \ arg(l) = (o_1, \ldots, o_n) \wedge o = o_i \tag{6.2}$$

Por ejemplo, on_1 es la propiedad del bloque `A` en el literal `(on A B)` y on_2 es la propiedad para el bloque `B`. El concepto de propiedades fue introducido originalmente por (Fox and Long, 1998) en su trabajo de análisis de dominios. Como las propiedades de un objeto representan la posición de éstos en los literales, y cada posición de un predicado está ocupada por un tipo de objeto del dominio, se puede utilizar esta correspondencia para abstraer los objetos y así poder representarlos por sus tipos a través de las propiedades. En esta misma línea, también se pueden utilizar las propiedades de los objeto para abstraer los sub-estados de objeto, convirtiéndolos en sub-estados de tipo. Como dos literales pueden producir la misma propiedad para un objeto se utilizará el concepto matemático de colección. Una colección es una generalización de la definición de conjuntos, en la que sus elementos (no ordenados) pueden estar repetidos.

Definición 3 Un **sub-estado de tipo** es la colección de propiedades de un objeto que abstraen un sub-estado de objeto. La función que calcula el sub-estado de tipo para el objeto o en el estado s se define como

$$\varphi_{s,o} = \{\, U_o(l) \mid l \in s_o \} \tag{6.3}$$

Así, en el ejemplo anterior, el sub-estado del bloque `A` queda transformado en el sub-estado de tipo $(on_1\ clear_1)$. Si en el estado del ejemplo se aplica la acción `(STACK C A)`, el nuevo sub-estado de objeto para el bloque `A` sera [`(on A B)(on C A)`], por lo que su nuevo sub-estado de tipo será $(on_1\ on_2)$. Con la idea de almacenar de forma abstracta la información relativa al objeto, tanto el estado como el objeto que generan el sub-estado de tipo no son relevantes para representar una transición típica en un dominio. Por eso, se denotará con $\mathcal{T}$ a la colección de propiedades que representan un sub-estado de tipo, pero que no está asociado a un estado u objeto en concreto. Por el ejemplo, el sub-estado de tipo $\mathcal{T} = (on_1\ clear_1)$ no representa a un sub-estado de objeto específico, sino que representa todos los sub-estados de objeto en los que un bloque está encima de otro (on) y está libre $(clear)$.

Las transiciones de sub-estados de objeto se producen tras la aplicación de una acción en la que el objeto aparece como parámetro. Por tanto, el operador de esa acción puede asociarse con la transición equivalente de los sub-estados de tipo. De esta forma, se puede representar una transición típica del dominio. Por ejemplo, el sub-estado de tipo $\mathcal{T}_0 = (on_1\ clear_1)$ se transforma en un segundo sub-estado de tipo $\mathcal{T}_1 = (on_1\ on_2)$ después de la aplicación de una acción del operador `STACK`. Al concatenar todas las transiciones de sub-estados de tipo, acompañadas de la acción asociada, se obtiene una representación centrada en el objeto que permite recoger parcialmente un plan y sus estados intermedios. A esta representación se denomina secuencia de tipo.

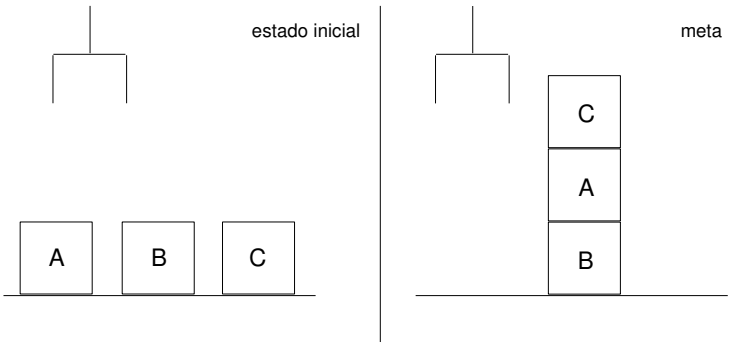
Definición 4 Una **secuencia de tipo** es una secuencia de pares ordenados de la forma

$$\mathcal{Q} = \{(\mathcal{T}_0, \emptyset), (\mathcal{T}_1, a_1), \ldots, (\mathcal{T}_n, a_n)\} \tag{6.4}$$

que se genera para un objeto o a partir de un plan $\mathcal{P} = \{a_1, a_2, \ldots, a_n\}$ y donde $\mathcal{T}_0 = \varphi_{I,o}$ es el sub-estado de tipo del estado inicial y cada $\mathcal{T}_i$ es el sub-estado de tipo obtenido a partir del estado al que se llega al aplicar la acción a_i.

Dado que una secuencia de tipo está centrada en un objeto en particular, no todas las acciones del plan tienen que influir en las transiciones del sub-estado de tipo. Por esta razón se ha definido una acción *no-op*, como una acción vacía que sustituya la acción real en caso de que el sub-estado de tipo permanezca igual tras aplicar una acción en la que el objeto no está como parámetro. De este modo, en el paso ($\mathcal{T}_i$, *no-op*), se asume que $\mathcal{T}_i = \mathcal{T}_{i-1}$ y que la acción a_i no es relevante para la secuencia generada para el objeto en particular. Para evitar concatenaciones sucesivas de pasos con *no-op* cuando hayan varias acciones irrelevantes, se acompañará *no-op* con el número de repeticiones. Semánticamente representarían lo mismo, pero así se mantienen las secuencias lo más compactas posible.

La Figura 6.1 muestra una secuencia de tipo completa para un objeto (bloque `A`) de tipo *Block* en el dominio *Blocksworld*. Se puede apreciar que el sub-estado de tipo correspondiente al estado inicial no tiene acción asociada y que la acción `(PICKUP C)` produce una acción *no-op* porque no tiene relación con el bloque `A`. La construcción de esta secuencia de tipo permite guardar la información del dominio que representa la situación de un bloque que inicialmente está sobre la mesa, y al final del problema termina sobre otro bloque, con un tercer bloque encima. La secuencia indica que para que se produzca esta situación tienen que darse las cuatro transiciones mostradas en el ejemplo.



Plan	Secuencia de Tipo (Bloque A)
Estado inicial	$[(clear_1\ ontable_1),\ nil]$
0: (PICKUP A)	$[(holding_1),\ pickup]$
1: (STACK A B)	$[(clear_1\ on_1),\ stack]$
2: (PICKUP C)	$[(clear_1\ on_1),$ `no-op`$]$
3: (STACK C A)	$[(on_1\ on_2),\ stack]$

Figura 6.1: Ejemplo de una secuencia de tipo en el *blocksworld*.

6.3. Ciclo de Razonamiento

En esta sección se explica cómo las secuencias de tipo pueden ser utilizadas en un ciclo completo de razonamiento basado en casos. Se ha modelado el sistema CBR para planificación heurística como un sistema consejero que acompaña al planificador heurístico según la Figura 6.2. El sistema SAYPHI se muestra como

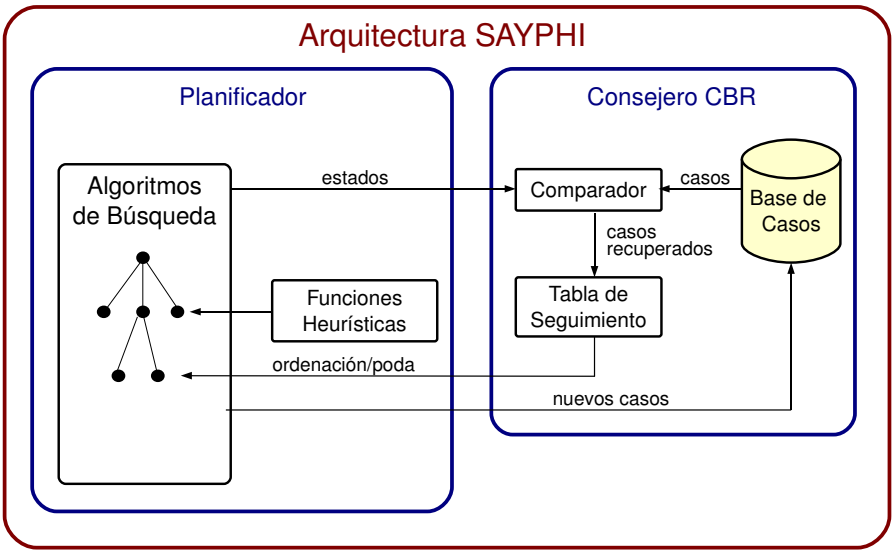


Figura 6.2: SAYPHI como plataforma de planificación y aprendizaje.

una plataforma de planificación en la que otras técnicas de aprendizaje podrían integrarse al planificador heurístico. Para el caso particular de este trabajo, el sistema CBR recibe un problema de planificación y recupera el conjunto de secuencias de tipo más parecidas para la situación de los objetos en el nuevo problema. Estas secuencias son utilizadas como heurística adicional a la heurística del planificador en forma de técnicas de ordenación y poda. En un enfoque de aprendizaje incremental, las soluciones obtenidas pueden almacenarse como nuevos casos en la base de casos. A continuación se explican en detalle los tres procesos del ciclo: retención, recuperación y reutilización.

6.3.1. Retención

El proceso de retención es el que se encarga de la gestión de la base de casos. Sus principales acciones se pueden dividir en

1. **Generación**: Crear las secuencias de tipo a partir de un problema de planificación solucionado.

2. **Almacenamiento**: Guardar las secuencias de tipos según la estructura de la base de casos.

3. **Mantenimiento**: Gestionar las secuencias repetidas, eliminar las secuencias no útiles, etc.

Estas actividades también se pueden identificar como la fase de entrenamiento para el sistema CBR.

Generación

Esta actividad consiste en tomar un problema de planificación, resolverlo con el planificador, y si se encuentra solución, generar una secuencia de tipo para cada objeto que exista en el problema. La generación es independiente del algoritmo con que se resuelva el problema, pero en la práctica es evidente que se obtendrán mejores casos si se obtienen de buenas soluciones. Por eso la mejor elección es resolver los problemas de entrenamiento con el algoritmo DFBnB, explicado en la Sección 5.4. Se busca una primera solución con el algoritmo EHC (o WBFS si el primero falla) y se refinan las soluciones con la técnica de expansión y acotamiento. Para la generación de las secuencias se utiliza la mejor solución encontrada hasta el límite de tiempo. Si el algoritmo termina es porque ha podido explorar el árbol de búsqueda completo.

Dentro de la generación importa además la selección de los problemas de entrenamiento. Para la correcta generación y selección de un conjunto de problemas de entrenamiento es necesario un conocimiento dependiente del dominio. En la comunidad de la PA se suelen utilizar generadores aleatorios de problemas para generar los conjuntos de entrenamiento. Sin embargo, aunque estos generadores intentan crear una distribución correcta en el espacio de problemas, tienen parámetros de entrada y sesgos internos que hacen difícil garantizar que un conjunto de entrenamiento produzca la experiencia necesaria para generar una buena base de casos.

Un estudio detallado de este aspecto escapa a los objetivos de esta tesis, por lo que se asumirá que la generación de problemas pequeños, similares a las primeras instancias de los conjuntos utilizados en las competiciones, serán suficientes para generar las bases de casos. Este supuesto se sustenta en la base de que en la mayoría de dominios, la dificultad de los problemas reside en un aumento en el número de objetos, por lo que problemas pequeños que contengan objetos de los diferentes tipos del dominio, deberían ser suficientes para producir el conocimiento para las secuencias de tipo.

Almacenamiento

Esta actividad se realiza para conservar los casos generados en una estructura física de almacenamiento que pueda ser consultada a la hora de recuperar las secuencias de tipo. El conocimiento generado es dependiente del dominio, por lo que

cada dominio tiene su base de casos independiente. Cada base de casos está formada por un conjunto de ficheros, uno por cada tipo definido en la jerarquía de tipos en el fichero de dominio en PDDL. Dentro de cada fichero de tipo, las secuencias son almacenadas secuencialmente según se van generando. Cada secuencia está acompañada de un conjunto de atributos que sirven como información adicional para los casos. Estos atributos son:

1. El número del caso: Un identificador del caso que indica el orden en que se han almacenado los casos después de su generación.

2. Indicador de si el caso está generalizado: Indica si es un caso único o es una generalización obtenida por el proceso de mezcla en la fase de mantenimiento.

3. Huella de propiedades del plan relajado: La colección de propiedades de los objetos que se adquieren al aplicar el plan relajado obtenido del estado inicial del problema.

4. Factor de utilidad: El ratio de acierto en el proceso de reutilización que se usa para el aprendizaje de valoraciones en el Capítulo 7.

Para el cálculo de la huella de propiedades se utiliza el grafo relajado de planificación y el plan relajado que se obtienen al evaluar el estado inicial con la función heurística. La idea consiste en almacenar información extra que sirva de forma aproximada para reconocer posibles transiciones intermedias que pueden tener los sub-estados de tipo.

Definición 5 La **huella de propiedades** del plan relajado $\mathcal{P}^+ = \{a_1, \ldots, a_n\}$ para un objeto o es la secuencia de sub-estados de tipo $\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ donde cada $\mathcal{T}_i = \varphi_{s^+,o}$ y donde

$$s^+ = \{add(a) \mid a \in \mathcal{P}^+ \wedge a \in A_{i-1}\}$$

con A_i como la capa de acciones i del grafo de planificación relajado.

La huella de propiedades recoge para cada capa del grafo relajado de planificación las propiedades del objeto que se obtendrían al aplicar las acciones del plan relajado que pertenecen a la capa anterior. Retomando el ejemplo de la Figura 6.1, la acción `(PICKUP A)` pertenece a la capa A_0 y las acciones `(STACK A B)` y `(STACK C A)` pertenecen a la capa A_1. Así, la huella del plan relajado del estado inicial para el bloque `A` es $\{(holding_1), (clear_1\ on_1\ on_2)\}$. La acción `(PICKUP C)` no aporta ninguna propiedad para el bloque `A`.

Una secuencia de tipo, más sus atributos correspondientes, es lo que conforma un caso. Por eso no se hace distinción entre casos o secuencias de tipo como la unidad de conocimiento en el sistema.

Mantenimiento

Para el mantenimiento de las bases de casos se ha incluido un proceso de generalización de secuencias, identificando las que sean iguales salvo por la posición o número de acciones *no-op*. La idea consiste en que cuando se va a guardar un nuevo caso, se calcula primero si éste pertenece a la base de casos de ese tipo.

Definición 6 Dos secuencias de tipo son **iguales** si todos los pares ordenados $(\mathcal{T}_i, a_i)$ de las dos secuencias son iguales.

Definición 7 Dos secuencias son **equivalentes** si al eliminar de ambas secuencias todos los pares $(\mathcal{T}_i, a_i)$ tal que $a_i =$ *no-op*, se obtienen dos secuencias iguales.

Si la nueva secuencia es igual a alguna secuencia de la base de casos, se descarta la nueva secuencia generada. Si la nueva secuencia es equivalente a una secuencia de la base de casos, se obtiene una secuencia generalizada mediante un proceso de mezcla de secuencias. Si la secuencia es totalmente nueva se guarda directamente al final de la base de casos. La Figura 6.3 muestra el algoritmo de mezcla de secuencias. El algoritmo permite generar una nueva secuencia generalizada a partir de dos secuencias si éstas son iguales o equivalentes. La secuencia generalizada se almacena en la base de casos y las otras dos son eliminadas. La nueva secuencia generada contiene los mismos pasos, sin incluir aquellos pasos con *no-op*, si en alguna de las dos secuencias no existe esta transición nula.

La idea subyacente a la decisión de generalizar las secuencias de esta forma es que si en alguna secuencia no existe el *no-op*, la transición del objeto puede hacerse directamente y la acción que generó el *no-op* en la otra secuencia no era necesaria o pudo haberse aplicado en otro instante de tiempo. Por ejemplo, si en un dominio genérico de transporte en el que se producen secuencias de acciones como *(cargar, transportar, descargar)*, un objeto que esté como parámetro del *cargar* y del *descargar*, no cambiaría su sub-estado durante la acción *transportar*, pero dicha acción es estrictamente necesaria, por lo que sí debe existir un *no-op* en la secuencia del objeto transportado. Sin embargo, pueden existir otras secuencias en las que las acciones intermedias no sean necesarias y son más bien el resultado de que en el plan quedan entrelazadas acciones para diferentes objetivos pero que no tienen relación entre sí, como por ejemplo transportar otros objetos a otros sitios.

Tras la generación de una secuencia de tipo, es necesario compararla con cada una de las existentes en los casos almacenados hasta el momento. Si la secuencia se generaliza el proceso termina en la comparación de la secuencia que generó la secuencia generalizada. La peor de las situaciones es en la que el caso es totalmente nuevo y tienen que realizarse todas las comparaciones, por lo que la complejidad computacional es $O(n \times m)$, siendo n el número de casos y m la longitud del caso más largo.

Otra actividad enmarcada dentro del mantenimiento de la base de casos es determinar el valor que tiene cada caso, para después preferir los casos más útiles en caso de haber empates en la recuperación. Como incluye todo un proceso de

mezcla-secuencias ($\mathcal{Q}$,$\mathcal{Q}'$): $\mathcal{Q}_m$ secuencia de tipo

---

$\mathcal{Q} = \{(\mathcal{T}_0, \emptyset), \ldots, (\mathcal{T}_n, a_n)\}$: Secuencia de la base de casos
$\mathcal{Q}' = \{(\mathcal{T'}_0, \emptyset), \ldots, (\mathcal{T}'_m, a'_m)\}$: Secuencia nueva a comparar

---

$i = 0$; $j = 0$; $\mathcal{Q}_m = \{\}$
while $\mathcal{T}_i \neq$ **null and** $\mathcal{T'}_j \neq$ **null do**
  if $(\mathcal{T}_i, a_i) = (\mathcal{T'}_j, a'_j)$ **then**
    añadir $(\mathcal{T}_i, a_i)$ a $\mathcal{Q}_m$
    $i = i + 1$; $j = j + 1$
  else if ($a_i =$ *no-op*) **then**
    $i = i + 1$
  else if ($a'_j =$ *no-op*) **then**
    $j = j + 1$
  else
    return null
return $\mathcal{Q}_m$

Figura 6.3: Algoritmo para calcular si dos secuencias de tipo son equivalentes.

cálculo y validación de estas preferencias, se explicará en detalle en el capítulo de análisis de las bases de casos. No se ha contemplado ningún método de olvido para eliminar de la base de casos las secuencias que no sean de utilidad para el sistema.

6.3.2. Recuperación

La recuperación es el proceso en el que se buscan en la base de casos, el caso o los casos más parecidos a una situación que se desea resolver. Para este enfoque se recupera el caso más parecido para cada objeto presente en el nuevo problema, asumiendo que las transiciones realizadas en la situación pasada serán semejantes en la resolución del nuevo problema. Para un buen diseño de un proceso de recuperación es importante descubrir las características que identifican a un episodio aprendido, para utilizarlas como clave de recuperación. Según la representación escogida, para cada objeto del nuevo problema se han identificado las siguientes características.

1. *el tipo del objeto*: Definido en el fichero PDDL del problema

2. *sub-estado de tipo inicial*: Extraído para el objeto a partir del estado inicial del problema

3. *sub-estado de tipo final*: Extraído para el objeto a partir de las metas del problema. Si el objeto no aparece en ningún literal de las metas, este sub-estado de tipo será igual al conjunto vacío.

4. *huella de propiedades del plan relajado*: Extraída a partir del cálculo de la función heurística del estado inicial, tomando las propiedades alcanzadas en cada nivel del grafo de planificación tras aplicar el plan relajado.

La primera característica es directa, porque permite hacer la recuperación sobre el conjunto de casos del tipo específico. Por eso no se considerará en el diseño de la medida de similitud. Por otro lado, la justificación de la segunda y tercera característica queda clara, porque son la abstracción semejante a como se crean las secuencias de tipo. El sub-estado de tipo inicial debería corresponder con el primer paso de una secuencia que se le parezca y el sub-estado tipo final debería contener parcialmente las propiedades del último paso de la secuencia que se esté comparando. La huella de propiedades del plan relajado se puede utilizar como característica adicional de recuperación porque en algunos dominios puede ser importante también las transiciones de los pasos intermedios. Se puede dar la situación que dos secuencias de tipo tengan los mismos sub-estados de tipo inicial y final, pero que se diferencian en sus pasos intermedios debido a que habrán seguido transiciones diferentes según las características de cada problema. No obstante, pruebas experimentales durante el desarrollo de esta tesis han mostrado que la huella de propiedades del plan relajado no representa en la práctica una característica representativa y sólo permite diferenciar correctamente secuencias en algunos casos particulares. Esto se debe a que se producen diferentes huellas sólo en dominios en los que los grafos relajados no sean muy paralelos, de lo contrario las huellas serán similares. Al final se ha decidido mantener una versión simple de recuperación, y relegar a un trabajo futuro el estudio detallado de las ventajas que podría ofrecer la huella de propiedades del plan relajado.

Medida de Similitud

En este enfoque de CBR es importante que el proceso de recuperación sea computacionalmente ligero, porque hay que repetirlo para cada objeto del problema, ya que la intención es recuperar una secuencia por cada objeto. El primer paso antes de calcular las medidas de similitud consiste en crear para cada objeto, un sub-estado de tipo extraído del estado inicial y otro sub-estado de tipo de las metas. Estos sub-estados de tipo se compararán con sus correspondientes sub-estados de tipo de cada secuencia del mismo tipo. Para las comparaciones se establece primero un nivel de correspondencia entre los sub-estados de tipo. Para eso, los sub-estados de tipo $\mathcal{T}$ se expresarán como el par (X, m) donde X es el conjunto de propiedades no repetidas de $\mathcal{T}$ y $m(x)$ es la función que representa el número de veces que cada elemento $x \in X$ aparece en $\mathcal{T}$.

Definición 8 Para dos sub-estados de tipo $\mathcal{T}_1$ y $\mathcal{T}_2$ expresados de la forma (X_1, m_1) y (X_2, m_2) se define un nivel de correspondencia de la siguiente forma:

1. La relación $\mathcal{T}_1 \subseteqq \mathcal{T}_2$ expresa una **correspondencia total** de $\mathcal{T}_1$ con $\mathcal{T}_2$, que se da cuando todas las propiedades de $\mathcal{T}_1$ están presentes en igual cantidad en $\mathcal{T}_2$. Formalmente, esta relación se define como

$$\mathcal{T}_1 \subseteqq \mathcal{T}_2 \leftrightarrow X_1 \subseteq X_2 \wedge \forall x \in X_1 : m_1(x) = m_2(x)$$

2. La relación $\mathcal{T}_1 \subsetneqq \mathcal{T}_2$ expresa una **correspondencia parcial** de $\mathcal{T}_1$ con $\mathcal{T}_2$, que se da cuando todas las propiedades $\mathcal{T}_1$ están presentes en $\mathcal{T}_2$, pero no todas en igual número de repeticiones. Formalmente, esta relación se define como

$$\mathcal{T}_1 \subsetneqq \mathcal{T}_2 \leftrightarrow X_1 \subseteq X_2 \wedge \exists x \in X_1 : m_1(x) \neq m_2(x)$$

3. La relación $\mathcal{T}_1 \nsubseteqq T_2$ expresa que $\mathcal{T}_1$ no tiene correspondencia con $\mathcal{T}_2$ porque no todas las propiedades de $\mathcal{T}_1$ están en $\mathcal{T}_2$. Formalmente esta relación se define como

$$\mathcal{T}_1 \nsubseteqq \mathcal{T}_2 \leftrightarrow X_1 \nsubseteqq X_2$$

En estas comparaciones interesa que las propiedades de los sub-estados de tipo del objeto del problema nuevo estén presentes en los pasos correspondientes de la secuencia almacenada. En este sentido, la comparación se puede entender como la acción de verificar que el sub-estado de tipo (inicial o final) es un sub-conjunto del sub-estado de tipo almacenado. Se hace distinción entre el nivel de correspondencia total y parcial para los casos en los que hay correspondencia entre las propiedades pero en diferentes cantidades. Por ejemplo, $(p_1, q_1) \subseteqq (p_1, q_1, r_1)$, pero sin embargo $(p_1, q_1) \subsetneqq (p_1, q_1, q_1, r_1)$. El sub-estado (inicial o final) del nuevo problema debe ser un sub-conjunto del sub-estado de tipo almacenado y no al revés, porque cuando hay que hacer las comparaciones con el sub-estado de tipo de las metas, éste puede estar descrito parcialmente.

Dado que sólo se está considerando la correspondencia entre los sub-estados de tipo inicial y de las metas, se establece la medida de similitud de un objeto o con respecto a una secuencia $\mathcal{Q} = \{(\mathcal{T}_0, \emptyset), \ldots, (\mathcal{T}_n, a_n)\}$ como una función lineal de ambas características expresada en la fórmula

$$\mathcal{M}_{o,\mathcal{Q}} = w_i \times \mathcal{C}(\varphi_{I,o}\,, \mathcal{T}_0) + w_g \times \mathcal{C}(\varphi_{G,o}\,, \mathcal{T}_n) \tag{6.5}$$

donde la función $\mathcal{C}$ representa el valor numérico del nivel de correspondencia de los dos sub-estados de tipo comparados. Para mantener el esquema sencillo del proceso de recuperación se utilizará la siguiente asignación para el nivel de correspondencia.

$$\mathcal{C}(\mathcal{T}_1, \mathcal{T}_2) = \begin{cases} 2 & \text{si } \mathcal{T}_1 \subseteqq \mathcal{T}_2 \\ 1 & \text{si } \mathcal{T}_1 \subsetneqq \mathcal{T}_2 \\ 0 & \text{si } \mathcal{T}_1 \nsubseteqq \mathcal{T}_2 \end{cases}$$

Comparación y Selección

La comparación de casos, el *ranking* en función de la medida de similitud y la selección de la mejor secuencia se realiza mediante el algoritmo mostrado en la Figura 6.4. Para cada objeto del problema se generan sus sub-estados de tipo correspondientes al estado inicial y final. Después, estos se comparan con cada una de las secuencias en el fichero de casos del tipo correspondiente, asignándoles un valor en función de la medida de similitud.

seleccion-de-secuencias ($\mathcal{B}$,O,I,G): $\mathcal{RQ}$ tabla de seguimiento

---

$\mathcal{B}$: Base de casos del dominio.
O: Conjunto de objetos del problema.
I,G: Estado inicial y final del problema.

$\mathcal{RQ}$ =[]
for each o en O **do**
  for each $\mathcal{Q} = \{\mathcal{T}_0, \ldots, \mathcal{T}_n\}$ del tipo de o en $\mathcal{B}$ **do**
    $\mathcal{M}_{o,\mathcal{Q}} = 2 \times \mathcal{C}(\varphi_{I,o}\ , \mathcal{T}_0) + \mathcal{C}(\varphi_{G,o}\ , \mathcal{T}_n)$
  $\mathcal{Q}_{seleccionado} \leftarrow \arg\max_{\mathcal{Q}_i \in \mathcal{B}} \mathcal{M}(o, \mathcal{Q}_i)$
  insertar $\mathcal{Q}_{seleccionado}$ en $\mathcal{RQ}$
return $\mathcal{RQ}$

Figura 6.4: Algoritmo para el ranking y selección de las secuencias de tipo.

Al final, por cada objeto se elige la secuencia con mayor medida de similitud. En la fórmula 6.5 se utilizan los valores de de $w_i = 2$ y $w_g = 1$ para darle más peso a la comparación del estado inicial. Esta elección se debe a que objetos en el problema no están presentes como argumentos del estado final (las metas son una descripción parcial de un estado meta), por lo que las correspondencias encontradas con el estado inicial deben ser más relevantes. Si no hay al menos correspondencia parcial en alguno de los sub-estados de tipo para ninguna de las secuencias en la base de casos, se considera que no hay una secuencia relevante para ese objeto y no se le asigna una secuencia recuperada. Para los casos en que haya empate en el valor de similitud de dos o más secuencias, se elige la secuencia de menor longitud. Esta selección es arbitraria y no está sustentada de forma genérica en todos los dominios. En algunos dominios esto garantiza que si las secuencias han sido generadas de transiciones similares se elija la de mejor calidad en término de número de transiciones. En capítulos posteriores se mostrará que una resolución más inteligente de estos empates permite mejorar el proceso de recuperación. Todas las secuencias recuperadas se almacenan en una tabla de seguimiento que después

será utilizada en la fase de reutilización.

6.3.3. Reutilización

En este proceso se utilizan las secuencias recuperadas como conocimiento dependiente del dominio que ayuda en la resolución del nuevo problema. Las secuencias recuperadas son gestionadas a través de la tabla de seguimiento, que para cada objeto del problema tiene asociado la secuencia recuperada y el número del paso actual de la secuencia (0 inicialmente). La idea consiste en que cuando un estado seleccionado de la búsqueda reproduzca la misma transición para el sub-estado del objeto, se considera que el objeto está siguiendo la secuencia y ésta se avanza al siguiente paso.

Para que el seguimiento de las secuencias y la correspondencia de las transiciones de sub-estados de tipo sean independientes del algoritmo de búsqueda, se utilizarán los conceptos de nodo recomendado y ratio de recomendación para reconocer los nodos sucesores que están reproduciendo alguna de las secuencias recuperadas.

La tabla de seguimiento se denominará con el símbolo $\mathcal{RQ}$ y la función *paso_siguiente*$(\mathcal{RQ}, o) = (\mathcal{T}_i, a_i)$ es la función que devuelve el paso siguiente que debe reproducir la secuencia recuperada para el objeto o. Dado un estado s, se consideran como nodos sucesores a los pares (s', a) siendo $s' = aplicar(a, s)$. Para saber si la secuencia recuperada para el objeto o se reproduce en un nodo sucesor (s', a) se define la función.

$$\mathcal{W}(\mathcal{RQ}, o, s', a) = \begin{cases} 1 & \text{Si} \quad paso\_siguiente(\mathcal{RQ}, o) = (\mathcal{T}_i, a_i) \wedge \\ & \quad o \in argop(a) \wedge op(a) = op(a_i) \wedge \\ & \quad \mathcal{C}(\varphi_{s',o}, \mathcal{T}_i) > 0 \\ 0 & \quad \text{de cualquier otro modo} \end{cases}$$

Definición 9 Un **nodo recomendado** es un nodo sucesor que reproduce la transición de la secuencia recuperada para algún objeto que esté como argumento de la acción del nodo. Formalmente, el sucesor (s', a) es un nodo recomendado si

$$\exists o \in argop(a) \mid \mathcal{W}(\mathcal{RQ}, o, s', a) = 1$$

Definición 10 El **ratio de recomendación** para un nodo sucesor es el factor que indica el nivel de recomendación en relación al total de objetos presentes como argumentos de la acción aplicada. Formalmente, el ratio de recomendación para un nodo sucesor (s', a) se define con la función

$$r(\mathcal{RQ}, s', a) = \frac{\sum_{\forall o \in argop(a)} \mathcal{W}(\mathcal{RQ}, o, s', a)}{|argop(a)|}$$

Saber si un nodo es recomendado sirve para definir una función de clasificación binaria que expresa si el nodo reproduce alguna transición en cualquiera de los argumentos de su acción aplicada. En cambio, el ratio de recomendación permite diferenciar entre varios nodos que sean recomendados y así elegir el que, en proporción al número de argumentos, produce más transiciones correctas. El ratio de recomendación produce una heurística complementaria que aporta más información que simplemente determinar si un nodo es recomendado, pero por otro lado el nodo recomendado es computacionalmente más barato de calcular ya que no tiene la necesidad de comparar todos los sub-estados de tipo de sus objetos si ya se ha determinado que alguno cumple con la transición correcta.

La heurística dependiente del dominio aportada por los nodos recomendados puede ser utilizada como conocimiento de control para la búsqueda y puede ser aplicada de diferentes formas. De forma genérica se pueden clasificar en esquemas de ordenación de nodos o esquemas de poda, explicadas en los siguientes apartados.

Esquema de Ordenación

Es un esquema que se integra dentro de una búsqueda guiada por una función heurística (h^{FF} por ejemplo), pero la recomendación de los nodos da un criterio de preferencia ante la selección de nodos en las siguientes circunstancias:

- **Ordenación de evaluaciones**: Utilizada para los algoritmos de búsqueda avariciosos en los que se diferencia explícitamente la generación de los sucesores frente a su evaluación con la función heurística. En este caso la ordenación se utiliza para indicar qué nodos evaluar primero. Si la recomendación sea correcta, la técnica permite ahorrar llamadas a la función heurística al evitar evaluaciones innecesarias.

- **Ordenación de empates**: Utilizada como heurística complementaria de la función heurística utilizada en la búsqueda. Para aquellas situaciones en las que el valor heurístico de los sucesores resulte igual, se realiza una selección desempatando en función del valor del ratio de recomendación.

Esquema de Poda

Es un esquema en el que el proceso de búsqueda utiliza de forma explícita la heurística aportada por las recomendaciones CBR, eliminando del árbol de búsqueda ciertos nodos bajo las siguientes circunstancias:

1. **Selección directa**: Utilizado para algoritmos de búsqueda local en los que se descarta el resto de sucesores después de la selección de un nodo recomendado o el nodo con mejor ratio de recomendación.

2. **Poda exclusiva**: Se utiliza para reducir el tamaño del árbol de búsqueda, dejando sólo como candidatos (lista abierta) a aquellos nodos que han sido

recomendados. Los nodos descartados pueden eliminarse directamente, o incluirse en una lista retrasada para garantizar la completitud de los algoritmos tipo mejor primero.

6.4. Algoritmos Heurísticos + CBR

En esta sección se muestra cómo se han modificado los algoritmos heurísticos utilizados tradicionalmente para planificación, de modo que incluyan técnicas de control para la búsqueda aportadas por las secuencias de tipo, utilizando los esquemas de ordenación o poda explicadas en el apartado anterior.

6.4.1. Escalada+CBR

Este algoritmo modifica el algoritmo tradicional de Escalada (*hill-climbing*), al que se le incluye una técnica de control bajo un esquema de poda de selección directa. El algoritmo se comporta como sigue. En cada nivel de la búsqueda se generan los sucesores y se elige el mejor entre ellos basándose en la función r. De este modo, se selecciona el estado que reproduce el mayor número de transiciones en las secuencias de la tabla de seguimiento. Si entre los sucesores no hay un nodo recomendado, se evalúan todos los sucesores con la función heurística utilizada por el algoritmo y se elige el nodo con mejor valor heurístico. La Figura 6.5 muestra el pseudocódigo para este algoritmo.

El nodo seleccionado por recomendación CBR es evaluado con la función heurística para poder generar las acciones útiles del siguiente nivel. La técnica de las acciones útiles se sigue manteniendo en este algoritmo. Un primera aproximación de este algoritmo (De la Rosa et al., 2006) mostró que una aplicación directa de la recomendación CBR, sin la técnica de las acciones útiles, valía sólo para problemas relativamente pequeños.

La Figura 6.6 muestra un ejemplo de seguimiento de secuencias en el Escalada+CBR, utilizado para resolver un problema del *Blocksworld*. En representación gráfica están el estado inicial y final del problema. En el centro está el plan que encuentra una solución y a los lados las cuatro secuencias recuperadas, una por cada objeto de tipo `Block`. La reproducción del seguimiento sobre el árbol de búsqueda que sólo considera las acciones *helpful* sería de la siguiente manera:

1. Los únicos sucesores del estado inicial, `(UNSTACK A B)` y `(UNSTACK C D)` tienen ambos ratio 1, porque sus dos argumentos corresponden con la transición de la secuencia recuperada para cada uno. Se elige `(UNSTACK A B)` por el orden establecido en las acciones. Ambas son correctas para encontrar el plan óptimo.

2. La acción `(PUTDOWN A)` corresponde con el tercer paso de la secuencia del bloque `A` (tiene ratio 1). La acción `(STACK A C)` tiene ratio 0, porque no cumple con las transiciones de `A` ni de `C`. Esta acción tendría correspondencia

Escalada+CBR $(I, G, \mathcal{RQ})$: *plan*

I, G: Estado inicial y metas del problema
$\mathcal{RQ}$: Tabla de seguimiento de secuencias

$s \leftarrow I$; $solucion =$ **false**
while $(s \neq$ **null and not** $solucion)$ **do**
  if $G \subseteq s$ **then**
    $solucion =$ **true**
  else
    $S' \leftarrow$ *generar-sucesores*(s)
    $s_r \leftarrow \arg\max_{s' \in S'} r(\mathcal{RQ}, s', a)$ /**a aplicada en s **/
    if $r(\mathcal{RQ}, s', a) > 0$ **then**
      $s \leftarrow s_r$; evaluar $h(s)$
    else
      evaluar $h(s'), \forall s' \in S'$
      $s \leftarrow \arg\min_{s' \in S'} h(s')$
if $solucion$ **then**
  return *camino* (I, s)
else
  return *fallo*

Figura 6.5: Algoritmo de Escalada guiado por recomendaciones CBR.

si la transición en la secuencia del bloque `C` tuviera el sub-estado de tipo $(on_1\ on_2)$.

3. Las acciones `(PICKUP B)` y `(UNSTACK C D)` tienen ratio 1. Se prefiere la segunda porque en ese caso se siguen dos secuencias a la vez [1], las correspondientes a `C` y a `D`.

4. La acción elegida `(PUTDOWN C)` tiene ratio 1. `(STACK B C)` tiene ratio 0 y `(STACK C A)` tiene ratio ½ porque `C` no concuerda, pero la transición actual de `A` sí, ya que espera un bloque encima en la transición $\{(ontable_1\ on_2)$, `STACK`$\}$.

5. Las acciones `(PICKUP B)` y `(PICKUP D)` tienen ambas ratio 1. Aquí se aprecia claramente cómo la poda de las acciones *helpful* deja un conjunto

[1] La resolución de empates en la función r se considera arbitraria dado que la preferencia de seguir más secuencias a la vez no implica de por sí un orden correcto en la selección de las acciones.

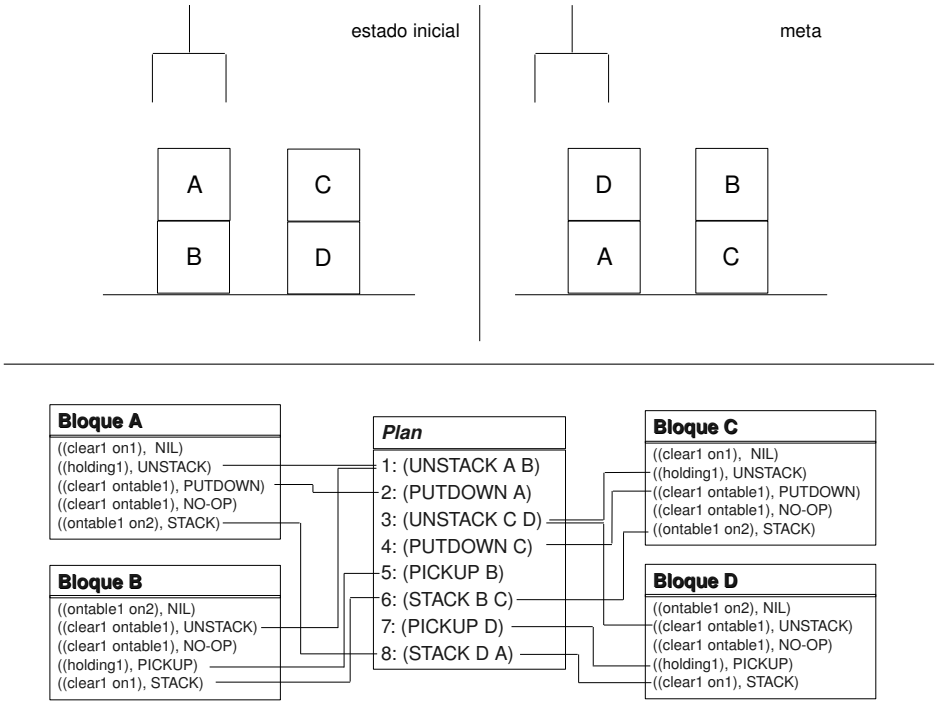


Figura 6.6: Ejemplo en el seguimiento de las secuencias en el algoritmo Escalada+CBR.

reducido y coherente de sucesores, al no considerar levantar los bloques `A` y `C`. Del conjunto de sucesores, los dos son correctos, y se elige por el orden establecido la acción `(PICKUP B)`.

6. La acción `(STACK B D)` tiene ratio ½, porque `B` concuerda, pero `D` no. Las acciones `(STACK B D)` y `(STACK B A)` tienen ratio 1. La selección es la correcta por casualidad, dado el orden de las acciones, por lo que es una situación en la que podría darse una equivocación. La acción `(STACK B A)` se marca como útil porque consigue la sub-meta del brazo libre.

7. La acción `(PICKUP D)` con ratio 1 es la única que se considera como nodo sucesor.

8. La acción `(STACK D C)` con ratio 1 es el único sucesor y con éste se consiguen las metas del problema.

En el ejemplo, aunque las secuencias se han seguido en todos sus pasos, hay situaciones en las que la decisión no implica sólo a la recomendación CBR. En los pasos 1, 5, 7 y 8 fue suficiente la poda de las acciones útiles. La función r fue

determinante en los pasos 2 y 4, porque permitió seleccionar la acción correcta. Los pasos 3 y 6 indican que hay situaciones en las que la recomendación no permite decidir. No obstante, si la decisión arbitraria es la correcta, esto permite avanzar las transiciones de las secuencias.

El Escalada+CBR permite el ahorro de tiempo de planificación por la reducción de evaluación de nodos con la función heurística. Si en un nivel un nodo es seleccionado, el resto de hermanos no es evaluado. En el mejor de los casos si un plan tiene n pasos, se evalúan $n + 1$ nodos, el nodo inicial más todos los nodos que pertenecen a la solución. El peor de los casos se comportaría peor que el Escalada porque además del tiempo de evaluación de todos los nodos, habría que añadir el tiempo de comparaciones infructuosas con las secuencias recuperadas. Esta situación es diferente a intentar aplicar el algoritmo con una base de casos vacía, en cuyo caso sí se produciría un comportamiento equivalente al del algoritmo de Escalada.

6.4.2. EHC+CBR

Este algoritmo modifica el algoritmo EHC (Hoffmann and Nebel, 2001) para aprovechar su característica avariciosa en términos de evaluación de nodos. Aquí se implementa un esquema de ordenación para las evaluaciones, de modo que al evaluar primero nodos que son más prometedores, se intenta ahorrar evaluaciones de la función heurística.

El algoritmo se comporta de la siguiente forma: A partir del nodo inicial se sigue realizando una búsqueda en amplitud de forma iterativa para encontrar nodos que mejoren el valor heurístico. Para cada nodo, después de la generación de sus sucesores se determina si el sucesor a evaluar es recomendado, en cuyo caso se evalúa, de lo contrario su evaluación se pospone. Después se determina si el segundo sucesor es recomendado y así sucesivamente. Si entre los nodos recomendados que ya han sido evaluados no hay un nodo que mejora el valor heurístico, se evalúan el resto de nodos y se continúa con el algoritmo de forma normal. La Figura 6.7 muestra el pseudocódigo para este algoritmo.

La modificación al EHC resulta interesante porque mantiene el mismo esquema del algoritmo, dando la oportunidad de ahorrar llamadas a la función heurística. Su principal inconveniente es que un buen conocimiento aprendido no implica siempre un mejor rendimiento porque la selección de los nodos en la búsqueda depende exclusivamente de la función heurística. A diferencia de Escalada+CBR donde había selección directa por recomendación, en este caso sólo hay evaluación, dando como resultado que si un nodo recomendado (que podría pertenecer a un plan solución) no tiene buena estimación heurística, no será seleccionado para continuar la búsqueda.

EHC+CBR $(I, G, \mathcal{RQ})$: *plan*

I: Estado inicial
G: Metas
$\mathcal{RQ}$: Tabla de seguimiento de secuencias

$s \leftarrow I$; $abierta = \emptyset$
$mejor\_h = h(I)$; $solucion =$ **false**
while $s \neq$ **null and not** $solucion$ **do**
  $S' \leftarrow$ *generar-sucesores*(s)
  while $S' \neq \emptyset$ **do**
    $s' \leftarrow$ sacar primer nodo de S'
    if *recomendado*$(s', \mathcal{RQ})$ **then**
      if $h(s') < mejor\_h$ **then**
        $S' \leftarrow \emptyset$; $S'' \leftarrow \emptyset$; $abierta \leftarrow \emptyset$
        $mejor\_h = h(s')$
      poner s' al final de $abierta$
    else
      poner s' al final de S''
  while $S'' \neq \emptyset$ **do**
    $s'' \leftarrow$ sacar primer nodo de S''
    if $h(s'') < mejor\_h$ **then**
      $S'' \leftarrow \emptyset$; $abierta \leftarrow \emptyset$
      $mejor\_h = h(s'')$
    poner s'' al final de $abierta$
  $s \leftarrow$ sacar primer nodo de $abierta$
  if $G \subseteq s$ **then**
    $solucion =$ **true**
if $solucion$ **then**
  return *camino*(I, s)
else
  return *fallo*

Figura 6.7: Algoritmo EHC guiado por recomendaciones CBR.

6.4.3. WBFS+CBR

Esta es la integración de las recomendaciones de las secuencias con la familia de algoritmos de mejor primero, como por ejemplo el BFS o el WBFS. Para estos algoritmos se han planteado dos enfoques. Primero utilizando una estrategia de poda exclusiva explicada anteriormente y segundo utilizando una estrategia de ordenación para deshacer los empates de los valores heurísticos.

El algoritmo de mejor primero con poda exclusiva se comporta de la siguiente forma. Después de la generación de los sucesores, se introducen en la lista abierta sólo los nodos recomendados y ésta se ordena con la función de evaluación utilizada en el algoritmo. Los nodos no recomendados pueden descartarse, o almacenarse en una lista retrasada en caso de que se quiera garantizar la completitud del algoritmo. Las primeras pruebas experimentales con este enfoque mostraron que la poda por recomendación podía ser perjudicial en algunas situaciones porque comprometía la calidad de las soluciones en muchos problemas: la recomendación errónea puede eliminar el camino correcto a la solución. Debido a esto, no se considerará como algoritmo para las pruebas finales presentadas en esta tesis.

El segundo enfoque utiliza una estrategia de ordenación para los empates de la función de evaluación en el WBFS. Para mantener el mismo esquema del algoritmo, esta idea se implementa modificando la propia función de evaluación. Una alternativa para esta modificación es tomar la función de evaluación:

$$f(s) = g(s) + w_h \times (h(s) + 1 - h^{cbr}(s)) \tag{6.6}$$

donde $h^{cbr}(s)$ es la función que calcula el ratio de recomendación del estado s al igual que en los algoritmos anteriores. La idea consiste en modificar el valor que aporta la función heurística $h(s)$ a la función de evaluación $f(s)$, de modo que nodos recomendados tengan preferencia para seguir la búsqueda. Este enfoque no es válido para la función de evaluación del A*, $f(s) = g(s) + h(s)$. Suponiendo que la función heurística utilizada fuera admisible, si s_m es un estado meta, tendrían que evaluarse todos los nodos generados que cumplan que $f(s) < g(s_m)$, por lo que no tiene sentido establecer una preferencia de evaluación para ese conjunto de nodos. Aunque las funciones utilizadas en los dominios de planificación no son admisibles, pruebas experimentales han mostrado que el efecto es el mismo, por lo que la preferencia de nodos recomendados tampoco produce cambios significativos en la búsqueda, aunque la heurística sea no admisible. Por el contrario, en el WBFS no se reconsideran cierto grupo de nodos después que el árbol de búsqueda tiene cierta profundidad, por lo que la preferencia de unos nodos sobre otros sí tiene sentido en estos casos. En este sentido, valores más altos de w_h serán más influyentes para mostrar el efecto de las recomendaciones al igual que ya lo son para avanzar más rápidamente la búsqueda hacia una solución.

En ambos enfoques CBR aplicados a los algoritmos de mejor primero hay que considerar que las secuencias pueden estar recomendando diferentes caminos a la vez a diferencia del Escalada+CBR y EHC+CBR que hacen búsqueda local. Por eso es necesario que cada nodo de la búsqueda tenga asociado una referencia a la

posición actual de las secuencias en la tabla de seguimiento. Así, cada nodo sucesor utilizará y actualizará las referencias a las secuencias de su nodo padre.

6.4.4. Técnica Lookahead Basada en Casos

La construcción de nodos *lookahead* explicada en la Sección 5.4 implica la ordenación del conjunto de acciones aplicables que pertenecen al plan relajado. En la construcción por omisión, la ordenación escogida es la definida por el propio plan relajado, lo que plantea la posibilidad de que la ordenación más apropiada puede decidirse mediante recomendaciones de las secuencias de tipo, de manera similar a cómo las evaluaciones son ordenadas en el algoritmo EHC+CBR.

La idea consiste en modificar el algoritmo de construcción de los nodos *lookahead*, de modo que se elija del conjunto de acciones aplicables del plan relajado la acción y el estado asociado con el mejor ratio de recomendación. Después, se recalculan las acciones aplicables, se vuelve a elegir la mejor acción, y así sucesivamente. El proceso incluye un control de estados repetidos, de forma que si en algún momento se encuentra un estado visitado en el proceso de búsqueda, se descarta la acción y se elige la segunda mejor. Para la condición de parada se pueden escoger diferentes estrategias:

- No añadir más acciones al nodo *lookahead* si no hay más nodos recomendados
- Aplicar las acciones sucesivamente con o sin recomendación hasta que no se puedan aplicar más acciones del plan relajado.

No se podría justificar una elección hasta verificar la mejor opción con resultados empíricos, pero se ha decidido adoptar el segundo enfoque dado que la técnica *lookahead* es sin ordenación bastante buena en muchos dominios, por lo que parar la construcción por no haber recomendación no parecería lo más eficiente ante el objetivo de mejorar el rendimiento. Estudios alternativos podrían analizar si se mejora la calidad de las soluciones utilizando el segundo enfoque. La Figura 6.8 muestra el pseudo-código de la función que construye un nodo *lookahead* utilizando las recomendaciones de las secuencias de tipo. Esta función es sustituida por la creación estándar de nodos *lookahead* en los algoritmos que utilizan esta técnica, dando lugar a nuevos algoritmos que integran las recomendaciones CBR para ayudar al proceso de búsqueda. Estos algoritmos son:

- **EHC Lookahead+CBR**: Que utiliza la función *Lookahead+CBR* para la creación de los nodos *lookahead* dentro del algoritmo EHC+Lookahead.
- **WBFS Lookahead+CBR**: Que utiliza la función *Lookahead+CBR* para la creación de los nodos *lookahead* dentro del algoritmo WBFS+Lookahead.

Función Lookahead+CBR $(s, RP, \mathcal{RQ})$: *nodo-lookahead*

---

s: Estado actual
RP: Plan relajado del estado s
$\mathcal{RQ}$: Tabla de seguimiento de secuencias

$la\_s \leftarrow s$; $la\_plan = \emptyset$
$A \leftarrow$ acciones aplicables de RP
while $A \neq \emptyset$ **do**
  $LA \leftarrow \{la \mid la = \textit{aplicar}(a, la\_s), \forall a \in A\}$
  $la\_s \leftarrow \arg\max_{la \in LA} r(\mathcal{RQ}, la, a)$
  $a' \leftarrow$ acción aplicada en la_s
  sacar a' de RP y añadirla a la_plan,
  $A \leftarrow$ acciones aplicables de RP
return *crear-nodo-lookahead*(la_s,la_plan)

Figura 6.8: Función para generar un nodo *lookahead* con recomendación CBR.

6.5. Experimentación y Resultados

En esta sección se presentan los experimentos realizados para la evaluación de los diferentes algoritmos CBR. Además de las características descritas en la Sección 4.3, se explican las características de generación de la base de casos para cada dominio.

6.5.1. Generación de las Bases de Casos

La primera fase en la experimentación que evaluará los algoritmos guiados por CBR consiste en la generación de la base de casos para cada dominio de evaluación. Cada base de casos se generará a partir de un conjunto de veinte problemas de entrenamiento generados con dificultad incremental con los generadores aleatorios de problemas. El conjunto de entrenamiento es de una dificultad menor que el conjunto de evaluación, para garantizar que todos sus problemas puedan resolverse con el algoritmo DFBnB en poco tiempo, y que a la vez las soluciones obtenidas sean de buena calidad. La Tabla 6.1 muestra las características de los problemas de entrenamiento para cada dominio de evaluación. La última columna es el total de secuencias de tipo generadas para cada dominio. En el Capítulo 7 se hará un análisis más detallado de la generación de las secuencias.

Los problemas de entrenamiento fueron resueltos con el algoritmo DFBnB utilizando un límite de tiempo de 60 segundos. Para los casos en que un problema

Dominio	Objetos		Literales		Metas		Total
	min.	max.	min.	max.	min.	max.	Casos
Blocksworld	4	7	6	12	1	5	26
Matching-bw	7	10	20	32	1	6	119
Parking	7	13	9	18	4	8	79
Logistics	12	28	10	24	3	10	133
Mystery'	18	39	29	60	5	15	187
Portcrane	7	10	16	30	2	4	93
Satellite	13	32	8	43	4	13	94
Rovers	13	28	34	148	2	10	202

Tabla 6.1: Características de los problemas de entrenamiento.

agotó todo el tiempo, se utilizó la mejor solución encontrada hasta ese instante. Para los casos en que el algoritmo termina en un tiempo menor, se asume que se ha explorado el árbol completo de búsqueda. Aunque en la práctica la mayoría de estas soluciones son óptimas, no se puede garantizar que se dé para todos los casos porque la poda del algoritmo se realiza con la función $f(s) = g(s) + h(s)$, donde $h(s)$ es la heurística h^{FF}, que es una heurística no admisible.

En los siguientes apartados se presentan los resultados de los experimentos realizados para evaluar los algoritmos guiados por las recomendaciones CBR. Para cada algoritmo se presentará la evaluación de:

- **Algoritmo Base**: Corresponde al algoritmo original utilizando la heurística h^{FF}.
- **Algoritmo + CBR**: Corresponde al algoritmo modificado que utiliza alguna de las estrategias para integrar las recomendaciones de las secuencias de tipo. También se utiliza la heurística h^{FF}.

En todos los experimentos se han utilizado las características expuestas en la Sección 4.3, por lo que son además comparables con los resultados de la heurística h^{diff}.

6.5.2. Resultados en Escalada+CBR

En estos experimentos se evalúa el comportamiento del algoritmo de Escalada guiado por recomendaciones CBR. La Tabla 6.2 muestra el número y porcentaje de problemas resueltos para los distintos dominios de evaluación. Se puede observar que con Escalada+CBR hay una mejora en problemas resueltos en cinco de los ocho dominios, y en un sólo caso el comportamiento se ve perjudicado. La Figura 6.9 muestra la distribución acumulada de tiempo de cuatro dominios. El resto se pueden apreciar en la Figura C.3 en los resultados adicionales recopilados en el Apéndice C.

En el *Blocksworld* se produce una mejora en tiempo y en calidad de los planes encontrados. La mejora en tiempo es debido a la reducción del número de evaluaciones producidas en los pasos en los que se seleccionaba directamente el nodo

Dominio	Escalada		Escalada + CBR	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	34	85.0 %	**37**	**92.5 %**
Matching-bw	**11**	**27.5 %**	**11**	**27.5 %**
Parking	38	95.0 %	**39**	**97.5 %**
Logistics	**34**	**85.0 %**	**34**	**85.0 %**
Mystery'	11	27.5 %	**15**	**37.5 %**
Portcrane	37	92.5 %	**38**	**95.0 %**
Satellite	**38**	**95.0 %**	36	90.0 %
Rovers	38	95.0 %	**39**	**97.5 %**

Tabla 6.2: Problemas resueltos para el experimento de Escalada+CBR.

Dominio	Escalada	Escalada+CBR	% mejora CBR
Blocksworld (33)	891.8	**459.9**	**48.4 %**
Matching-bw (10)	42.8	**35.5**	**17.1 %**
Parking (38)	**66.4**	76.7	-13.4 %
Logistics (34)	977.2	**725.1**	**25.8 %**
Mystery' (9)	**16.4**	18.0	-8.8 %
Portcrane (36)	780.8	**423.3**	**45.8 %**
Satellite (35)	**67.1**	69.3	-3.2 %
Rovers (38)	152.0	**112.0**	**26.3 %**

Tabla 6.3: Promedio de longitud de los planes para problemas resueltos en las configuraciones del experimento de Escalada+CBR. Entre paréntesis el número de problemas resueltos en ambas configuraciones.

indicado por la recomendación de las secuencias. La mejora en calidad también ayudó a la reducción del número de evaluaciones. De los 33 problemas resueltos por ambas técnicas, Escalada obtuvo un promedio de 891.8 acciones por plan, mientras que Escalada+CBR obtuvo un promedio de 459.9 acciones. La mejora en calidad es relativa, porque en ambos casos las soluciones son de baja calidad, comparadas por ejemplo con las soluciones que obtiene EHC. Como la heurística h^{FF} no aporta información en muchas regiones de la búsqueda, la selección de un sucesor se produce de forma ciega, permitiendo que en esas situaciones una buena indicación de las secuencias recuperadas por CBR produzca un mejor camino. Las mejoras en calidad no se producen de forma sistemática dado que Escalada+CBR logra obtener mejores planes en 26 problemas pero sin embargo en 12 problemas Escalada resulta ser mejor. La Tabla 6.3 muestra el promedio de la longitud de los planes para los problemas resueltos por ambas configuraciones en los ocho dominios de evaluación. En la tercera columna se muestra el porcentaje de mejora que ha obtenido Escalada+CBR.

A pesar de que en el *Matching Blocksworld* se obtiene una pequeña reducción en el número de evaluaciones, el resultado relevante es que el algoritmo de Escalada es inapropiado para dominios con caminos sin salida, como ya se expuso

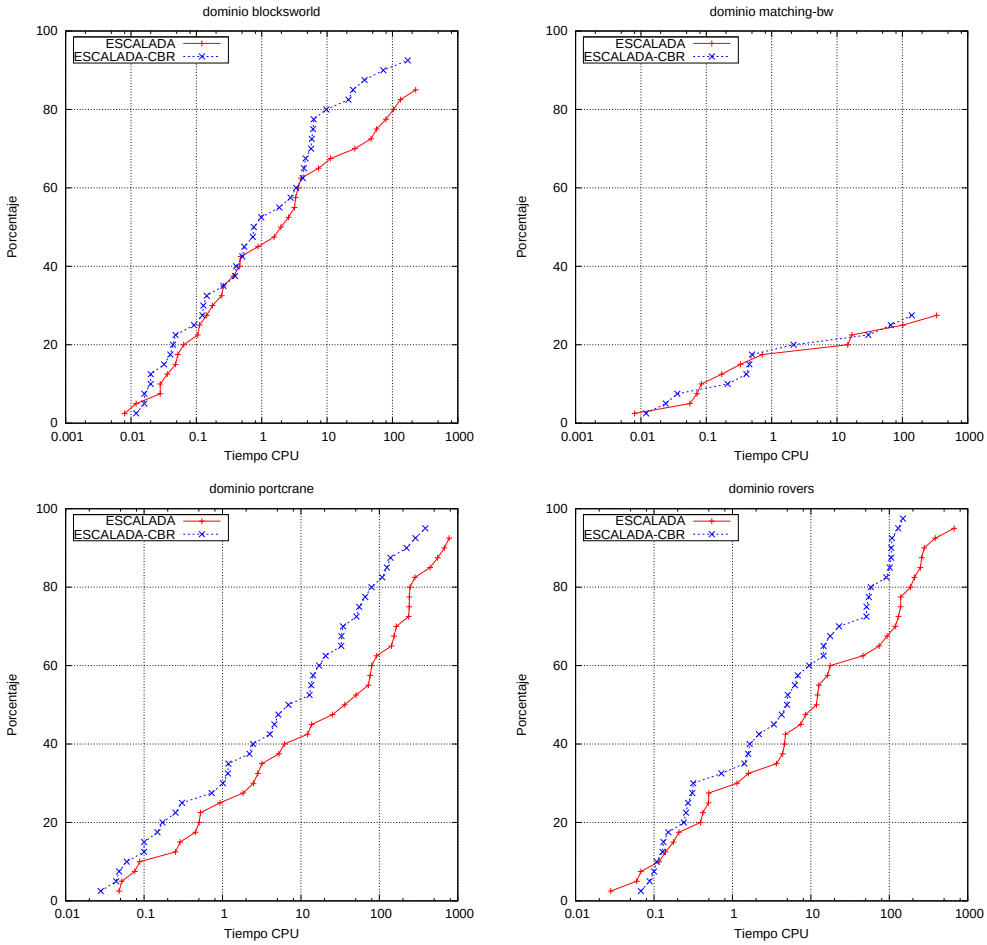


Figura 6.9: Distribución acumulada de tiempo de CPU para experimentos con Escalada y CBR.

en los experimentos de h^{diff}. En el *Parking*, Escalada+CBR resuelve un problema más. Sin embargo, las mejoras en tiempo o en calidad no son apreciables. En este dominio h^{FF} tiene mínimos locales, perjudiciales por ejemplo para EHC, pero sin embargo, está localmente bien informada. Esta situación se aprecia por ejemplo en los estados de la solución. En ellos, el valor heurístico tiene que aumentar durante ciertos pasos, pero el resto de estados candidatos del correspondiente nivel tenían un valor heurístico aún peor. De esta manera Escalada puede encontrar soluciones, inclusive con calidad similar al resto de algoritmos. Por otro lado, las transiciones intermedias por las que tienen que pasar los coches no son identificables por las secuencias, que suelen ser más cortas por el proceso de recuperación. Debido a esto, muchos pasos de la solución se siguen guiando por la función heurística al no encontrar un nodo recomendado que corresponda con dichos pasos intermedios.

En el *Logistics*, a pesar de que se obtiene el mismo número de problemas re-

sueltos, Escalada+CBR obtiene una pequeña mejora en la distribución acumulada de tiempo y en la calidad de las soluciones. Como se vio en los experimentos de h^{diff}, el algoritmo obtiene planes de mala calidad porque incluye en las soluciones muchas acciones de mover los medios de transporte de forma innecesaria. Las secuencias de tipo recomiendan las acciones de recoger o dejar los paquetes siempre que sea posible, y esto permite reconocer las acciones innecesarias en muchas situaciones. En el *Portcrane* hay una mejora significativa en la distribución acumulada de tiempo, como se aprecia en la Figura 6.9. Las secuencias de tipo permiten reconocer los tipos de contenedores, porque son identificables mediante las diferentes propiedades que tienen asociadas ($frozen_1$, $locked_1$ o $dryvan_1$). Esto le permite a Escalada+CBR elegir correctamente las acciones de levantar y dejar los diferentes contenedores, lo que se traduce en una mejora en la calidad de los planes. Al igual que en otros dominios la mejora en tiempo se debe a la reducción significativa en el número de evaluaciones de la función heurística. Para el caso del *Mystery'*, Escalada+CBR consigue mejorar el número de problemas resueltos. No obstante, al ser un dominio con caminos sin salida, esta mejora no es relevante por el gran número de problemas que se dejan de resolver debido a que el algoritmo realiza búsqueda local.

En el *Satellite* y en el *Rovers* se observa que Escalada+CBR tiene una mejor distribución acumulada de tiempo. En el *Rovers*, además de resolver un problema más, esta mejora es más notoria como se aprecia en la Figura 6.9. Sin embargo, en el *Satellite* se resuelven dos problemas menos. En este dominio la recuperación de las secuencias más cortas no resulta beneficioso, porque en los problemas grandes serían necesarias secuencias que continuamente recomienden las acciones de girar los satélites y tomar las imágenes.

Por otro lado, en varias de las distribuciones acumuladas de tiempo se observa un corte inicial (siempre por debajo de un segundo) en el que Escalada es más eficiente que Escalada+CBR. Esto se debe a que el proceso de recuperación tiene un coste computacional al inicio de la planificación que no compensa cuando los estados de los problemas pequeños son evaluados rápidamente por la función heurística. Cuando los estados son más grandes y hay más nodos evaluados, el tiempo empleado por CBR en la recuperación ya no resulta relevante. Este punto de corte representaría el tiempo de planificación a partir del cual tendría sentido utilizar las recomendaciones en el algoritmo.

Al analizar en conjunto los resultados de Escalada(+CBR) se pueden resumir los detalles comunes en:

- Escalada y Escalada+CBR obtienen soluciones de muy baja calidad en la mayoría de dominios, comparado con el resto de algoritmos, y tienen un comportamiento deficiente en los estados de punto muerto.

- La combinación de las acciones útiles con la heurística aportada por el ratio de recomendación permite discriminar localmente los estados candidatos en situaciones en las que h^{FF} no aporta dicha discriminación. Esto se traduce en

la reducción de nodos evaluados, y en algunos dominios la mejora de la calidad de las soluciones al poder eliminar de los planes acciones innecesarias.

- El conocimiento aportado por las secuencias influye directamente sobre el rendimiento de Escalada+CBR, debido a la estrategia de selección directa. Un buen conocimiento, con una buena recuperación, se traducirá en un buen seguimiento.

- El coste computacional de la recuperación y el asociado a la comparación de cada estado implica un punto de corte en la distribución acumulada de tiempo. No compensa la utilización de casos en problemas triviales.

6.5.3. Resultados en EHC+CBR

Los experimentos con EHC guiados por CBR miden las ventajas de ordenar la evaluación de los nodos con las recomendaciones de las secuencias de tipo. La Tabla 6.4 muestra los resultados de comparar el algoritmo EHC con su variante EHC+CBR. En los resultados se muestra el número y porcentaje de problemas resueltos utilizando ambas técnicas. Se observa que EHC+CBR mejora el número de problemas resueltos en cinco de los ocho dominios de evaluación y sólo empeora el comportamiento de EHC en el *Matching Blocksworld*.

Dominio	EHC		EHC+CBR	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	19	47.5 %	**20**	**50.0 %**
Matching-bw	**4**	**10.0 %**	3	7.5 %
Parking	30	75.0 %	**34**	**85.0 %**
Logistics	**40**	**100.0 %**	**40**	**100.0 %**
Mystery'	26	65.0 %	**29**	**72.5 %**
Portcrane	**39**	**97.5 %**	**39**	**97.5 %**
Satellite	38	95.0 %	**39**	**97.5 %**
Rovers	33	82.5 %	**38**	**95.0 %**

Tabla 6.4: Problemas resueltos para el experimento de EHC+CBR.

La Figura 6.10 muestra la distribución acumulada de tiempo en cuatro dominios. La correspondiente al *Blocksworld* muestra que ambas curvas se cruzan constantemente, revelando que no hay un algoritmo que domine a otro, por lo que ordenar la evaluación de nodos no tiene sentido en este dominio. El *Blocksworld*, además de caracterizarse por tener grandes *plateaus*, tiene la particularidad de que existen muchos nodos de escape de *plateau* a la profundidad mínima que puede encontrar la búsqueda en amplitud realizada por EHC. Esa característica hace que en muchas situaciones no valga la pena ordenar para evaluar todos los nodos hasta la salida del *plateau*. Tampoco se observa que un algoritmo domine en la distribución acumulada de longitud del plan, mostrada en la Figura C.5 del Apéndice C. Al igual que sucedía en los algoritmos de Escalada y Escalada+CBR, el algoritmo EHC y

EHC+CBR obtienen unos resultados muy pobres en el *Matching Blocksworld*. Esto se debe a que tampoco EHC o EHC+CBR son apropiados para dominios con caminos sin salida. En el *Parking* EHC+CBR consigue resolver cuatro problemas más y la distribución acumulada de tiempo muestra una mejora discreta, pero que siempre domina a EHC desde antes de los 0.1 segundos. La ordenación resulta beneficiosa porque en muchos problemas se da la siguiente situación: Un coche x en un bordillo tiene que ubicarse en otro sitio, pero tiene otro coche y delante. Si hay dos estados sucesores, uno que coloca el coche y en un bordillo libre, y otro que lo coloca detrás de otro coche z, ambos suelen mejorar el valor heurístico del estado original. Sin embargo, colocar el coche y detrás del coche z y no ocupar el bordillo libre suele ser más beneficioso porque en ocasiones el coche x es el que debe ocupar el bordillo libre en el estado final y eso significaría una forma más directa de alcanzar la meta.

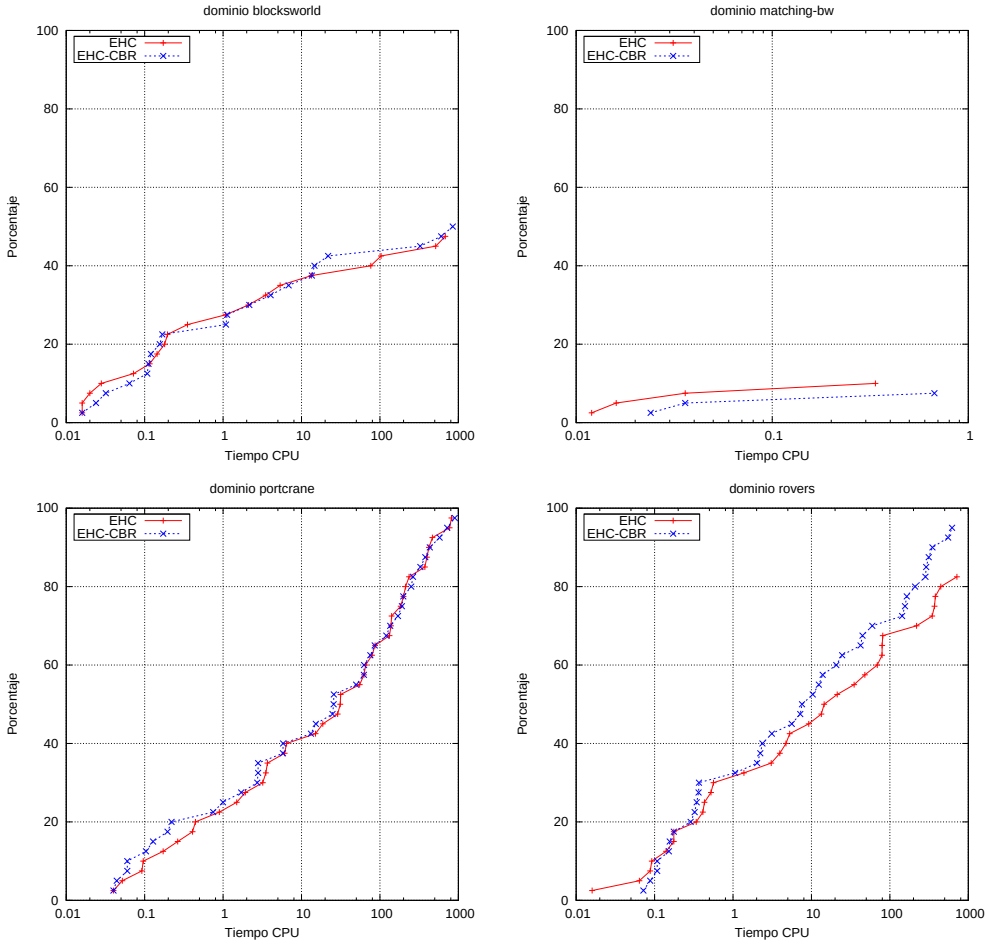


Figura 6.10: Distribución acumulada de tiempo de CPU para experimentos con EHC y EHC+CBR.

El *Logistics* es un dominio fácil para el EHC. Tanto el algoritmo original como su variante CBR resuelven todos los problemas. Sin embargo, la distribución acumulada de tiempo resulta un poco peor en EHC+CBR, manteniendo la misma calidad de los planes que EHC. La explicación se encuentra en la definición del dominio PDDL que tiene todas las acciones de cargar y descargar primero que las acciones de mover aviones o camiones. Todas las acciones de cargar y descargar mejoran el valor heurístico, por lo que la característica avariciosa del algoritmo se aprovecha de ese orden. Los *plateaus* que se producen con las acciones de mover medios de transporte son de profundidad dos dentro de la búsqueda local en amplitud, y tienen a su vez varios nodos de escape, por lo que una ordenación en estas situaciones tampoco tiene mucho sentido. Al final, la pérdida de tiempo por parte de EHC+CBR se debe al coste computacional de equiparar las secuencias con los estados sucesores, sin que esto beneficie el proceso de planificación. Pruebas experimentales adicionales con el orden invertido de las acciones en la definición del PDDL comprobaron que EHC+CBR mejora en tiempo, porque en esta versión sí tiene sentido la ordenación para la evaluación de nodos. En la evaluación sobre el mismo conjunto de prueba se obtuvo 39 problemas resueltos para EHC y 40 problemas resueltos para EHC+CBR. La Figura 6.11 muestra la distribución acumulada de nodos evaluados para la versión del *Logistics* con las acciones invertidas. Al lado la misma distribución en la versión estándar del dominio. En la gráfica del *Logistics* invertido se puede apreciar la reducción en el número de evaluaciones gracias a que las recomendaciones CBR hacen evaluar primero las acciones de cargar o descargar.

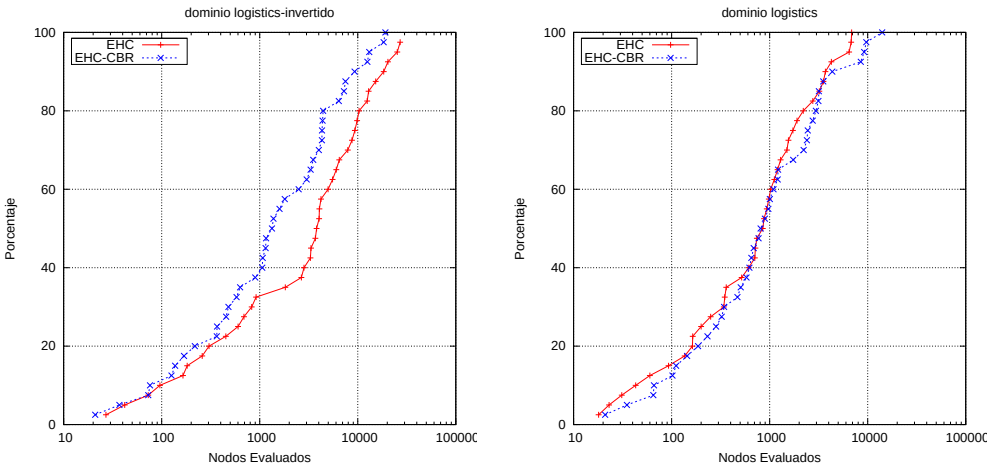


Figura 6.11: Distribución acumulada de nodos evaluados para los experimentos de EHC y EHC+CBR en las dos versiones del *Logistics*.

En el *Portcrane* se da una situación similar. Los *plateaus* suelen ser de profundidad dos con varios nodos de escape. Sí tiene sentido ordenar las evaluaciones, pero el beneficio es equivalente al coste de las comparaciones con las secuen-

cias, dando como resultado un comportamiento equivalente de ambos algoritmos, apreciado en la Figura 6.10. EHC+CBR, gracias a la correcta selección de las acciones de levantar y dejar los contenedores, consigue mejorar la calidad de los planes en unas pequeñas acciones por problema. En los 39 problemas resueltos por ambas técnicas, EHC obtuvo un promedio de 289.9 acciones por plan, mientras que EHC+CBR obtuvo un promedio de 263.5 acciones. En el *Mystery'* se observa una mejora en el número de problemas resueltos, tiempo y longitud de los planes, porque la ordenación de las acciones, dando preferencia a las acciones de cargar y descargar, hace el algoritmo caiga en menos caminos sin salida. La distribución acumulada de tiempo se puede apreciar en la Figura C.4 del Apéndice C.

EHC+CBR obtiene en el *Satellite* y en el *Rovers* una mejora en el número de problemas resueltos y una mejora significativa en la distribución acumulada de tiempo, que se aprecia después de los problemas iniciales. En estos dominios se observa también un punto de corte en las gráficas, que indican que por debajo de ese tiempo no compensa el coste computacional que representa la recuperación de las secuencias al inicio de los problemas. La ganancia en tiempo se debe a la considerable reducción de nodos evaluados en ambos dominios gracias a la correcta ordenación de las evaluaciones. En regiones del espacio de búsqueda donde hay un alto factor de ramificación, la selección de las acciones de calibrar y enviar las muestras en el *Rovers* y las acciones de calibrar y tomar imagen en el *Satellite* permiten descartar el resto de acciones que no mejoran el valor heurístico de su correspondiente estado padre. El número de acciones en los planes puede variar en pocas unidades de un problema a otro, pero en general puede considerarse que tienen un comportamiento similar.

Interpretando los resultados de los dos algoritmos CBR que se han visto hasta el momento, se puede deducir que el comportamiento de Escalada+CBR depende en gran medida del conocimiento aprendido en las secuencias de tipo, dado que la selección directa sólo toma en cuenta la heurística dictada por la función r. Sin embargo, el comportamiento de EHC+CBR no depende directamente del conocimiento de las secuencias porque éstas sólo se utilizan para ordenar las evaluaciones de los nodos con la función heurística. En este sentido, hay muchas circunstancias que influyen dependiendo de cada dominio, que se pueden resumir en:

- Definición del PDDL: El orden en que están declaradas las acciones en el dominio influye directamente en la posibilidad de aprovechar la propiedad avariciosa de EHC. El algoritmo EHC+CBR podrá aprovechar esta propiedad si el dominio no tiene la mejor ordenación de acciones posible o si la ordenación correcta depende del estado actual.

- Factor de ramificación: El número de operadores diferentes y el número acciones instanciadas influye directamente en que la ordenación sea más relevante o no. EHC+CBR mostrará resultados más significativos bajo estas condiciones como se apreció en el dominio *Rovers*.

- Nodos de escape de *plateau*: La búsqueda local en amplitud de EHC se re-

aliza con la técnica de poda de las acciones útiles, por lo que en ciertos dominios, en el nivel de profundidad que se mejora el valor heurístico, la mayoría de los nodos son válidos para continuar la búsqueda. EHC+CBR conseguirá una reducción en el número de evaluaciones siempre y cuando no haya muchos nodos de escape en el último nivel de la búsqueda en amplitud y por tanto tenga sentido realizar una ordenación sobre dichos nodos.

6.5.4. Resultados en WBFS+CBR

Los experimentos con el algoritmo WBFS guiado por CBR intentan mostrar en qué medida las recomendaciones CBR pueden diferenciar los nodos de igual evaluación mediante el supuesto de que están siguiendo alguna de las secuencias de tipo o no. La Tabla 6.5 muestra los resultados en número y porcentaje de problemas resueltos para los dominios de evaluación. WBFS+CBR consigue mejorar el comportamiento del algoritmo original en cinco de los ocho dominios y sólo empeora en dos de ellos.

Dominio	WBFS		WBFS + CBR	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	24	60.0 %	**25**	**62.5 %**
Matching-bw	25	62.5 %	**27**	**67.5 %**
Parking	32	80.0 %	**33**	**82.5 %**
Logistics	**19**	**47.5 %**	17	42.5 %
Mystery'	23	57.5 %	**26**	**65.0 %**
Portcrane	12	30.0 %	**14**	**35.0 %**
Satellite	**18**	**45.5 %**	14	35.5 %
Rovers	**15**	**37.5 %**	**15**	**37.5 %**

Tabla 6.5: Problemas resueltos para el experimento de WBFS+CBR.

Las distribuciones acumuladas de tiempo para cuatro dominios se muestran en la Figura 6.12. Las distribuciones acumuladas de longitud del plan se encuentran en los resultados adicionales del Apéndice C. De los resultados del *Blocksworld* y el *Matching Blocksworld* no se puede extraer ninguna idea concluyente, porque las distribuciones acumuladas de tiempo, longitud y número de nodos evaluados muestran que ninguna técnica logra dominar a otra a lo largo de las diferentes dificultades de los problemas de prueba. En el *Parking* se observa que WBFS+CBR domina a WBFS en la gráfica de distribución acumulada de tiempo a partir de los 2 segundos. Sin embargo, en el análisis detallado de los problemas se aprecia que no es una mejora sistemática, porque en varios problemas el tiempo y número de nodos evaluados seguía siendo mejor con WBFS.

WBFS+CBR resuelve dos problemas más en el *Portcrane* y dos menos en el *Logistics*. Para ambos dominios no queda claro que las secuencias afecten directamente a su comportamiento porque las distribuciones acumuladas de tiempo no muestran que un algoritmo domine a otro. El detalle a resaltar es que tanto WBFS

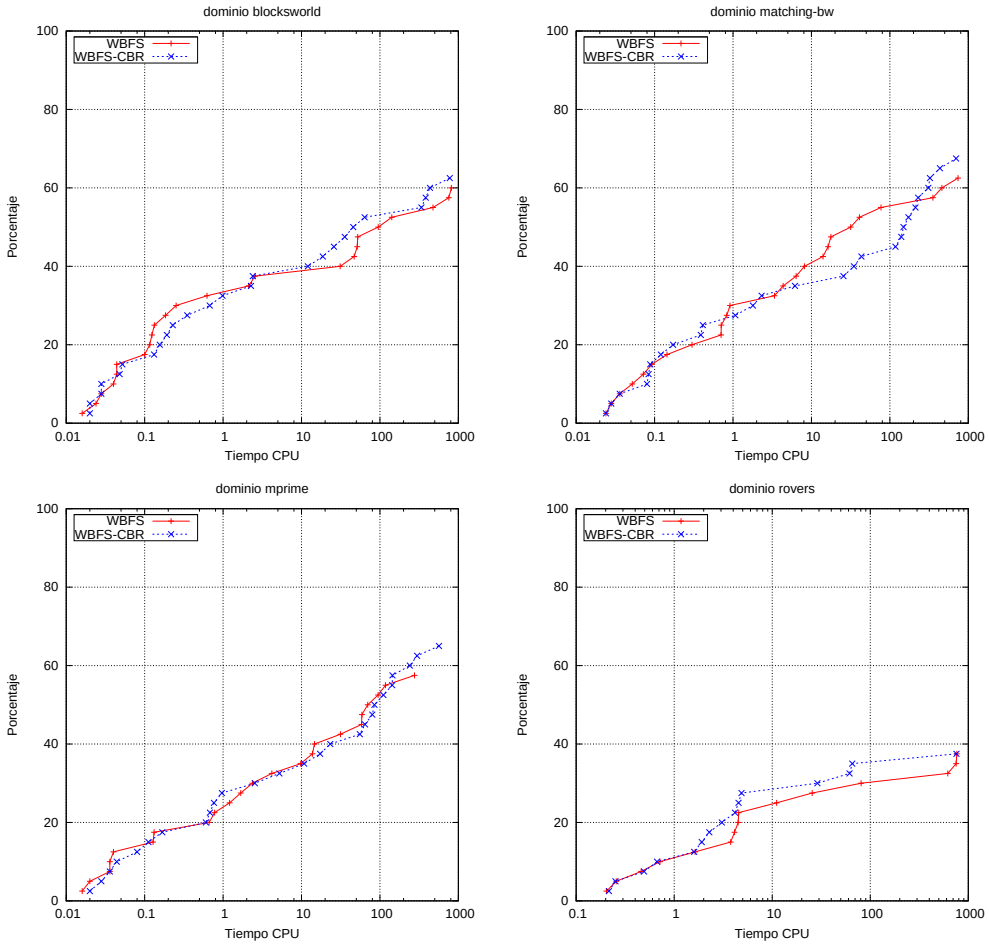


Figura 6.12: Distribución acumulada de tiempo de CPU para experimentos con WBFS y WBFS+CBR.

como WBFS+CBR resuelven menos problemas que con técnicas de búsqueda local. A diferencia del *Matching Blocksworld* que sí mejoró su comportamiento con la búsqueda global, el dominio *Mystery'* no reporta resultados mejores que EHC. El dominio *Mystery'* tiene estados de punto muerto, pero estos son menos frecuentes en el espacio de estados que en el *Matching Blocksworld*. WBFS+CBR mejora en tres problemas al algoritmo original, pero ninguna técnica domina a otra en las gráficas de distribución acumulada de tiempo, longitud del plan o número de nodos evaluados.

El *Satellite* y el *Rovers* presentan resultados similares al resto de dominios y también ven afectado su comportamiento en comparación con los algoritmos de búsqueda local. En general las secuencias de tipo valen para discriminar localmente entre unos estado y otros, pero no pueden hacerlo efectivamente dentro de una búsqueda global. Así como las pruebas de poda exclusiva fueron descartadas

por evaluaciones experimentales previas, WBFS+CBR no puede considerarse como una modificación de interés al algoritmo WBFS, porque en ningún dominio muestra un comportamiento que mejore sistemáticamente al algoritmo estándar.

6.5.5. Resultados en EHC-Lookahead+CBR

En los experimentos con EHC-Lookahead o WBFS-Lookahead guiados por CBR se intenta evaluar si la ordenación de las acciones que conforman los estados *lookahead* es relevante para la búsqueda. Para los casos que esta ordenación sea relevante, se debe determinar si el tiempo invertido en ordenar dichas acciones, utilizando el conocimiento de las secuencias de tipo, compensa frente al tiempo total de planificación. En el caso particular de EHC-Lookahead, la construcción del estado *lookahead* y posible ordenación de sus acciones tiene sentido si el valor heurístico del estado *lookahead* mejora al valor heurístico de su estado padre. De lo contrario se continuarían evaluando localmente en amplitud el resto de sucesores.

La Tabla 6.6 muestra el número y porcentaje de problemas resueltos utilizando ambas técnicas en los dominios en los que no se resuelven todos los problemas. La técnica de estados *lookahead* mejora drásticamente el comportamiento en algunos dominios, por lo que tanto EHC-Lookahead, como su variante CBR resuelven los cuarenta problemas dentro del límite de tiempo. Para estos dominios la Tabla 6.7 muestra el tiempo máximo que reporta la distribución acumulada de tiempo en ambas técnicas. Esto sería el equivalente a decir que es el tiempo utilizado por el problema más complicado de resolver para una técnica, tomando en cuenta que no se refiere necesariamente al mismo problema.

Dominio	EHC-Lookahead		EHC-Lookahead + CBR	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	26	65.0 %	**28**	**70.0 %**
Matching-bw	6	15.0 %	**7**	**17.5 %**
Parking	**40**	**100.0 %**	37	92.5 %
Mystery'	**22**	**55.0 %**	**22**	**55.0 %**

Tabla 6.6: Problemas resueltos en dominios difíciles para el experimento de EHC-Lookahead+CBR.

Dominio	EHC-Lookahead	EHC-Lookahead+CBR
Logistics	**1.06**	1.90
Portcrane	11.87	**5.72**
Satellite	**1.62**	3.04
Rovers	707.37	**41.56**

Tabla 6.7: Tiempo máximo en los dominios sencillos para el experimento EHC-Lookahead+CBR.

En el *Blocksworld*, EHC-Lookahead+CBR consigue resolver dos problemas

más que EHC-Lookahead. Sin embargo, sólo 21 problemas fueron resueltos por ambas técnicas. En este dominio, los planes relajados suelen ser incompletos, y la construcción de estados *lookahead* con varias acciones resulta muy difícil. Por ejemplo, si la acción `(PICKUP X)` es la primera en el plan relajado, y el bloque `X` no puede colocarse en su destino final en ese momento, el brazo quedará ocupado y ninguna de las acciones restantes del plan relajado se podrá aplicar, porque faltaría la acción `(PUTDOWN X)`. Casos como éste hacen que en EHC-Lookahead se produzca el efecto de construir el estado *lookahead* y después descartarlo porque no mejora el valor heurístico. No obstante, las recomendaciones CBR parecen influir positivamente en la ordenación de las acciones del plan relajado cuando se consigue construir los estados *lookahead*. Por otro lado, en el *Parking* EHC-Lookahead+CBR resuelve tres problemas menos que EHC-Lookahead. Sin embargo, las distribuciones acumuladas muestran comportamientos similares. Al realizar un análisis detallado de los problemas se encuentran variaciones considerables en el tiempo de planificación. Dado que no son sistemáticas, sino que aventajan a EHC-Lookahead en ocasiones y a EHC-Lookahead+CBR en otras, sólo se puede interpretar que las ordenaciones diferentes permiten explorar caminos diferentes sin que esto represente una superioridad de una técnica sobre la otra.

EHC-Lookahead+CBR logra apenas resolver siete problemas en el *Matching Blocksworld*, uno más que EHC-Lookahead. Las técnicas *lookahead* son muy avariciosas, haciendo caer la búsqueda constantemente en caminos sin salida. EHC estándar se comporta mal en el *Matching Blocksworld*, y sus causas se reproducen en la versión con *lookahead*. Los planes relajados, además de ser incompletos, tienen acciones equivocadas, porque eligen indistintamente el brazo sin tomar en cuenta la polaridad de los bloques. El *Mystery'*, que tiene también caminos sin salida, presenta resultados peores que EHC; 26 frente a 22 problemas resueltos. Los problemas que EHC-Lookahead no pudo resolver y EHC si, obtuvieron un fallo en la búsqueda por alcanzar camino sin salida. Esto revela que la característica avariciosa adicional que aportan los estados *lookahead* no beneficia la búsqueda en este tipo de dominios. La ordenación de las acciones aportada por CBR no afecta el comportamiento del algoritmo original.

Para el resto de dominios, la técnica de los estados *lookahead* produce una mejora impresionante en tiempo de planificación, que puede llegar hasta tres órdenes de magnitud como sucede en el *Logistics* y en el *Satellite*. La causa fundamental es la drástica reducción de nodos evaluados, que se consigue porque los estados *lookahead* tienen muchas acciones encadenadas. Se pueden encadenar un gran número de acciones porque los planes relajados en estos dominios contienen las acciones correctas y en muchos casos el orden de las acciones es razonable, permitiendo así que las acciones se puedan aplicar unas a continuación de otras. No obstante, cabe analizar si una ordenación diferente aportada por las recomendaciones de las secuencias puede aportar una mejora adicional o por el contrario perjudica el tiempo total de planificación porque el orden resulta irrelevante.

Los estados *lookahead* en el *Logistics* pueden encadenar muchas acciones. Por ejemplo, el problema con el plan de mayor longitud (421) apenas evaluó trein-

ta nodos. En algunos estados *lookahead* se entrecruzan acciones equivocadas de movimientos de medios de transporte. Estas acciones son recuperadas por lo general, en el siguiente estado *lookahead* que se cree en el plan. En estos casos se pierde calidad en los planes, comparado con los resultados de EHC. La ordenación realizada por EHC-Lookahead+CBR surte un efecto negativo al tener los planes relajados una buena ordenación de las acciones cargar y descargar, las típicas recomendaciones de las secuencias en este dominio. El tiempo invertido en ordenar innecesariamente se traduce en un tiempo adicional de planificación que se aprecia en la Figura 6.13. La línea correspondiente a EHC-Lookahead-CBR esta nombrada como CABALA-EHC del acrónimo (*CAse-BAsed LookAhead*)+EHC.

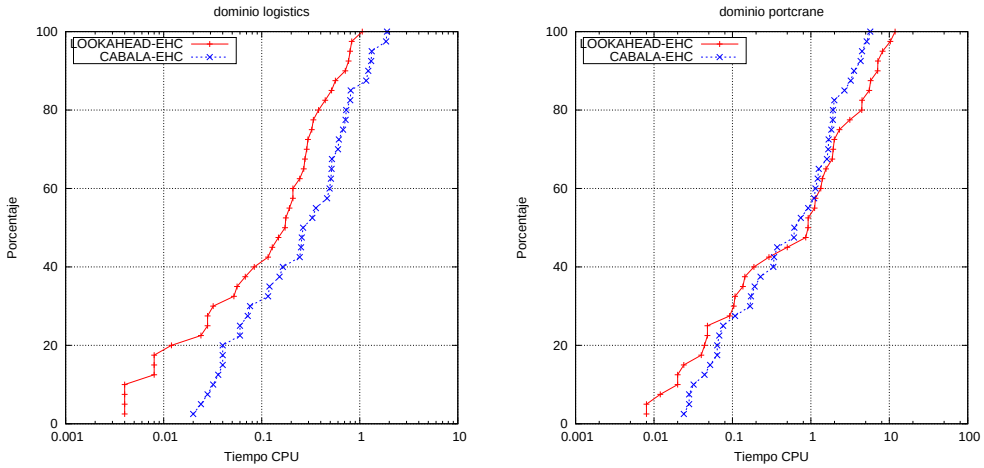


Figura 6.13: Distribución acumulada de tiempo de CPU para experimentos con EHC-Lookahead y EHC-Lookahead+CBR.

En el *Portcrane*, la ordenación de las acciones de los *lookahead* compensa en tiempo a partir de los problemas que tardan 0.4 segundos, tal como se muestra en la distribución acumulada de tiempo en la Figura 6.13. Sin embargo, a pesar de que la ordenación tiene sentido en este dominio, los planes relajados no contienen las mejores acciones, lo que perjudica la calidad de los planes finales al compararlos con el resto de algoritmos. Dado que las acciones correspondientes a levantar o a dejar un contenedor según su tipo, no son las apropiadas en el plan relajado, se puede intuir que una mejor construcción del plan relajado permitiría que la ordenación de sus acciones fuera más relevante.

Los planes relajados en el *Satellite* tienen las acciones correctas y su orden es válido, salvo por la acción `SWITCH-OFF`. El momento correcto de aplicar esta acción tampoco es detectado por las secuencias de tipo, que al estar centradas en los objetos particulares, no pueden determinar el número de imágenes que hay que tomar con un instrumento antes de apagarlo. Por eso, el tiempo invertido en las equiparaciones de las secuencias se suma al tiempo total de planificación, por lo que EHC-Lookahead+CBR produce un comportamiento peor. Por ejemplo el tiem-

po máximo empleado en un problema fue de 1.62 segundos para EHC-Lookahead frente a los 3.04 de EHC-Lookahead+CBR. Por otro lado, en el *Rovers*, las acciones del plan relajado son las correctas, pero el orden no es el que permite encadenar más acciones. Por ejemplo, las acciones de `NAVIGATE` aparecen en el plan relajado antes de las acciones de `CALIBRATE`. Las secuencias de tipo recomiendan el orden correcto, permitiendo que después que un *rover* esté calibrado pueda tomar la imagen si se desplaza al sitio correspondiente y así seguir encadenando más acciones. Como de todas formas ambas técnicas reducen drásticamente el número de evaluaciones, la ventaja de las ordenaciones de EHC-Lookahead+CBR, sólo se aprecia en los problemas grandes, donde se obtienen mejoras en tiempo (Tabla 6.7).

6.5.6. Resultados en WBFS-Lookahead+CBR

Para estos experimentos, se realiza la misma estrategia de ordenar con recomendaciones CBR las acciones para construir los estados *lookahead*, pero en este caso se aplica a un algoritmo de mejor primero. Se utiliza el peso $w_h = 3$ igual que en el WBFS, y en este caso se utiliza la técnica de poda de las acciones útiles. Se decidió utilizar esta técnica porque la técnica de los *lookahead* es de por sí avariciosa y no tendría mucho sentido reducir evaluaciones de nodos con los estados *lookahead* si por otro lado se pierde tiempo evaluando estados que en la mayoría de dominios está probado que no son necesarios.

Se han separado igualmente los resultados de los dominios difíciles de los que resultaban más sencillos para los algoritmos. La Tabla 6.8 muestra el número y porcentaje de problemas resueltos en los dominios en los que no se resolvieron todos los problemas. La Tabla 6.9 muestra el tiempo máximo utilizado para resolver un problema en los dominios en los que ambas técnicas resolvieron los 40 problemas.

WBFS-Lookahead+CBR logra mejorar el rendimiento del algoritmo original en tres de los cinco dominios que se consideran difíciles. El rendimiento se ve empeorado en el *Blocksworld* y en el *Matching Blocksworld*. El WBFS-Lookahead y su variante CBR no son tan avariciosos como la pareja EHC-Lookahead(+CBR). Esto se refleja en que dominios como el *Logistics*, *Rovers* o *Satellite* no se resuelven tan rápido como con los *lookahead* sobre EHC. En cambio, tanto WBFS-Lookahead como WBFS-Lookahead+CBR resultan más robustos en una mayor cantidad de dominios, ya que por ejemplo pueden resolver problemas en el *Matching Blocksworld*, en donde otros algoritmos avariciosos no pueden.

En los dominios difíciles, la diferencia de problemas resueltos no es en ningún caso mayor que dos. Adicionalmente, las distribución acumulada de tiempo de los ocho dominios se cortan al menos más de una vez, indicando que en general, ninguna de las técnicas es dominante sobre la otra. Si se observa el tiempo acumulado para resolver los problemas resueltos por ambas técnicas, se puede apreciar que WBFS-Lookahead+CBR evalúa mucho menos nodos en el *Rovers* y hasta en el *Matching Blocksworld*, donde resuelve un problema menos que WBFS-Lookahead. Esto revela que la ordenación de las acciones de los estados *lookahead*

Dominio	WBFS-Lookahead		WBFS-Lookahead+CBR	
	Resueltos	Porcentaje	Resueltos	Porcentaje
Blocksworld	**32**	**80.0 %**	28	70.0 %
Matching-bw	**30**	**75.0 %**	28	70.0 %
Parking	39	97.5 %	**40**	**100.0 %**
Mystery'	33	82.5 %	**34**	**85.0 %**
Rovers	38	95.0 %	**39**	**97.5 %**

Tabla 6.8: Problemas resueltos para el experimento de WBFS-Lookahead+CBR.

Dominio	WBFS-Lookahead	WBFS-Lookahead+CBR
Logistics	**11.85**	14.93
Portcrane	**37.40**	40.47
Satellite	**40.03**	46.78

Tabla 6.9: Tiempo máximo en los dominios sencillos para el experimento WBFS-Lookahead+CBR.

resulta provechosa en algunos problemas, pero que el coste computacional que representa una ordenación errónea es más alto, que por ejemplo una mala recomendación en los algoritmos menos avariciosos como Escalada o EHC.

En los dominios fáciles, las diferencias entre WBFS-Lookahead y WBFS-Lookahead+CBR son aún menores. El tiempo máximo para resolver un problema es también semejante en ambos casos. En los tres dominios, WBFS-Lookahead+CBR se ve perjudicado porque los planes relajados en estos dominios contienen las acciones correctas y traen una ordenación aceptable. La ordenación adicional que podría aportar las recomendaciones CBR no se traducen en mejoras de rendimiento. El *Rovers*, que en EHC-Lookahead se clasificaba como fácil por haber resuelto los 40 problemas, en WBFS-Lookahead pasa a ser de la categoría de los difíciles. En WBFS-Lookahead(+CBR) se evalúan más nodos que en EHC-Lookahead(+CBR) y eso hace que no se lleguen a resolver los últimos problemas.

6.5.7. Discusión

Se ha visto cómo las secuencias de tipo pueden integrarse en los diferentes algoritmos de búsqueda con estrategia de ordenación o poda, según convenga al algoritmo. El objetivo de los experimentos con los algoritmos CBR consistía en probar si un conocimiento de control aportado por las secuencias de tipo podía traducirse en una mejora en rendimiento por medio de una reducción de nodos evaluados por la función heurística. El comportamiento de los algoritmos CBR no responde directamente a la clasificación que inicialmente se ha hecho de los dominios (construcción, transporte y secuencias). La topología del espacio de estados, la existencia de caminos sin salida, el conocimiento que se guarda en las secuencias y en ocasiones hasta el orden de los operadores en el dominio PDDL, son factores

que influyen en un mejor o peor rendimiento de los algoritmos CBR. A continuación se expondrán los detalles comunes que se pueden encontrar al analizar los resultados en conjunto.

El algoritmo Escalada+CBR es el que muestra más diferencias en las distribuciones acumuladas de tiempo respecto a su algoritmo base. En este sentido queda claro que una técnica de selección directa es la forma más avariciosa de reducir evaluaciones de la función heurística. En varios dominios, la gran capacidad de mejora se debe precisamente al mal rendimiento de Escalada en dichos dominios por la incapacidad de h^{FF} de decidir localmente por el camino correcto. Cabe señalar que la poda de las acciones útiles deja bastante restringido el ámbito de selección, por lo que el proceso de selección resulta más fácil. De producirse equivocaciones, éstas suelen tener efectos menores en el proceso de búsqueda. Contrario a lo que puede intuirse inicialmente, en términos de calidad la selección directa CBR puede llegar a ser más eficiente que la heurística a la hora de seleccionar entre las acciones útiles. Por ejemplo en la Figura 6.14 se puede ver cómo la distribución acumulada de la longitud del plan muestra claramente la ventaja del CBR en los dominios *Blocksworld* y *Portcrane*.

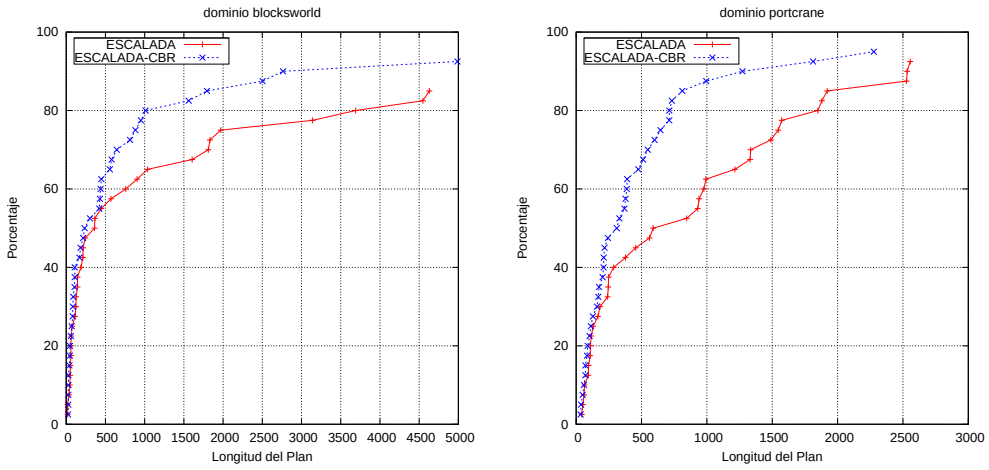


Figura 6.14: Distribución acumulada de longitud del plan para experimentos de Escalada(+CBR).

El algoritmo EHC+CBR reporta resultados equivalentes en términos de calidad con respecto al algoritmo base. Sin embargo, sí puede mejorar el rendimiento en tiempo por una reducción en el número de nodos evaluados gracias a la estrategia de ordenación a la hora de evaluar los estados sucesores. La estrategia de ordenación se ve influida además por otros factores como el orden de definición de los operadores del PDDL, el factor de ramificación o por la topología del espacio de estados que marca los nodos de escape de *plateaus*. Estos factores pueden marcar el que las recomendaciones sean beneficiosas o no en cada dominio.

Las técnicas *lookahead* son un salto cualitativo para la planificación. El algorit-

mo EHC-Lookahead es el algoritmo más rápido en los dominios en los que el plan relajado contiene acciones reales del plan final y estas acciones están ordenadas de una forma válida. Su inconveniente es que su estrategia es muy avariciosa y no resulta rentable para los dominios en los que el plan relajado no es bueno o existen caminos sin salida. En cambio, los algoritmos WBFS-Lookahead(+CBR) son más robustos en una mayor cantidad de dominios. En este sentido queda de manifiesto que además del conocimiento de control que aportan las secuencias de tipo, también es relevante la elección del algoritmo de búsqueda, sobre todo en algunos dominios. Por ejemplo en el *Satellite*, el algoritmo más rápido con diferencia es el EHC-Lookahead, dada la calidad de sus planes relajados y la drástica reducción de nodos evaluados. Sin embargo, para el *Mystery'*, el mejor algoritmo es el WBFS-Lookahead+CBR, porque aparte de los estados *lookahead* puede reconsiderar otros caminos al encontrar caminos sin salida. La desventaja es que no es posible decidir de antemano el algoritmo correcto para un dominio, salvo que ese conocimiento sea capturado por alguna técnica de aprendizaje.

Capítulo 7

Análisis de las Secuencias de Tipo

En este capítulo se presentan los estudios realizados sobre las bases de casos de los diferentes dominios de evaluación con el objetivo de analizar y comprender qué representa el conocimiento aprendido y cuál es la mejor forma de aprovecharlo en cada dominio en particular.

Tradicionalmente, los sistemas de razonamiento basado en casos presentan la problemática del tamaño de la base de casos en relación con los nuevos casos que se siguen aprendiendo. Se asume que llegado a un punto de vida del sistema, el coste computacional de almacenar un nuevo caso es más alto que el beneficio que aporta aprenderlo. Otro aspecto que suele presentar problemática en estos sistemas es la valoración de los casos. Existen situaciones en que la medida de similitud no puede discriminar entre dos casos y sólo se puede elegir uno correctamente en función de un estudio de la utilidad de los mismos.

Este capítulo está formado por cinco secciones. La primera es un análisis del contenido de las secuencias de tipo para intentar explicar qué características particulares a cada dominio se recogen en los casos. La segunda sección es un análisis del seguimiento de las secuencias, dado que dependiendo de las características del dominio, algunos tipos recogerán casos que acertarán en mejor medida que otros a la hora de hacer la recomendación de nodos durante la búsqueda. La tercera sección contiene el desarrollo de un modelo de valoración de las secuencias, con el objetivo de determinar una preferencia cuando no se pueda decidir entre dos o más secuencias en las fases de recuperación o reutilización. Se incluye un apartado de evaluación de estas valoraciones para determinar si es posible utilizarlas para mejorar el rendimiento del sistema. La cuarta sección es un estudio de la generación de las bases de casos para analizar la problemática del tamaño correcto para cada dominio y si es posible determinar un punto donde se encuentre el mejor rendimiento. Finalmente, la última sección es una evaluación de la capacidad que tiene el sistema CBR para aprender secuencias durante el propio proceso de prueba, llamado comúnmente aprendizaje en línea. Para cerrar, se muestran un resumen de todos los resultados obtenidos en los diferentes apartados de evaluación.

7.1. Análisis Semántico de las Secuencias de Tipo

La idea de las secuencias de tipo surge como forma alternativa de almacenar de forma abstracta episodios de planificación. Uno de sus atractivos es precisamente el que esté centrado en los objetos que intervienen en el problema, y sus transiciones. Ahora, al prestar atención de modo particular en cada dominio surge la pregunta: ¿Qué información está realmente almacenanda en una secuencia? Por ejemplo, en un dominio de transporte, el objeto transportado generará las transiciones que muestran los eventos asociados a cargarlo y descargarlo en el medio de transporte, pero para el resto de objetos del dominio no está tan claro. Lo mismo pasa con otros dominios en el que los objetivos son otros. En esta sección se analizan secuencias de diferentes tipos en los dominios de prueba para descubrir qué características están recogiendo los casos.

7.1.1. Secuencias de Construcción

En los dominios de construcción, los objetos que deben terminar en una configuración específica producen secuencias que se pueden interpretar fácilmente a través de las transiciones que se dan a lo largo del problema. Por ejemplo en el *Blocksworld*, una secuencia que comience con un sub-estado de tipo $(on_1\ on_2)$ y termine con $(ontable_1\ clear_1)$, puede interpretarse como que el bloque estaba inicialmente encima de otro y un tercer bloque encima, y que al final terminó sobre la mesa y libre. De hecho, en este dominio las posibles transiciones son reconocibles y se muestran en la Figura 7.1. Se puede ver la figura como un diagrama de transición de estados del que se pueden deducir diferentes autómatas finitos utilizando como símbolos de entrada las acciones que acompañan a cada sub-estado de tipo en las secuencias. Por ejemplo, el sub-estado de tipo 1 podría ser el estado inicial de un autómata, y los sub-estados 1, 2, 4 y 5 podrían ser estados finales. A partir de estos autómatas se podrían generar las posibles secuencias que se aceptarían como transiciones válidas del dominio.

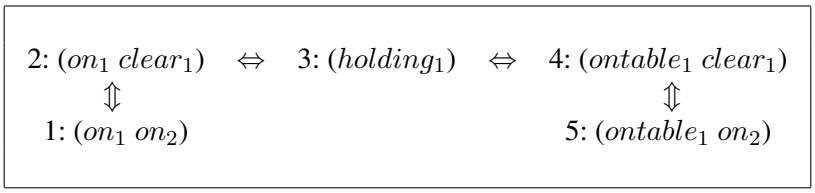


Figura 7.1: Transiciones entre los sub-estados de tipo en el *Blocksworld*.

Esta generación automática de secuencias, aunque es viable en este tipo de dominios, no resulta práctico en la realidad, porque muchas secuencias no se suceden en los problemas del *Blocksworld*. En el ejemplo, la secuencia que representa las transiciones (1-2-3-4-3-2) suele ser de las primeras generadas en la base de casos, por ser muy frecuente en el dominio. Sin embargo, la secuencia que representa las

transiciones (5-4-3-2-3-4) es muy difícil que aparezca en la base de casos, porque supone que un bloque que debe empezar y terminar sobre la mesa ha sido puesto encima de un bloque en algún momento del plan. En este sentido, se puede verificar que las secuencias de tipo permiten almacenar las secuencias típicas que se suceden en el dominio. Estas secuencias no serían identificables sólo con una técnica de análisis de dominio como las invariantes de estado extraídas por TIM (Fox and Long, 1998).

En el *Matching Blocksworld* existen las mismas características que en *Blocksworld* pero añadiendo nuevas relativas al dominio. Las propiedades $hand\_positive_1$ y $hand\_negative_1$ están presentes en todas las secuencias dependiendo del literal que defina la polaridad del bloque. Se puede encontrar ciertos detalles interesantes en el dominio. Si una secuencia tiene la propiedad on_2 en el sub-estado de tipo final, nunca pierde la propiedad $solid_1$ durante toda la secuencia, y a su vez no tiene una acción de polaridad intercambiada. Este conocimiento aprendido es relevante para el dominio porque recoge la idea de no equivocarse de polaridad cuando se va a terminar con un bloque encima. El inconveniente reside en que aunque la secuencia no recomiende una polaridad intercambiada al apilar los bloques, la decisión relevante se toma en la acción anterior de levantar (`PICKUP`) o quitar (`UNSTACK`). Esto se debe a que en la secuencias del tipo `Block` no hay información relevante a los brazos. En las secuencias de tipo `Hand` tampoco se recoge esta información porque en estas secuencias no se puede reconocer si los bloques fueron levantados con polaridad intercambiada.

En el *Parking* se puede identificar un conjunto finito de sub-estados de tipo como se hizo con el *Blocksworld*. Estos sub-estados se muestran en la Figura 7.2:

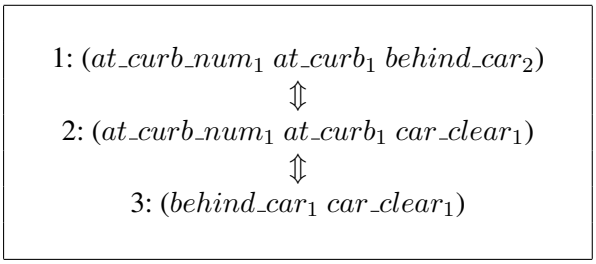


Figura 7.2: Transiciones entre los sub-estados de tipo en el *Parking*.

El dominio tiene la restricción de que sólo dos vehículos pueden estar en una posición de aparcamiento (tipo `Curb`) y como hay una cantidad limitada de ellos, la re-ubicación de los vehículos implica en muchas ocasiones que estos tengan que pasar por posiciones transitorias para liberar otras posiciones. Esta información puede o no estar recogida en las secuencias aprendidas, pero no hay ningún criterio para seleccionar la secuencia correcta que recoja este conocimiento. La huella del plan relajado no lo hace, porque una vez liberada una posición nunca se vuelve a marcar como ocupada en el grafo relajado.

7.1.2. Secuencias de Transporte

En los dominios de transporte, los tipos de los objetos transportables son los que juegan un papel importante en la interpretación de las secuencias. En el *Logistics* los objetos de tipo `Package` intercambian los sub-estados de tipo at_1 e in_1, para indicar que están en una localidad o en un medio de transporte. Estas transiciones forman distintas secuencias que dependen de la combinación de medios que haya que utilizar para llevarlos a su localidad de destino, por ejemplo (camión, avión, camión) o (avión, camión), etc. Por eso, las secuencias de tipo `Package` se suelen diferenciar más por las acciones aplicadas, asociadas a los sub-estados de tipo. El generador aleatorio del *Logistics* crea dos localidades por ciudad, que para el ejemplo se denominarán "aeropuerto" y "correos". Conociendo esta caraterística del dominio se puede deducir las posibles combinaciones que pueden generar diferentes orígenes y destinos de los paquetes. Estas combinaciones se muestran en la Figura 7.3.

1:	aeropuerto ⇒ aeropuerto	[1]
2:	correos → aeropuerto	[4]
3:	aeropuerto → correos	[4]
4:	correos → aeropuerto → aeropuerto	[2]
5:	aeropuerto ⇒ aeropuerto → correos	[5]
6:	correos → aeropuerto ⇒ aeropuerto → correos	[3]

Figura 7.3: Combinaciones de posibles transiciones de los paquetes en el dominio *Logistics*.

Las flechas implican las acciones asociadas de (cargar, *no-op*, descargar); las flechas finas para los camiones y las flechas anchas para los aviones. El número final entre corchetes es el número del caso generado que corresponde a cada combinación. Estos casos se recopilan en el Apéndice D. Estas combinaciones generan en total cinco secuencias que guardan la información necesaria para saber guiar el transporte de los paquetes. Sin embargo, los sub-estados de tipo inicial y final son iguales (ambos tienen at_1), lo que dificulta hacer una correcta recuperación. La huella del plan relajado no es una alternativa efectiva porque sólo diferenciaría tres clases, agrupándolas por longitud. El conocimiento en las secuencias es más rico que el que se puede obtener de las invariantes de estado, que para el tipo `Package` sería ($at_1 \leftrightarrow in_1$). No obstante, este conocimiento necesita de un análisis de dominio adicional para poder crear una clave de recuperación que pueda identificarlo correctamente. Por otro lado, los objetos de tipo `Truck` y `Airplane` se comportan teniendo siempre la propiedad at_1 y la propiedad in_2 cuando tienen uno o más objetos en su interior. La repetición de esta segunda propiedad plantea una limitación al lenguaje de representación del conocimiento, porque se puede inhabilitar

una secuencia, si ésta no da seguimiento al mismo número de objetos que deben ser cargados o descargados del medio de transporte. Dado que los problemas de entrenamiento son más pequeños, en la práctica, la secuencia es útil en los primeros pasos, en los que se carga el número de objetos igual a los representados en la secuencia. Los objetos de tipo `Airport` o `Location` tienen la propiedad in_city_1 y ganan o pierden at_2. Sus secuencias son siempre el mismo sub-estado de tipo con sus respectivas acciones asociadas, pero sin embargo, no guardan información relevante que sirva para conocimiento del dominio.

El *Mystery'*presenta características similares al *Logistics*, pero con el inconveniente añadido del combustible finito que caracteriza al dominio. En este caso los objetos de tipo `Fuel` no son objetos reales como tal, sino más bien una discretización de los niveles de combustible. Las secuencias almacenadas para este tipo guardan las relaciones entre niveles de combustible, pero no aportan conocimiento útil para el seguimiento de las secuencias.

En el *Portcrane*, los objetos transportados son de tipo `Container`. Aquí, las propiedades dry_van_1, $frozen_1$ o $locked_1$ caracterizan la tipología de las secuencias, ya que cada tipo de contenedor debe ser movido con las acciones correctas para evitar perder las propiedades $frozen_1$ y $locked_1$, que son parte de las metas del problema. Esta simplificación muestra que en dominios más reales, la descripción de los objetos por medio de literales del estado puede utilizarse para caracterizar las transiciones que deberían seguirse. Como ejemplo, en un dominio *Logistics* ampliado, se podría marcar a los paquetes especiales para que tengan una propiedad $express_1$, para así diferenciarlos del resto, porque producirán unas secuencias acorde a las intenciones de esa caracterización. Continuando con el *Portcrane*, los objetos de tipo `Crane` o grúas tienen en sus secuencias las propiedades estáticas que marcan las posibles acciones que puedan realizar. Por ejemplo, una grúa con un $reefer\_crane_1$ podrá tener en algún paso una acción asociada `DROP-YARD-FREEZE`. No obstante, presenta el inconveniente por un lado del número de contenedores movidos en la secuencia, y por el otro lado que la clase de contenedor se desconoce por no estar recogido en la secuencia, que sólo guarda información acerca de la grúa.

7.1.3. Secuencias de Ejecución de Tareas

En los dominios de ejecución de tareas los tipos suelen ir ganando propiedades según se vayan ejecutando las tareas de configuración o las tareas definitivas que consiguen las metas. En este sentido, la abstracción de los objetos en este tipo de secuencias recoge bastante bien el orden en que se deben ejecutar las acciones para resolver los problemas. En el *Satellite*, los objetos de tipo `Satellite` pierden la propiedad $power\_avail_1$ al aplicar la acción `SWITCH-ON`, y después mantienen el mismo sub-estado de tipo, salvo en los casos que el instrumento en el satélite haya sido apagado. En cambio, los objetos de tipo `Instrument` ofrecen un poco más de información, porque deben ganar la propiedad $power\_on_1$, y en otra transición posterior la propiedad $calibrated_1$. A partir de aquí y sobre el mismo sub-estado

de tipo, se suceden las acciones TAKE-IMAGE con *no-op*, representando todas las imágenes que debe tomar el instrumento. Aquí, la secuencia tendrá un número limitado de imágenes tomadas, por lo que en este dominio puede ser contraproducente el aprender de problemas muy pequeños, dado que secuencias más grandes asegurarían más imágenes por instrumento y así una capacidad de seguimiento durante más pasos en el plan. Las secuencias de tipo Direction son útiles para reconocer si la dirección debe ser apuntada antes de una acción CALIBRATE. Este tipo de secuencias son fácilmente seleccionadas por la propiedad estática $calibration\_target_1$.

El dominio *Rovers* se comporta de forma similar al *Satellite*. En este caso los objetos de tipo Camera deben tener la propiedad $calibrated_1$ antes de la ejecución de un TAKE-IMAGE. Los de tipo Objective deben ganar la propiedad $have\_image_2$ y después la propiedad $communicated\_image\_data_1$. El resto de tipos producen información equivalente, y al igual que en el *Satellite*, las secuencias están limitadas al número de tareas que se hayan realizado, por lo que un análisis de la generación de ejemplos de entrenamiento debería verificar si es más beneficioso generar ejemplos de problemas más grandes.

7.1.4. Relación de las Secuencias de Tipo con el PDDL

A lo largo del desarrollo de este trabajo se ha resaltado la capacidad que tienen las secuencas de tipo para abstraer los episodios de planificación que típicamente reproduce cada tipo de objeto dentro de un dominio. Por eso puede interpretarse inicialmente, que un dominio con definición de tipos es requisito para el uso de las secuencias de tipo.

La primera versión de PDDL no incluía una definición de tipos para el dominio y por lo tanto los esquemas tanto de predicados como de las acciones podían instanciarse con cualquier objeto del problema, al no estar tipificadas sus variables. No obstante, los tipos del dominio estaban codificados implícitamente porque estos se representaban con predicados que no eran añadidos ni borrados por ninguna acción. Estos predicados se incluían como precondiciones de las acciones que involucraban al tipo de objeto, y de ese modo restringían las instanciaciones válidas de esas acciones. Por ejemplo, la versión del *Logistics* utilizada en la IPC-1 definía predicados adicionales como `(truck ?truck)` para los camiones y `(airplane ?airplane)` para los aviones. El primero se incluía como precondición en las acciones que involucraban a los camiones LOAD-TRUCK, UNLOAD-TRUCK y DRIVE-TRUCK, y el segundo predicado se incluía en las acciones correspondientes, relativas a los aviones. (Koehler and Hoffmann, 1999) mostraron que estos predicados eran equivalentes a la tipificación de las variables, y que en tiempo de pre-proceso previo a la planificación se podían codificar como tipos del dominio.

Por un lado SAYPHI necesita que los esquemas de los predicados estén tipados para poder instanciar los literales a su representación específica de vectores de bits. Sin embargo, esto no significa que el PDDL necesite como tal una jerarquía de tipos. Bastaría con definir un tipo genérico en la forma `(:types object)`

y después declarar todos los objetos del problema de tipo `object`. En esta misma línea, las secuencias de tipo no necesitan de una definición explícita de tipo del dominio. Retomando el ejemplo del *Logistics*, se puede establecer una correspondencia entre los sub-estados de tipo con ambas representaciones. El dominio original también definía un predicado estático `(obj ?obj)` equivalente al tipo `package` del dominio actual. La Figura 7.4 muestra las transiciones de los diferentes sub-estados de tipo que formarían secuencias para un paquete y un camión en el *Logistics*, utilizando las diferentes alternativas de codificación de tipos. Utilizando los tipos explícitos, el paso inicial de la secuencia del paquete coincide con la secuencia del camión, dado que ambos tienen el sub-estado de tipo (at_1). Estas secuencias se pueden diferenciar porque pertenecen a ficheros diferentes, cada uno asociado al tipo del objeto. En la representación implícita de los tipos, existiría un sólo fichero para todas las secuencias, pero las antes citadas se podrían diferenciar porque la secuencia del camión siempre tendrá la propiedad $truck_1$ y la secuencia del paquete siempre tendra la propiedad obj_1.

Truck	Genérico	Acción	Package	Genérico	Acción
(at_1)	$(at_1\ truck_1)$	*estado inicial*	(at_1)	$(at_1\ obj_1)$	*estado inicial*
$(at_1\ in_2)$	$(at_1\ in_2\ truck_1)$	load-truck	(in_1)	$(in_1\ obj_1)$	load-truck
$(at_1\ in_2)$	$(at_1\ in_2\ truck_1)$	drive-truck	(in_1)	$(in_1\ obj_1)$	*no-op*
(at_1)	$(at_1\ truck_1)$	unload-truck	(at_1)	$(at_1\ obj_1)$	unload-truck

Figura 7.4: Correspondencia de transiciones de sub-estados de tipo en el *Logistics* con representación explícita o implícita de los tipos en el dominio.

Algunos dominios tienen predicados estáticos de un argumento que no representan un tipo de objeto en el dominio. Este caso se da cuando no todas las acciones tienen el predicado en las precondiciones. La idea consiste en caracterizar algunos objetos de un tipo sin tener que especializar las acciones para tipos diferentes. Este ejemplo se puede observar en el *Satellite*, donde algunos objetos de tipo `direction` tienen asociado el predicado `calibration_target` (y su correspondiente propiedad). Así, la acción `CALIBRATE` exige como precondición que su dirección sea `calibration_target`, pero la acción `TURN_TO` puede girar los satélites a las diferentes direcciones si importar si tienen la propiedad `calibration_target` o no. Este tipo de propiedades caracterizan muy bien a los objetos, y son de ayuda para diferenciar las secuencias dentro de un mismo fichero de tipo.

Al final se puede resumir la relación que tiene la definición de un dominio PDDL con la capacidad de almacenar conocimiento de las secuencias de tipo en las siguientes ideas.

- Las secuencias de tipo pueden reconocer transiciones típicas de los objetos siempre y cuando haya una codificación explícita o implícita de los tipos del dominio.
- La codificación explícita de los tipos será más beneficiosa para el sistema

CBR porque permite una forma directa de indexar los casos a través de los ficheros que separan las secuencias por tipo. La representación implícita implica que las comparaciones para la medida de similitud se tienen que hacer con todos los casos y no sólo con los casos del tipo correspondiente.

- Las propiedades derivadas de predicados estáticos que no representan un tipo del dominio valen para caracterizar ciertos objetos, y permiten diferenciar las secuencias de un mismo tipo.

7.2. Análisis de Seguimiento de Secuencias

Otro aspecto interesante de analizar, de forma particular para cada dominio, es cómo se produce el seguimiento de las secuencias. Aquí surgen las preguntas: ¿las secuencias de qué tipo se siguen más? ¿qué porcentaje de pasos se utilizan por cada tipo? ¿por qué se dejan de seguir algunas secuencias? En esta sección se intentará dar respuesta a estas preguntas analizando para cada dominio cómo se produce el seguimiento de las secuencias de tipo.

Para este análisis se han tomado los resultados del algoritmo de Escalada+CBR y se ha calculado el seguimiento de las diferentes secuencias que se han seleccionado en cada problema. La elección de este algoritmo está justificada por la selección directa que realiza el algoritmo cuando se puede seguir una secuencia. En el resto de algoritmos los valores de la función heurística pueden influir en que la búsqueda prefiera otro camino, por lo que el porcentaje de seguimiento será igual o menor que los obtenidos con Escalada+CBR. La Tabla 7.1 muestra los resultados obtenidos en el cálculo de seguimiento para cada tipo en los dominios de evaluación.

El cálculo de seguimiento de secuencias se ha realizado sólo sobre los problemas resueltos en el experimento. Para cada problema se calculan:

- *Total de pasos por tipo*: La suma del número de pasos que tienen todas las secuencias recuperadas para un tipo, es decir, todos los objetos del tipo en el problema.

- *Pasos reutilizados por tipo*: La suma del número de pasos utilizados en la recomendación. Este valor se determina a través del último paso al que apunta la secuencia al finalizar el plan.

La división de estos dos valores da un **ratio de seguimiento de tipo** por problema. En los resultados se muestran los valores mínimos y máximos encontrados para estos ratios a lo largo del experimento. El valor global se calcula sobre la suma de todos los pasos recomendados de todas las secuencias recuperadas sobre el total de pasos de dichas secuencias. El valor global no es un promedio de los ratios de seguimiento. Aunque en la práctica este valor suele ser similar, en ocasiones podría verse sesgado dado que los problemas más grandes aportan mayor número de pasos para el conteo. El ratio global muestra una medición genérica sin considerar

Dominio	Tipo	Mínimo	Máximo	Global
Blocksworld	block	66.67	79.41	77.24
Matching-bw	block	20.00	80.00	72.19
	hand	33.33	66.67	63.64
Parking	curb	13.04	63.33	40.37
	car	55.00	70.21	66.71
Logistics	package	25.00	67.50	53.05
	location	37.50	87.76	74.93
	airport	8.33	84.62	55.40
	truck	62.50	75.00	74.56
	city	61.11	66.67	66.47
	airplane	37.50	75.00	71.54
Mystery'	cargo	25.00	75.00	57.41
	vehicle	25.00	50.00	46.67
	location	16.13	87.50	44.04
	space	27.78	82.61	58.45
	fuel	23.08	52.17	42.32
Portcrane	container	18.75	85.71	68.06
	crane	9.38	77.78	23.68
	groundarea	52.38	84.62	74.00
	shiparea	0.00	85.71	63.30
Satellite	direction	10.00	72.09	64.68
	mode	44.44	85.71	75.59
	instrument	0.00	48.65	27.33
	satellite	0.00	61.54	41.83
Rovers	objective	16.67	74.19	57.69
	camera	8.33	53.33	26.86
	waypoint	12.50	56.13	40.21
	store	0.00	41.67	23.46
	rover	3.45	60.00	33.24
	mode	38.89	81.25	65.45
	lander	50.00	75.00	73.72

Tabla 7.1: Porcentajes de seguimiento de las secuencias de tipo.

el tamaño de los problemas. Por otro lado, un alto ratio de seguimiento no implica una buena calidad en los planes obtenidos, porque el último paso que se alcanzó en una secuencia es un indicador que marca hasta dónde se hizo el seguimiento, que no necesariamente se hizo en el mismo número de acciones.

En el *Blocksworld*, el valor mínimo y máximo del ratio de seguimiento no están muy distantes, lo que muestra que la variabilidad de los problemas no afecta en gran medida a la capacidad de ofrecer el conocimiento aprendido. Esto es comprensible dado el número reducido de secuencias que se generan en el dominio. En general, no se obtienen valores más altos porque las secuencias pueden contener transiciones alternativas de pasos intermedios, que luego no se pueden reproducir en el problema que se está resolviendo. Por ejemplo, si un bloque está levantado y no puede colocarse en su destino final, hay que dejarlo sobre la mesa. Pero, existe la alternativa de dejarlo encima de otro bloque (sólo o de una torre bien construida)

sin que esto afecte a la solución final. Si la secuencia recuperada tiene la alternativa de dejar el bloque levantado sobre otro bloque, en lugar de sobre la mesa, no siempre será posible encontrar dicho bloque en el nuevo problema, lo que producirá que esa secuencia no pueda continuarse. Como conclusión a este efecto se puede sacar que el proceso de generación de las secuencias puede refinarse al evaluar la utilidad de las diferentes soluciones (y posteriores secuencias) que pueden generar los problemas de entrenamiento.

En el *Matching Blocksworld*, los valores mínimos de 20.0 para el tipo `Block` y 33.3 para el tipo `Hand` están muy alejados de los valores globales. Esto indica que en algunos problemas el seguimiento de las secuencias es particularmente difícil. Las posibles transiciones en este dominio son limitadas al igual que en *Blocksworld*, por lo que se intuye que a la base de casos le faltan secuencias típicas para esas situaciones. El análisis de generación de las bases de casos, que se realizará más adelante en este capítulo, podría complementar esta conjetura. El *Parking* arroja valores bajos en sus tipos. Además, los valores globales están cercanos a la media de los valores máximo y mínimo. La interpretación en este caso es que la base de casos tiene buena diversidad de ejemplos, pero el efecto de no poder identificar los pasos intermedios, necesarios en la re-ubicación de los coches, hace que algunas secuencias se sigan razonablemente bien y otras no.

En el *Logistics*, las secuencias de tipo `package` tienen un bajo rendimiento, contrario a lo que se debía esperar. La razón principal es la incapacidad de recuperar correctamente las secuencias, según fue explicado en el análisis semántico de las secuencias. Uno de los errores típicos es que la propiedad in_1, que indica que un paquete está dentro de un medio de transporte, no asocia en sí el tipo de medio, sea avión o camión. Así, la secuencia recuperada podría tener asociada en esa transición un `LOAD-TRUCK` (cargar el paquete en un camión) o `LOAD-AIRPLANE` (cargar el paquete en un avión). Esta ambigüedad hace que muchas secuencias no puedan seguirse correctamente. El resto de tipos contienen un conocimiento para el dominio menos relevante. Por ejemplo, las secuencias de tipo `truck` no indican la cantidad de movimientos que tienen que realizar los objetos del nuevo problema. Los seguimientos se producen por la correspondencia que se pueden dar en los pasos iniciales.

En el *Mystery'*, los objetos transportables son de tipo `cargo`. Este tipo tiene una única secuencia de la forma (cargar, *no-op*, descargar), de tal modo que la fase de recuperación no puede tener equivocaciones. Los resultados arrojan un seguimiento imperfecto porque en los problemas generados suelen aparecer objetos que ya están en su localidad de destino, por lo que la secuencia recuperada para ese objeto nunca es seguida durante la búsqueda. Esto plantea una posible mejora al proceso de recuperación, de tal forma que no se incluyan en la tabla de seguimiento objetos que no van a intervenir en el plan. Las secuencias de tipo `vehicle` son difíciles de seguir porque contienen acciones de cargar o descargar que no tienen por qué coincidir en el momento correcto respecto al nuevo problema. Como el resto de los dominios de transporte, el resto de tipos no contienen información relevante que pueda ayudar al proceso de búsqueda.

En el *Portcrane*, los contenedores son los objetos transportables. Las secuencias de tipo `container` son tres, y se recuperan correctamente porque son fácilmente reconocibles por las propiedades que tipifican los diferentes contenedores. Sin embargo, los resultados muestran un seguimiento de un 68.06 %. El seguimiento no es tan bueno como cabría pensar por dos razones. La primera y más evidente es la disponibilidad de las grúas. Por ejemplo, se quiere descargar un contenedor con propiedad $reefer\_crane_1$ y la grúa que puede hacer el `UNLOAD-SHIP-FREEZE` no está en el área del barco. En cambio, si se encuentra una grúa que puede hacer una descarga normal con `UNLOAD-SHIP`, sólo aparecerá como acción aplicable la segunda acción, por lo que no se aceptará la recomendación CBR y se habrá realizado otra transición que inhabilita la secuencia. La segunda razón es por la interferencia de las recomendaciones de otros objetos. Por ejemplo, la Tabla 7.5 muestra un ejemplo de un estado inicial, donde dos de sus acciones aplicables se muestran con sus respectivos valores de la función r y las secuencias que han aportado a estos valores. El `CONTAINER1` tiene la propiedad $reefer\_crane_1$ y su secuencia está bien recuperada. Sin embargo, las secuencias de los objetos `PORTAINER1` y `SHIP` cumplen con el sub-estado de tipo que relacionan, con el inconveniente de que tienen asociado el operador de la segunda acción. Se asume pues, que la recomendación de dos secuencias incorrectas interfiere en la recomendación de la secuencia que es correcta.

Acciones	ratio	Secuencias
(UNLOAD-SHIP PORTAINER1 CONTAINER1 SHIP)	2/3	PORTAINER1, SHIP
(UNLOAD-SHIP-FREEZE PORTAINER1 CONTAINER1 SHIP)	1/3	CONTAINER1

Figura 7.5: Ejemplo de acciones en las que se produce interferencia en el seguimiento de secuencias.

El ejemplo mostrado no quiere indicar que las secuencias de tipo `crane` y `shiparea` estén mal recuperadas, sino que no hay forma de asociar las acciones correctas. Esto sucede por ejemplo con los medios de transporte, donde las acciones de cargar, mover o descargar no tienen porqué seguir el orden que se produjo en el problema original. En este sentido, el porcentaje de acierto del resto de tipos depende más de la casualidad que se ha comentado en los otros dominios de transporte.

En el *Satellite*, las secuencias de tipo `satellite` tienen bajo porcentaje de seguimiento porque pueden fallar en el orden de las transiciones. Por ejemplo, hacer `TURN-TO` y luego `SWITCH-ON`, es equivalente a hacerlo a la inversa, pero si el resto de recomendaciones (de los otros objetos) hacen elegir la acción asociada al revés, dicha secuencia no podrá seguirse. En cambio, las secuencias de tipo `instrument` se suelen reproducir mejor, porque primero ganan la propiedad $power\_on_1$ con la acción `SWITCH-ON` y después la intercambian por la propiedad $calibrated_1$ con la acción `CALIBRATE`, para después encadenar *no-ops* con pasos del mismo sub-estado de tipo junto a acciones `TAKE-IMAGE`. Su bajo porcentaje

se debe a que en los problemas hay instrumentos que recuperan alguna secuencia, pero que luego no son utilizados en el problema porque se utilizan otros. Las secuencias de tipo `mode` se siguen bastante bien porque igualmente intercalan las acciones de *no-op* con la transición que gana la propiedad $have\_image_1$ al aplicar la acción `TAKE-IMAGE`. El inconveniente en estas secuencias radica en que la recuperación de la secuencia más corta, cuando hay empates en la medida de similitud, no beneficia a los problemas más grandes, que ven terminadas las secuencias de este tipo sin poder ofrecer iterativamente la misma recomendación. Una alternativa es que un ranking de las secuencias valore mejor las más largas de este tipo o de forma más genérica buscar una representación que reconozca estos ciclos para poder aplicarlos en los nuevos problemas.

El *Rovers* produce efectos parecidos a los del *Satellite*. Las secuencias del tipo `rover` serían homólogas de las de tipo `satellite`; las de tipo `camera` a las de tipo `instrument`. El tipo `mode` es un poco diferente porque incluye una transición adicional en la que se obtiene la propiedad $communicated\_image\_data_1$ al aplicar la acción de enviar la imagen recolectada. Entre los tipos adicionales también está el tipo `lander` cuyas secuencias se pueden seguir fácilmente porque es una sucesión de acciones de enviar las muestras o imágenes recolectadas, con acciones como `COMMUNICATE_ROCK_DATA` o `COMMUNICATE_IMAGE_DATA`. En las secuencias de tipo `lander` se revela la importancia de las acciones asociadas a los sub-estados de tipo, ya que el sub-estado de tipo no cambia durante toda la secuencia. La forma de diferenciar estas secuencias es mediante la sucesión de las acciones asociadas. En general, las recomendaciones en el *Rovers* se ven afectadas por las longitudes de las secuencias recuperadas, por lo que al igual que en el *Satellite*, secuencias más largas serían más provechosas en los problemas más grandes. Al menos habría que estudiar la posibilidad de recuperar las secuencias en función de la equivalencia en dificultad del problema actual.

7.3. Análisis de Valoración

En esta sección se presenta una técnica para valorar las diferentes secuencias de tipo aprendidas y unos experimentos adicionales que mostrarán en qué medida la valoración de las secuencias puede utilizarse para mejorar el rendimiento del conocimiento aprendido. Como se explicó en el proceso de recuperación, se prefirió un esquema sencillo que recuperara las secuencias de forma rápida. El criterio para decidir entre dos secuencias con igual medida de similitud fue tomar la secuencia más corta, sin que esta decisión estuviera realmente fundada. En este sentido surge la pregunta: ¿cuál secuencia sería más útil recuperar? Adicionalmente, se mostró que en la fase de reutilización, se pueden recibir recomendaciones de diferentes secuencias, por lo que también tiene sentido preguntar ¿cuál paso de qué secuencia es mejor seguir? Esta pregunta surge partiendo de la base que seguir algunas transiciones es más provechoso para la búsqueda que seguir otras. Por eso, el coste computacional que no se decidió asumir en una medida de similitud,

plantea un problema de utilidad de los casos para las situaciones en que se producen empates, tanto en la fase de recuperación como en la de seguimiento. Hacer una correcta valoración de estas situaciones permitiría obtener un mejor rendimiento en el proceso de planificación que si las situaciones se resuelven de forma arbitraria.

La valoración de los pasos de las secuencias tiene sentido en algoritmos como EHC en donde la estrategia de ordenación puede producir la situación de empate explicada anteriormente, cuando dos secuencias dan como estados recomendados dos sucesores diferentes. Conceptualmente, se entiende el valor de un paso en una secuencia de tipo como su capacidad para dar buenas recomendaciones. Por tanto, un paso tuvo un acierto en recomendación si el estado recomendado es parte de la solución del plan final. En este mismo sentido, el número de intentos de recomendación es el total de veces que una recomendación ha sido dada para un estado, y entonces, que la valoración del paso n en la secuencia $\mathcal{Q}$ está expresada como el factor (*aciertos* / *intentos*) de dicho paso. La valoración de la secuencia será la frecuencia de acierto calculada de forma global y viene dada por la suma de todas las recomendaciones marcadas como acierto entre la suma de los intentos realizados. Estas valoraciones son fácilmente calculables a partir de un plan, simplemente guardando una traza de los pasos de las secuencias recomendados y utilizados en cada nodo.

7.3.1. Aprendizaje de las Valoraciones

Para poder sacar provecho de las valoraciones de las secuencias es necesario determinar dichas valoraciones con respecto a un algoritmo específico. Las valoraciones pueden depender del algoritmo de búsqueda, dado que estos se aprovechan de la recomendación CBR de diferentes formas. En cualquier caso, estas valoraciones se deben determinar después de la generación de la base de casos. Por eso es necesario una fase posterior que permita el aprendizaje de las valoraciones, a la que se denomina fase de validación, que es equivalente a la realizada por otros algoritmos para el análisis de utilidad. Esta fase debe ser anterior a una fase de prueba o evaluación como las realizadas en los experimentos del capítulo 6. Las situaciones de empate suelen sucederse indistintamente en los problemas por lo que hay que aprender las valoraciones de forma incremental durante esta fase. Aquí se presenta el dilema de la exploración vs. la explotación. Por un lado es interesante utilizar las secuencias que se han reconocido como útiles hasta el momento y por otro lado interesa explorar nuevas secuencias de las que no se ha establecido su valoración. Este dilema, típico del aprendizaje por refuerzo, suele resolverse de varias formas, que si se traslada al enfoque CBR de este trabajo se pueden resumir las más populares en:

- Enfoque ε-*greedy*: Con una probabilidad ε se selecciona una secuencia aleatoria para deshacer la situación de empate (exploración), y con probabilidad (1 - ε) se elige la secuencia de mayor valoración (explotación).

- Enfoque naïve: Se evalúan las secuencias un número fijo de veces y a partir

de ahí se escoge la secuencia con mejor valoración. En la práctica esta idea se implementa con modelo de actualización de valoraciones (en un rango [0,1]), tales como el algoritmo R-MAX (Brafman and Tennenholtz, 2002) que asumen que en principio todas las secuencias tienen valoración 1.

La ventaja del segundo enfoque radica en que dadas unas pocas exploraciones ya se puede tener un modelo de las valoraciones para irlas utilizando, y no caer en la aleatoriedad del ε-*greedy* cuando ya se tiene claro que una secuencia es mejor que otra. Los modelos de actualización de valoraciones sólo tienen que partir del supuesto de que todas las opciones son buenas, e ir estimando la valoración real a medida que se van viendo más ejemplos. Este enfoque está motivado por la ventaja que permite desde el inicio tomar la secuencia de mejor valoración. En este sentido, se va a modificar la forma de calcular las valoraciones, para poder reproducir un esquema simple del enfoque naïve. Si se representan los aciertos de un paso i de una secuencia $\mathcal{Q}$ como la función $\lambda(\mathcal{Q}, i)$ y el número de intentos como la función $E(\mathcal{Q}, i)$, se puede escribir el cálculo de la valoración $\mathcal{V}_{\mathrm{P}}$ de un paso como:

$$\mathcal{V}_{\mathrm{P}}(\mathcal{Q}, i) = \frac{\lambda(\mathcal{Q}, i) + C}{E(\mathcal{Q}, i) + C} \tag{7.1}$$

donde C es una constante positiva que permite dar el valor $C/C = 1$ para los pasos desconocidos. Valores más altos de C penalizarán menos la actualización de las valoraciones cuando las recomendaciones no sean acertadas. Por ejemplo, si una recomendación es errónea, en las subsiguientes situaciones de empate, se volverá a seleccionar ese paso sólo cuando el resto de opciones tengan una valoración menor de $C/(C + 1)$. Del mismo modo se modifica la forma de cálculo de la valoración de una secuencia para obtener la valoración global $\mathcal{V}$ que ahora viene dada por la fórmula

$$\mathcal{V}(\mathcal{Q}) = \frac{\sum_{i=0}^{n} \lambda(\mathcal{Q}, i) + C}{\sum_{i=0}^{n} E(\mathcal{Q}, i) + C} \tag{7.2}$$

donde n es la longitud de la secuencia $\mathcal{Q}$. En la fase de aprendizaje de las valoraciones, cuando se presentan situaciones de empate, tanto los pasos como las secuencias se ordenan por este cálculo de valoraciones y se selecciona el mejor. Si las valoraciones son iguales el empate no puede resolverse y se selecciona una arbitrariamente.

7.3.2. Evaluación de Valoraciones

El diseño de experimentos para la evaluación de las valoraciones de las secuencias consiste en reutilizar los experimentos planteados en el Capitulo 6 y añadir una prueba adicional del algoritmo CBR que utilice dichas valoraciones aprendidas durante una fase de validación. Cada algoritmo CBR se utiliza para aprender las valoraciones respecto al propio algoritmo. De esta forma cada experimento consiste en:

- **Algoritmo Base**: El algoritmo original. Ej. Escalada, EHC, etc.
- **Algoritmo+CBR**: El algoritmo modificado para seguir las recomendaciones CBR.
- **Algoritmo+CBR-V**: El algoritmo CBR que utiliza las valoraciones aprendidas en la fase de validación.

La Tabla 7.2 muestra los resultados del algoritmo Escalada+CBR utilizando las valoraciones de los casos que fueron aprendidas en la fase de validación. Los detalles de las columnas significan lo siguiente. La columna (Δ Resueltos) muestra la diferencia de problemas resueltos respecto al algoritmo CBR sin utilizar las valoraciones, tal como se mostró en la evaluación de los algoritmos en el Capítulo 6. Valores negativos significarían que el aprendizaje de las valoraciones produce peores resultados para el dominio. La columna (Δ Tiempo Max.) muestra la diferencia de los tiempos máximos necesitados para resolver la misma cantidad de problemas, y por último la columna (Δ Longitud Max.) refleja la diferencia de la longitud máxima de los planes registrada en la resolución de la misma cantidad de problemas. Tanto para el tiempo como longitud máxima, valores negativos significarían que los valores máximos eran menores en los experimentos sin utilizar las valoraciones.

Dominio	Δ Resueltos	Δ Tiempo Max.	Δ Longitud Max.
Blocksworld	-1	-67.03	-1210
Matching-bw	+1	+76.98	-128
Parking	-1	-106.05	-80
Logistics	+5	+313.02	+1737
Mystery'	0	-556.66	-53
Portcrane	-1	-275.39	-989
Satellite	+4	+124.76	+31
Rovers	0	-32.12	+12

Tabla 7.2: Resultado de Escalada + CBR con aprendizaje de utilidades de casos.

En el *Blocksworld*, Escalada+CBR-V resuelve un problema menos que Escalada+CBR, pero dos más que Escalada. La longitud y el tiempo que se obtienen con el algoritmo CBR y su variante con valoraciones varían de un problema a otro y no sistemáticamente. Dado que en el *Blocksworld* son pocas secuencias, las situaciones de empate en el proceso de recuperación se dan con poca frecuencia. Las recuperaciones alternativas que se dan por las valoraciones de las secuencias no afectan significativamente el comportamiento de Escalada+CBR. La gran diferencia entre la longitud máxima (-1210) se debe a que Escalada obtiene muy mala calidad en los problemas grandes, y pueden llegarse a encontrar este tipo de diferencias. De igual forma, Escalada+CBR-V, no obtiene resultados significativamente diferentes en los dominios *Matching Blocksworld* y *Parking*.

En el *Logistics* no se realizaba una correcta recuperación de las secuencias de tipo `package`. A todos los objetos se le asignaba el caso número cuatro, cor-

respondiente a las transiciones dos y tres de la Tabla 7.3, explicadas en el análisis semántico. Al utilizar las valoraciones siempre se asigna el caso número 5, que corresponde a la transición 5: (aeropuerto ⇒ aeropuerto → correos). Esta transición cubre la transición 1 y parcialmente la 6, lo que representa una mayor probabilidad de seguir las secuencias. En el tipo `truck` se eligen por ejemplo, secuencias que tienen asociadas dos acciones de cargar, un *no-op* y luego dos de descargar. Estas diferencias en la recuperación hacen que Escalada+CBR-V se comporte mejor que la versión sin valoraciones. Esta mejora se aprecia en la distribución acumulada de tiempo mostrada en la Figura 7.6, que en los problemas medianos y grandes llega a un orden de magnitud. Se puede interpretar que ante la no posibilidad de distinguir entre las secuencias de un tipo, la valoración de las mismas puede encontrar una secuencia que globalmente beneficie al proceso de seguimiento.

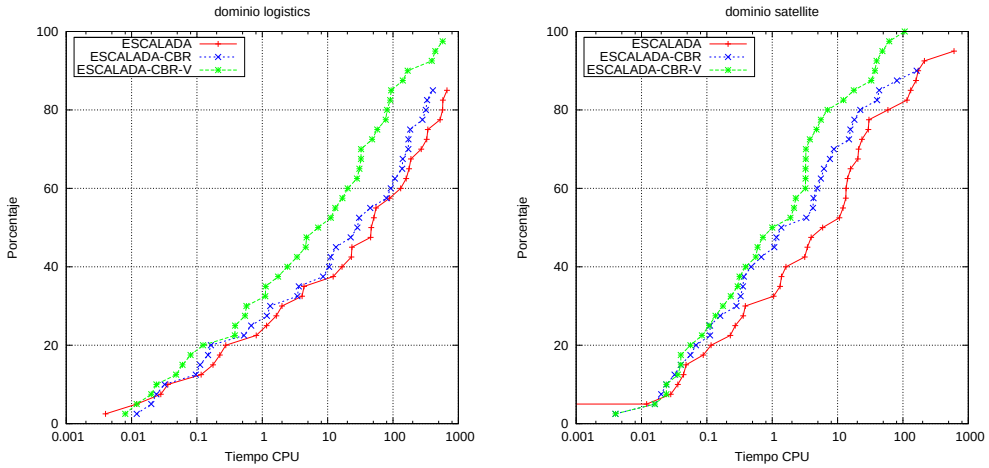


Figura 7.6: Distribución acumulada de tiempo de CPU para experimentos con Escalada CBR utilizando la valoración de las secuencias (Escalada+CBR-V).

En el *Portcrane* se dan muy poco las situaciones de empate a la hora de la recuperación. Por ejemplo, los contenedores suelen estar muy diferenciados en las propiedades de los estados inicial y final. Para el resto de tipos, las diferencias en la recuperación no repercuten considerablemente. Los resultados, aunque varíen de un problema a otro, no son realmente significativos. De igual forma, en el *Mystery'* se resuelven los mismos problemas, sólo con diferencias en tiempo y calidad entre unos problemas y otros.

En el *Satellite* se observa una mejora en la distribución acumulada de tiempo (Figura 7.6). La explicación radica en que el proceso de valoración hace preferir secuencias más largas en los tipos `instrument` y `mode`, lo que permite encadenar iterativamente las diferentes tomas de imágenes. En cambio en el *Rovers*, esta diferencia del tamaño de las secuencias no afecta el rendimiento de la búsqueda, porque tanto Escalada+CBR como Escalada+CBR-V resuelven los mismos problemas, con pequeñas variaciones en las mediciones de tiempo o longitud del plan

entre los distintos problemas. En este dominio, la mayor diversidad de tipos de objetos hace por ejemplo que un `rover` tenga que hacer varias actividades (tomar ejemplos de rocas, de suelo, tomar imágenes) y no sea estrictamente necesario hacer el encadenamiento iterativo de actividades como se da en el *Satellite*. Además, las secuencias de tipo `waypoint` y `objective` tienen ya asociadas las acciones que debe ejecutar el `rover`, por lo que el seguimiento, si se ve interrumpido en una secuencia, puede también continuarse por otras.

La Tabla 7.3 muestra la diferencia de resultados entre los algoritmos EHC+CBR y EHC+CBR-V. Las diferencias son más discretas que con el algoritmo de Escalada+CBR, pero en general, EHC+CBR-V mejora en tiempo y longitud a EHC+CBR. Las más importantes a destacar son la mejora en el tiempo máximo y en longitud máxima conseguidas en el *Satellite*, en el que ya EHC+CBR arrojaba buenos resultados. La elección de las secuencias de mayor longitud permite que las mejoras obtenidas por EHC+CBR sean aún más significativas con EHC+CBR-V.

Dominio	Δ Resueltos	Δ Tiempo Max.	Δ Longitud Max.
Blocksworld	0	+235.95	+2
Matching-bw	+2	+0.25	+10
Parking	0	-58.36	+12
Logistics	0	+6.66	-7
Mystery'	+1	+1.80	+19
Portcrane	0	+106.73	+2
Satellite	+1	+332.48	+44
Rovers	0	+13.73	-20

Tabla 7.3: Resultados de EHC + CBR con aprendizaje de las valoraciones de las secuencias.

7.4. Análisis de Generación de la Base de Casos

En esta sección se analiza el proceso de generación de las bases de casos en los dominios de evaluación. De forma empírica se ha visto, que después de generar unos pocos casos a partir de problemas resueltos, el sistema de recomendación CBR puede comportarse de forma razonable. Las preguntas para esta situación son: ¿hasta qué punto interesa seguir aprendiendo nuevas secuencias? ¿cuántas secuencias son suficientes para cada dominio? Preguntas de este tipo, dentro del ámbito del aprendizaje de conocimiento de control, se conocen como el problema de la utilidad (Minton, 1988). Referido al razonamiento basado en casos, el problema de la utilidad implica que hay que determinar qué ganancia, respecto a una métrica de evaluación, se obtiene por cada nuevo caso que se almacena en la base de casos. Así, para hacer una evaluación de las utilidades de los casos se suele calcular una curva de aprendizaje que muestre el rendimiento de un sistema en función del tamaño de la base de casos. Además, la utilidad de los casos es dependiente del algoritmo que se utilice. Por ejemplo, la ganancia que se puede obtener en el

rendimiento de Escalada+CBR al almacenar un caso más, no es necesariamente la ganancia que obtiene EHC+CBR que es menos dependiente del conocimiento que aportan las secuencias.

Para esta evaluación se ha decidido calcular las curvas de aprendizaje para los algoritmos Escalada+CBR y EHC+CBR. Se han generado un conjunto nuevo de entrenamiento, el doble de grande que el utilizado en los experimentos del Capítulo 6. En total cada conjunto de entrenamiento tiene 40 problemas. Las características de estos problemas son similares al conjunto original, que fueron mostradas en la Tabla 6.1. Simplemente, en la generación por los distintos grupos de dificultad, se les pidió a los generadores aleatorios que sacaran el doble de problemas. El conjunto de prueba se mantiene igual que en el resto de experimentos. El proceso de evaluación es como sigue: Se realiza una iteración con la base de casos vacía para intentar resolver los problemas sin conocimiento. A continuación se realizan cuarenta iteraciones, una por cada problema de entrenamiento. En la iteración i, primero se crea una base de casos con los i primeros problemas de entrenamiento. Después se resuelve el conjunto de prueba y se almacenan los resultados para dicha iteración. Al final se determina:

1. El número de casos que hay en la base de casos en cada iteración
2. El número de problemas resueltos del conjunto de prueba en cada iteración
3. Los tiempos, longitudes del plan y nodos evaluados acumulados de los problemas resueltos en todas las iteraciones.

Por la gran cantidad de ejecuciones que se debían realizar en el experimento, se redujeron los tiempos límite de evaluación y de prueba a 10 segundos. Estas evaluaciones son orientativas, tomando en cuenta que ya se tiene una idea del comportamiento real de los algoritmos en cada dominio usando un límite de tiempo considerable. Sobre los resultados de las iteraciones es importante mostrar varios puntos para sacar posibles conclusiones. Estos son:

- Número de problemas resueltos en la iteración 0: Para comparar con el resto de iteraciones, por si en algún caso el comportamiento resulta peor que con la base de datos vacía.
- Número de problemas resueltos en la iteración 40: Para determinar si hay algún tipo de saturación de la base de casos que empeore el rendimiento respecto al mejor resultado.
- Mínimo número de problemas resueltos en cualquier iteración: Para determinar la peor iteración de todas. Si este valor corresponde a la iteración 0 (con la base de datos vacía), se entiende que alguna iteración ha mejorado el comportamiento. De lo contrario alguna iteración empeora el comportamiento de la iteración 0. En caso de empate, interesa la mayor iteración, para poder descubrir este efecto.

- Máximo número de problemas resueltos: Para determinar la ganancia respecto a la base de datos vacía. En caso de empate, interesa saber la mínima iteración, para calcular el mejor rendimiento con el menor número de casos.

La Tabla 7.2 muestra los puntos claves en la evaluación que se realizó para determinar la utilidad de las secuencias en el algoritmo Escalada+CBR. Los valores entre paréntesis indican la iteración en la que se ha conseguido el valor máximo o mínimo. Todos los valores mínimos coinciden con la iteración 0, pero en el *Parking* y en el *Matching Blocksworld* se encuentran bases de casos en diferentes iteraciones que reproducen el mismo comportamiento. Resulta de interés el hecho de que en varios dominios los valores máximos se obtienen en las primeras iteraciones. Por ejemplo, el *Blocksworld*, el *Parking* y el *Satellite* lo consiguen en la iteración uno. Esto es un indicativo de que el conocimiento que aportan las secuencias ya se encuentra en los primeros casos almacenados.

Por otro lado, el límite de 10 segundos dificulta tener una franja más amplia en donde se puedan apreciar diferencias reales en términos de problemas resueltos. No obstante, cabe mencionar que en el *Satellite* la diferencia de tener o no conocimiento se refleja en una diferencia de 11 problemas y en el *Mystery'* en una diferencia de diez.

Dominio	Iter.0	Iter.40	Mínimo	Máximo
Blocksworld	26	30	26 (0)	31 (1)
Matching-bw	7	9	7 (14)	9 (10)
Parking	21	22	21 (29)	23 (1)
Logistics	14	15	14 (0)	16 (2)
Mystery'	8	11	8 (0)	18 (12)
Portcrane	16	20	16 (0)	20 (3)
Satellite	21	28	21 (0)	32 (1)
Rovers	19	22	19 (0)	24 (27)

Tabla 7.4: Problemas resueltos en las iteraciones del análisis de la generación de las bases para el algoritmo Escalada+CBR.

La Tabla C.2 del Apéndice C muestra el detalle del número de casos aprendidos en cada iteración. En el *Blocksworld* se ha podido comprobar que hay un número reducido de secuencias, tal como fue deducido en el análisis semántico de las secuencias. A partir de la iteración 32 la base de casos queda estabilizada a 28 secuencias diferentes. Hay un total de 21 problemas resueltos en todas las iteraciones. De estos problemas el menor tiempo acumulado fue conseguido en la segunda iteración, la misma que consiguió el máximo número de problemas resueltos, teniendo sólo cuatro secuencias en la base de casos. La mejor calidad de los planes se obtuvo en la iteración 5 con 11 casos. Los tiempos acumuladas se estabilizan después de la iteración 13 cuando había 22 casos. Estos valores estables, sin ser los mejores, están bastante cercanos a ellos, como puede apreciarse en la Figura 7.7.

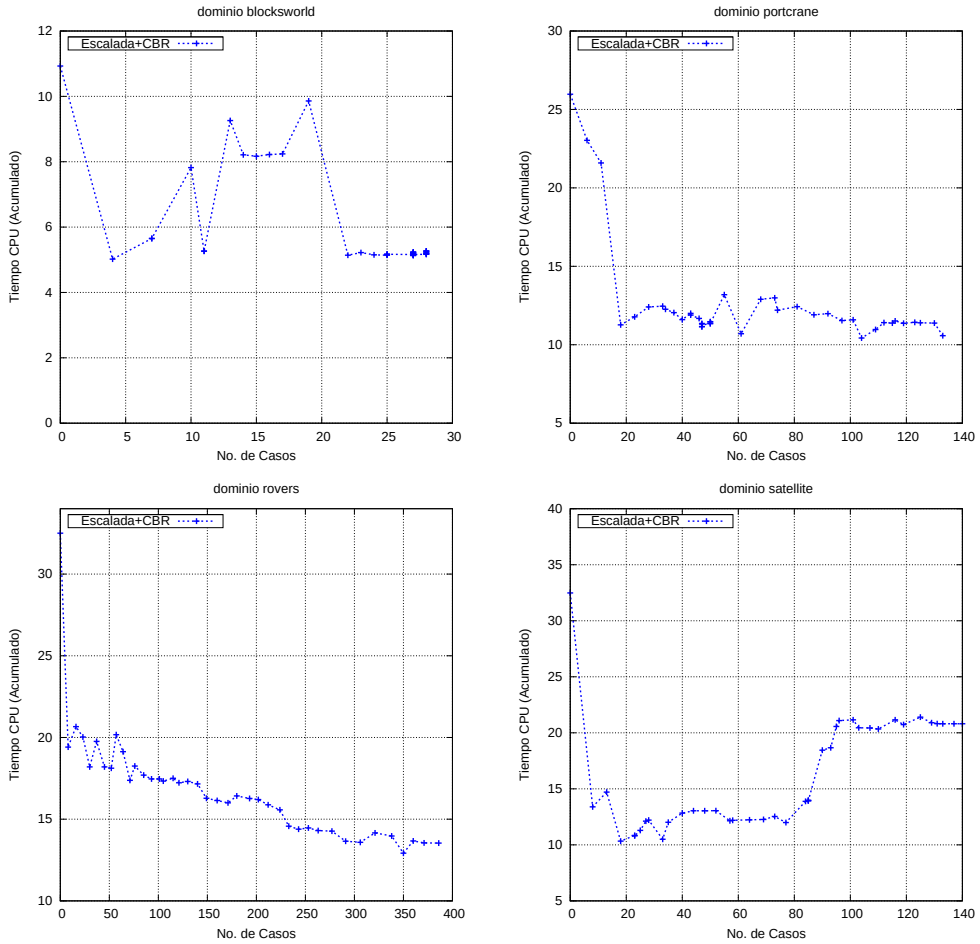


Figura 7.7: Tiempo acumulado de Escalada+CBR en resolver los problemas durante las iteraciones de generación de las bases de casos.

El *Matching Blocksworld* no tiene un número acotado de casos, porque las secuencias de tipo `hand` son de longitud variable según el tamaño del problema, determinado por el número de bloques. Adicionalmente, las secuencias de tipo `block` aumenta por la diferenciación de bloques positivos y negativos, y por las alternativas que conlleva operar con cualquier brazo el bloque superior de cada torre (como no lleva otro bloque encima puede perder la propiedad $solid_1$). Sólo 7 problemas fueron resueltos en todas las iteraciones, por lo que el tiempo acumulado no es muy representativo para analizar la mejor iteración. Ya se ha comentado que Escalada(+CBR) no son algoritmos apropiados para resolver este dominio.

El *Parking* arroja poca variabilidad en términos de problemas resueltos. Todas las iteraciones están en el rango entre 21 y 23 problemas. El número de casos va creciendo por cada iteración hasta llegar a 142 en la iteración 40. Como se explicó previamente, las transiciones intermedias para la re-ubicación de los coches

aumentan con el tamaño de los problemas, por lo que se puede intuir que es una base de casos que siempre va a crecer. El tiempo acumulado, a pesar de las fluctuaciones, muestra que a partir de 66 secuencias no se encuentra un mejor rendimiento, posiblemente debido al coste computacional que supone ir teneniendo un base de casos más grande.

En el *Logistics*, la base de casos crece indefinidamente debido a las nuevas secuencias que aparecen en función del tamaño del problema, generadas para los medios de transporte de tipo `truck` y `airplane`. Sin embargo, resulta de interés ver que en el tipo `package` están bien definidas las cinco secuencias explicadas en el análisis semántico. Las secuencias detalladas se encuentran en el Apéndice D. De los 14 problemas resueltos en todas las iteraciones, se obtiene el menor tiempo en la iteración 9 con 47 secuencias en la base de casos. Al igual que en el *Logistics*, en el *Mystery'* la base de casos crece indefinidamente. Dado que no hay restricciones de tipos de medios de transporte para transportar los objetos de un sitio a otro, hay una única secuencia con las acciones asociadas (`LOAD`,*no-op*, `UNLOAD`). Para el resto de tipos el tamaño del problema influye en su longitud. Sólo se resuelven siete problemas en todas las iteraciones, por lo que la cierta estabilidad encontrada a partir de los 106 casos en la iteración 17 debe tomarse únicamente como referencia. Más relevante resultan los 18 problemas resueltos obtenidos en la iteración 12. El *Portcrane* también incrementa el tamaño de su base de casos en cada iteración. El tipo de objeto transportable en este dominio, el `container`, tiene 3 secuencias diferentes. Estas no se diferencian por las restricciones a los medios de transporte, que en este caso son iguales para todas. La diferencia reside en la tipología de los contenedores que está codificada en los diferentes predicados por los que el sub-estado inicial y final de tipo son diferenciados. Estas tres secuencias están detalladas también en el Apéndice D. Tanto el número de problemas resueltos como el tiempo acumulado para resolver los 16 problemas comunes, encuentran cierta estabilidad con mínimas fluctuaciones a partir de la tercera iteración cuando la base de casos tenía en total 18 secuencias. Este efecto es apreciable en la Figura 7.7 en donde además no se perciben indicios de saturación hasta ese momento.

El *Rovers* muestra en su curva de aprendizaje una tendencia a ir reduciendo el tiempo acumulado de los 18 problemas resueltos por todas las iteraciones. Esto indica que Escalada+CBR sigue ganando tiempo aún con los 386 casos almacenados en la iteración 40. Dado que los problemas de entrenamiento están ordenados por dificultad, las nuevas secuencias en este tipo de dominios serán más largas, lo que explica que haya un mayor seguimiento y por tanto una mejora en ahorro de evaluaciones y en tiempo de CPU. Este efecto se verifica además en el número de problemas resueltos que encuentra el mejor rendimiento en iteraciones altas, a diferencia de otros dominios. El *Satellite* por su parte, sí encuentra un punto de inflexión en el que la base de casos comienza a degradar su comportamiento. A partir de la iteración 13, con 48 casos, se observan fluctuaciones que tienen tendencia a aumentar el tiempo acumulado de los 21 problemas resueltos en todas las iteraciones. Intuitivamente se asume que la diferencia con el *Rovers* es que éste tiene más variabilidad en sus posibles secuencias por tener más tipos y más operadores

en el dominio. La comparación puede apreciarse visualmente en la Figura 7.7. Las curvas de aprendizaje del resto de dominios se encuentran en la Figura C.8 de la sección de evaluaciones complementarias del Apéndice C.

La Tabla 7.5 muestra los puntos importantes sobre los problemas resueltos en las iteraciones del experimento que determina la utilidad de las secuencias para el algoritmo EHC+CBR. Como este algoritmo es menos dependiente de las recomendaciones, queda de manifiesto que la ordenación de los nodos para su evaluación puede en algunos casos perjudicar el proceso de planificación, aún asumiendo que el conocimiento de las secuencias es el correcto. Resalta el deterioro en el *Matching Blocksworld* y en el *Logistics* tras la primera iteración o la situación invariable en el *Portcrane*, que consiguió 16 problemas resueltos en todas sus iteraciones. Positivamente se puede destacar la mejora en el *Satellite* que al igual que en Escalada+CBR, consigue el mejor resultado en la primera iteración. Resultados como el del *Mystery'*, con el peor resultado en la iteración 10 y el mejor resultado en la iteración 11, revelan que para algunos dominios el tamaño de la base de casos no es tan relevante como el contenido en sí de las secuencias.

Dominio	Iter.0	Iter.40	Mínimo	Máximo
Blocksworld	14	14	14 (40)	17 (2)
Matching-bw	6	5	2 (1)	6 (0)
Parking	20	20	17 (3)	22 (21)
Logistics	24	23	20 (1)	24 (0)
Mystery'	19	22	18 (10)	23 (11)
Portcrane	16	16	16 (40)	16 (0)
Satellite	23	29	23 (0)	32 (14)
Rovers	20	21	20 (32)	22 (11)

Tabla 7.5: Problemas resueltos en las iteraciones del análisis de la generación de las bases para el algoritmo EHC+CBR.

El número de casos generados en este experimento es idéntico a los casos generados en las iteraciones realizadas con Escalada+CBR (Tabla C.2). Aquí sólo cambia el número de problemas resueltos y el tiempo acumulado para resolver los problemas comunes, dado que se cambia el algoritmo de evaluación. En general, las curvas de aprendizaje no aportan mucho más información que la arrojada por Escalada+CBR, salvo en los casos en que el comportamiento de EHC y Escalada sea muy diferente. Por ejemplo, así como EHC+CBR no era una buena alternativa para el *Blocksworld*, se puede apreciar que en el momento en que se estabiliza la base de casos, los valores de tiempo acumulado suelen ser más altos que los de las primeras iteraciones, contrario a lo que sucedió en Escalada+CBR. Por el contrario, en el *Satellite*, donde EHC+CBR sí aprovecha el conocimiento de las secuencias, se puede encontrar una estabilidad con un buen tiempo acumulado desde las primeras iteraciones, al igual que Escalada+CBR.

7.5. Aprendizaje en Línea de Secuencias de Tipo

En esta sección se presenta la aplicación de las técnicas CBR en un proceso continuo de aprendizaje, llamado comúnmente aprendizaje en línea (*on-line*). El aprendizaje vago, en el que entran las técnicas CBR, se caracteriza porque no necesita tener un modelo completo del conocimiento, generado normalmente en una fase de entrenamiento. Una de sus ventajas es poder aportar el conocimiento aprendido hasta el momento. De ahí viene la justificación del ciclo constante de CBR de recuperar → reutilizar → almacenar.

Con la evaluación del aprendizaje en línea se quiere determinar si es posible ir mejorando el rendimiento de los diferentes problemas a medida que se va aprendiendo sobre el mismo conjunto de prueba. Por eso se ha diseñado un nuevo experimento, que partiendo de una base de casos vacía, consiste en el siguiente esquema:

1. Resolver el primer problema de prueba.
2. Generar las secuencias de tipo a partir del problema resuelto.
3. Recuperar las secuencias para el siguiente problema. La base de casos tiene las secuencias de todos los problemas resueltos anteriormente.
4. Resolver el problema utilizando las secuencias recuperadas.
5. Continuar con el paso 2.

Se ha mantenido el límite de tiempo de 900 segundos y se ha utilizado el mismo conjunto de prueba de los experimentos presentados en el Capitulo 6, de modo que se puedan contrastar con los resultados previos. Se ha realizado pruebas con los algoritmos Escalada+CBR y EHC+CBR. La Tabla 7.6 muestra los resultados de Escalada+CBR juntos con los nuevos resultados en los que el algoritmo hace aprendizaje en línea.

Dominio	Escalada		Escalada+CBR		On-line Escalada+CBR	
	Resueltos	%	Resueltos	%	Resueltos	%
Blocksworld	34	85.0 %	37	92.5 %	**39**	**97.5 %**
Matching-bw	11	27.5 %	11	27.5 %	**12**	**30.0 %**
Parking	38	95.0 %	39	97.5 %	**40**	**100.0 %**
Logistics	**34**	**85.0 %**	**34**	**85.0 %**	**34**	**85.0 %**
Mystery'	11	27.5 %	**15**	**37.5 %**	13	32.0 %
Portcrane	37	92.5 %	38	95.0 %	**40**	**100.0 %**
Satellite	**38**	**95.0 %**	36	90.0 %	**38**	**95.0 %**
Rovers	38	95.0 %	39	97.5 %	**40**	**100.0 %**

Tabla 7.6: Problemas resueltos para el experimento de aprendizaje en línea de Escalada+CBR.

Se puede observar que en ningún caso la diferencia del aprendizaje en línea y los resultados anteriores de Escalada+CBR difieren en más de dos problemas

resueltos. La versión en línea de Escalada+CBR mejora el número de problemas resueltos en seis dominios comparado con los resultados de la evaluación original. En el *Mystery'*, único dominio en el que se empeora el rendimiento, se obtiene de todos modos un rendimiento superior al de Escalada estándar. En este sentido se puede afirmar que el aprendizaje en línea de Escalada+CBR tiene una eficiencia equivalente al Escalada+CBR, con la ventaja de que el aprendizaje en línea no necesita una fase de entrenamiento.

La Tabla 7.7 muestra los resultados obtenidos de EHC+CBR a los que se le ha añadido el resultado del aprendizaje en línea del algoritmo. Al igual que en Escalada+CBR, el aprendizaje en línea obtiene el mejor rendimiento en seis de ocho dominios y sólo varia el resultado final en número de problemas resueltos en un margen de dos. En este caso, en el *Portcrane* se ve empeorado en dos problemas el rendimiento de EHC estándar. En general se puede hacer la misma interpretación, diciendo que el aprendizaje en línea de EHC+CBR tiene un comportamiento equivalente al de EHC+CBR y en algunos casos aún mejor.

Dominio	EHC		EHC+CBR		On-line EHC+CBR	
	Resueltos	%	Resueltos	%	Resueltos	%
Blocksworld	19	47.5 %	**20**	**50.0 %**	**20**	**50.0 %**
Matching-bw	4	10.0 %	3	7.5 %	**5**	**12.5 %**
Parking	30	75.0 %	34	85.0 %	**36**	**90.0 %**
Logistics	**40**	**100.0 %**	**40**	**100.0 %**	39	97.5 %
Mystery'	26	65.0 %	29	72.5 %	**30**	**75.0 %**
Portcrane	**39**	**97.5 %**	**39**	**97.5 %**	37	92.5 %
Satellite	38	95.0 %	39	97.5 %	**40**	**100.0 %**
Rovers	33	82.5 %	**38**	**95.0 %**	**38**	**95.0 %**

Tabla 7.7: Problemas resueltos para el experimento del aprendizaje en línea de EHC+CBR.

7.6. Resumen de Resultados

En este capítulo se ha realizado varias evaluaciones que complementan la experimentación de los capítulos anteriores. Por eso se ha decidido recopilar todos los resultados obtenidos en los experimentos que cumplen las características comunes en sus variables independientes. Aquí se retoma la métrica de calidad de la IPC-6 utilizada también en la sección de aprendizaje de la competición; explicada en Sección 4.2. La Tabla 7.8 muestra las puntuaciones de cada algoritmo en cada dominio de evaluación. Cada dominio puede obtener un máximo de 30 puntos. Al final se presenta un columna con la suma de todos los dominios, a manera de como se realizaría en la competición de planificación. Marcado en negrita se presenta el algoritmo ganador en cada dominio. Según esta métrica resulta ganador el WBFS-Lookahead+CBR, seguido muy de cerca de WBFS-Lookahead, lo que confirma que ambos algoritmos son los más robustos en el conjunto de dominios que se han

evaluado. Cabe señalar, que salvo en el caso de EHC-Lookahead, todos los algoritmos base fueron superados en puntos por cualquiera de las técnicas presentadas en este trabajo.

Algoritmo	blo	mbw	par	log	mpr	pcr	rov	sat	Total
Escalada	8.57	5.94	23.53	13.05	9.27	17.14	27.95	32.43	133.49
Escalada (hdiff)	14.75	8.51	18.30	21.18	10.93	22.41	32.68	**37.37**	161.43
Escalada+CBR	12.37	6.06	20.12	14.40	11.06	28.73	34.49	26.67	158.62
Escalada+CBR-V	8.02	4.03	16.51	21.01	9.46	25.62	35.17	26.48	152.74
Escalada+CBR (online)	11.83	5.91	18.70	17.05	9.45	28.77	34.95	27.03	157.96
EHC	16.28	3.62	24.69	**38.72**	22.45	34.12	31.85	33.76	189.84
EHC (hdiff)	18.08	4.66	22.59	38.39	16.13	35.45	33.36	34.18	191.46
EHC+CBR	15.77	2.79	28.06	38.31	26.07	**37.91**	**37.17**	35.65	205.87
EHC+CBR-V	16.71	4.46	27.72	38.15	26.82	37.83	36.70	36.47	209.75
EHC+CBR (online)	16.81	4.44	29.96	37.55	27.33	32.81	37.02	36.91	204.79
WBFS	22.86	23.95	29.99	17.77	21.61	11.72	14.83	17.51	135.52
WBFS (hdiff)	**26.43**	**25.68**	30.20	19.71	25.48	13.53	12.72	22.85	151.53
WBFS+CBR	23.95	25.46	30.68	16.31	24.48	13.88	14.59	13.52	138.74
EHC-Lookahead	16.89	4.85	30.89	33.71	17.31	35.80	35.40	34.79	201.86
EHC-Lookahead+CBR	20.13	5.83	28.91	32.68	16.67	35.32	36.10	34.66	201.28
WBFS-Lookahead	17.25	22.03	32.18	33.75	27.38	35.61	34.23	34.96	226.71
WBFS-Lookahead+CBR	18.33	21.81	**32.43**	32.64	**28.51**	35.29	35.54	36.74	**227.02**

Tabla 7.8: Puntuaciones de los algoritmos de evaluación según la métrica de calidad de la IPC-6.

En el Apéndice C.4 se muestran otras dos tablas con resumen de resultados. La Tabla C.3 muestra el resumen de los problemas resueltos por los algoritmos en cada dominio. La Tabla C.4 muestra el el factor de eficiencia en la puntuación de calidad, calculado como la división de la puntuación en la métrica de calidad entre el número de problemas resueltos. Visto desde otro modo, este factor es el promedio de puntos que aporta cada problema en cada dominio. El objetivo de mostrar este factor de eficiencia es para diferenciar a los algoritmos que pueden estar resolviendo el mismo número de problemas pero con distinta calidad. Por ejemplo, un 0.25 obtenido por Escalada en el *Blocksworld* representa que en promedio el algoritmo obtiene soluciones cuatro veces más largas que las mejores obtenidas. En cambio, el 0.98 obtenido por WBFS con h^{diff} indica que este algoritmo encontró la mejor solución en la mayoría de los casos.

7.7. Discusión

En las diferentes secciones de este capítulo se han analizado con mayor detalle las secuencias de tipo, con los objetivos de alcanzar una mejor interpretación del conocimiento aprendido, y de intentar mejorar los procesos que involucran el ciclo de razonamiento. En este sentido, es importante debatir las ventajas y las limitaciones del enfoque desarrollado en cada una de las etapas del ciclo de razonamiento. Partiendo de las ideas planteadas en esta discusión se podrán sacar parte de las conclusiones de este trabajo, y las posibles líneas futuras que complementen los detalles no abordados en la tesis.

En la etapa de almacenamiento se puede valorar positivamente la generación de secuencias a partir de problemas resueltos con DFBnB, que sin garantizar la optimalidad, permitió generar secuencias de buena calidad, habiéndose observado pocos casos en los que existía una alternativa de menor longitud para presentar las mismas transiciones. La limitación en este aspecto se encuentra en las ambigüedades que se pueden encontrar al generar las secuencias con diferentes transiciones que están resolviendo el mismo problema, como la situación encontrada en el *Blocksworld*. Desde el punto de vista de CBR tiene sentido almacenar ambos casos, sin embargo, a la hora de reproducir las transiciones, unas resultan ser más beneficiosas que otras. Aquí entran en juego el análisis de las múltiples soluciones que se pueden encontrar en un problema, y la elección de la más conveniente para generar los casos. En nuestro enfoque la decisión es arbitraria, porque DFBnB elige para la generación, la primera solución de entre las encontradas con mejor coste.

También se puede valorar positivamente el proceso de generalización como parte del mantenimiento. De esta manera se puede determinar que en algunos tipos de objetos el número de casos posibles está acotado, y en algunos dominios hasta se puede determinar en función de las posibles transiciones que se dan entre sub-estados de tipo, como sucede en el *Blocksworld* o en el *Parking*. Dado que después del entrenamiento, el número total de casos almacenados no era considerablemente grande, se entiende que almacenar los casos en ficheros separados por tipo es una solución aceptable, y no es necesario contemplar alguna alternativa para el esquema de almacenamiento.

La idea de recuperar una secuencia para cada objeto al principio del proceso de planificación, de forma rápida y sencilla, parece "a priori" una buena elección. El punto de corte entre el algoritmo base y el algoritmo CBR en las distribuciones acumuladas de tiempo mostraron que el coste computacional de la recuperación no es significativo si el problema no es trivial. Sin embargo, el ejemplo mostrado con las transiciones del *Logistics* (análisis semántico) revela que las secuencias, aún teniendo el conocimiento necesario para guiar correctamente la búsqueda, no son recuperadas de forma adecuada, por no considerar otras características aparte de los sub-estados de tipo inicial y final. La huella de propiedades del plan relajado del estado inicial podría refinar en cierta medida la recuperación para algunos tipos. No obstante, es necesario buscar otras alternativas, porque esta característica de recuperación no parece producir diferencias considerables. Esta limitación se sopesa en cierta medida con el aprendizaje de preferencias mostrado en la sección de valoración de las secuencias. Ante la no posibilidad de decidir entre las secuencias en la etapa de recuperación, las valoraciones permitieron seleccionar la secuencia que globalmente reportara mejores resultados en el seguimiento.

En la reutilización, las estrategias de selección o de ordenación permiten en mayor o menor medida conseguir una reducción en el número de nodos evaluados con la función heurística. Las recomendaciones sirven como heurística complementaria que permite discriminar entre estados, prefiriendo aquellos que reproducen transiciones de sub-estados de tipo que han sido vistas en el pasado. Esta heurística complementaria sólo tiene la facultad de discriminar localmente entre

estados, razón por la que sus beneficios se perciben en los algoritmos Escalada y EHC. Por el contrario, en los resultados se vio que las secuencias no tienen forma de discernir acertadamente entre los estados de la lista abierta del WBFS. Todos los intentos de control de búsqueda fueron integrados dentro de la función de evaluación en donde el valor heurístico tiene mucho peso. En este sentido habría que estudiar otras formas de seguimiento simultáneo para poder compensar los fallos de los caminos sin salida. Los múltiples caminos permitirían encontrar soluciones de forma más eficiente en dominios como el *Matching Blocksworld* o el *Mystery'*.

Otro aspecto a resaltar del seguimiento es el valor particular que reciben las secuencias recuperadas. En nuestro sistema, todas las secuencias aportan por igual a la función r. En el análisis de seguimiento se observó que el conocimiento de ciertos tipos de secuencias es más relevantes que el de otros. Por ejemplo, en el *Matching Blocksworld*, el tipo `block` importa más que el tipo `hand`, o en el *Portcrane* el tipo `container` importa más que cualquier otro tipo. Este conocimiento es dependiente del dominio y requiere de un experto para descubrirlo. Una fase de validación como la utilizada en la evaluación de las valoraciones podría ser útil para determinar en cada dominio los tipos de secuencias más relevantes, y así mantener sólo las importantes en la tabla de seguimiento. Experimentación previa mostró que recuperar secuencias sólo para los objetos que aparecen en las metas resultaba menos beneficioso que recuperar secuencias para todos los objetos. Queda determinar si se puede obtener una mejora excluyendo ciertos tipos en lugar de excluir ciertos objetos. El ejemplo de interferencia en el *Portcrane*, que se mostró en el análisis de seguimiento, también revela que el seguimiento de las secuencias de ciertos tipos es más importante. No obstante, con este efecto queda en evidencia la limitación que involucra separar del todo las transiciones, centrándolas en los objetos individuales. Hay información relacional que no puede deducirse durante la planificación porque las secuencias se recuperan y se siguen por separado. En ocasiones esta limitación no es tan evidente, porque en el caso de Escalada, el ratio de recomendación preferirá las acciones donde más secuencias se reproduzcan a la vez, creando una dependencia en las transiciones de las secuencias involucradas. Por el contrario, la limitación sí es evidente en las situaciones de empate o en situaciones intrínsecas al dominio, como en el caso de los dominios de construcción, en los que el orden en que se resuelven las metas impone restricciones que sólo se deducen a partir de la información relacional mencionada.

Además de las etapas en el ciclo de razonamiento, entran en el debate los aspectos relacionados con la representación de los casos. Las secuencias de tipo recogen claramente los episodios de planificación relacionados con los objetos del problema, y en ocasiones almacenan más información de la que luego se puede utilizar (en la versión actual del sistema). La capacidad de recoger las transiciones típicas a través de la experiencia, permite a las secuencias ofrecer más información de la que se puede deducir de las invariantes de estado. Cabe destacar la importancia de la acción asociada en cada sub-estado de tipo, que en varias situaciones es la que vale para reconocer el avance en el seguimiento de la secuencia. Visto de otro modo, la acción asociada a los sub-estados de tipo hace ver a las secuencias como políticas

ligeras, que ante una correspondencia parcial con un estado, dicen qué acción se debe elegir. No obstante, la representación utilizada plantea sus limitaciones en los siguientes aspectos.

- Todo el conocimiento reside en las secuencias y no existe una forma de recopilar una meta-información del dominio que sirva también de soporte. Por ejemplo, un vector de pesos que represente la importancia que tiene el seguimiento de las secuencias de ciertos tipos frente a otros.

- Algunas secuencias están limitadas al tamaño de los problemas de entrenamiento y en ocasiones guardan un esquema de repetición que no puede deducirse por falta de una representación más rica. En el *Satellite*, bastaría representar después del operador `CALIBRATE`, un bloque de tipo **repetir**{*no-op*, `TAKE-IMAGE`}, con sus respectivos sub-estados de tipo.

- Las secuencias sólo guardan episodios positivos de la planificación. En el proceso de reutilización, si no se puede reproducir la misma transición que en la secuencia de tipo, la búsqueda se guía por la heurística, sin que la secuencia pueda advertir, por ejemplo, de un camino sin salida en alguna transición alternativa, como suele suceder en el *Matching Blocksworld*. El aprendizaje de ejemplos negativos centrados en los objetos es perfectamente viable, extrayendo las transiciones de los estados a partir del árbol de búsqueda generado por el DFBnB.

Por otro lado, las decisiones sobre la representación escogida se apoyaron en la posibilidad de utilizar el conocimiento para su reutilización como conocimiento de control en la búsqueda. No se pensó plantear la interpretación del conocimiento almacenado como herramienta para la re-ingeniería de dominio. Después del análisis semántico y de seguimiento de las secuencias se han visto cuestiones que intuitivamente no pueden descubrirse en los dominios. Por tanto, se puede plantear una línea de investigación que vaya en dirección inversa. Esto es, ver qué transiciones no pueden reproducirse por las secuencias (o incluso en relación con la heurística) para intentar redefinir predicados o acciones del dominio. Cuestiones como el número limitado de transiciones típicas que se dan, sobre todo en los dominios de construcción, son características que no son adquiribles a simple vista, necesitando técnicas como las planteadas en este trabajo o técnicas específicas de análisis de dominio como las invariantes de estado. Como apunte adicional cabe señalar que aunque varios dominios de los utilizados por la comunidad están inspirados en aplicaciones reales como por ejemplo el *Rovers*, otros fueron diseñados para plantear retos para las técnicas actuales de planificación. Por ejemplo el *Matching Blocksworld*, diseñado para la sección de aprendizaje de la IPC-6, tiene alternativas de representación posiblemente más eficientes que podrían deducirse en una re-ingeniería dominio. En este ámbito quedan preguntas abiertas relacionadas con la eficiencia que puedan tener las recomendaciones CBR sobre diferentes representaciones de un mismo dominio.

Capítulo 8

Conclusiones

A continuación se presenta un capítulo resumen de esta tesis para recoger las principales conclusiones obtenidas. Se ha dividido en dos secciones. La primera es una recopilación de los puntos claves sacados de las secciones de discusión de cada capítulo, junto con otras ideas importantes presentadas a lo largo del desarrollo. La segunda sección es la exposición de las aportaciones que realiza esta tesis en el ámbito científico y a la comunidad de planificación automática.

8.1. Resumen

Uno de los problemas centrales que tiene la planificación heurística es el coste computacional que tiene evaluar las heurísticas independientes del dominio construidas a partir de la relajación de la lista de borrados. Queda de manifiesto que diferentes estrategias ayudan a mejorar: heurísticas que discriminan más entre estados, como h^{diff}; conocimiento de control dependiente del dominio o técnicas de estados *lookahead*. A pesar de esto, el tiempo empleado por las evaluaciones sigue siendo un problema abierto, porque falta conseguir que estas técnicas sean más robustas frente a una mayor variedad de dominios.

La heurística h^{diff} complementa la heurística h^{FF}, porque le permite discriminar entre estados con el mismo valor heurístico. Utilizando información del grafo de planificación relajado, se calcula un valor adicional que se suma a la heurística original, de modo que el valor heurístico final pueda diferenciar entre estados en los que se pueden construir planes más paralelos, frente a estados en los que los planes equivalentes son más secuenciales.

Las secuencias de tipo son una alternativa de representación a los planes parciales. Las transiciones de estado centradas en los objetos, junto con la acción asociada, guardan buena información que puede ser utilizada como conocimiento de control para la búsqueda. Este control de la búsqueda, conceptualizado como recomendación CBR, permite su fácil integración con diferentes estrategias sobre los diferentes algoritmos de búsqueda.

Las evaluaciones experimentales revelaron que las secuencias de tipo pueden

servir como heurísticas dependientes del dominio que valen para discriminar localmente entre estados, permitiendo una reducción en el número de evaluaciones en varios de los dominios de prueba. En ciertos dominios, las secuencias de tipo permiten reproducir caminos sin acciones irrelevantes, que se obtienen normalmente por la poca información que ofrece la función heurística. En general, con independencia del algoritmo de búsqueda, las secuencias de tipo permitieron obtener resultados mejores o al menos equivalentes al algoritmo base en los dominios de evaluación.

La evaluación en línea del ciclo CBR mostró que el enfoque permite una adquisición incremental del conocimiento, sin una estricta necesidad de una fase de entrenamiento como suelen necesitar otras técnicas de aprendizaje automático. Los resultados mostraron mejoras equivalentes en aquellos dominios en los que un enfoque tradicional también funcionó.

Los análisis realizados en el Capítulo 7 revelaron que el planteamiento actual de la técnica permite muchos puntos de refinamiento. El pobre seguimiento de algunos tipos de secuencias, la importancia de unos tipos frente a otros, información relevante en las secuencias que no puede ser utilizada, son, entre otros, ciertos detalles que dejan el campo abierto a futuras líneas de investigación.

8.2. Contribuciones

Se consideran como contribuciones en el ámbito científico a aquellas técnicas desarrolladas que dan un avance al estado del arte. Esta tesis contribuye en el campo de planificación y aprendizaje en los siguientes puntos:

- La heurística h^{diff}, como una mejora a la heurística h^{FF}, lo que influye directamente a aquellos planificadores heurísticos basados en FF. Experimentalmente se mostró que h^{diff} es más eficiente en una variedad de dominios.

- Las secuencias de tipo como forma de representación de episodios de planificación centrados en los objetos, que pueden ser almacenadas en una base de casos asociada a un dominio.

- La integración de las secuencias de tipo como conocimiento de control en los algoritmos de búsqueda, dando lugar a los diferentes algoritmos CBR.

- Un modelo para las valoraciones de las secuencias, que permite, mediante una fase de validación, asignar preferencias de recuperación o seguimiento a las secuencias aprendidas.

- Las interpretaciones a las secuencias aprendidas en varios dominios, que enriquecen el análisis de dominio que aportan las invariantes de estado. Las secuencias de tipo recogen mediante la experiencia, las transiciones típicas en cada dominio, que son acompañadas de las acciones asociadas, construyendo así una forma de política general.

- Evaluación experimental de una librería de algoritmos implementados sobre un mismo sistema, que permite sacar las diferentes conclusiones del resumen antes expuesto.

Las contribuciones a la comunidad de planificación son el legado que deja el trabajo en términos tangibles, y que pueden utilizarse por los demás investigadores dentro del campo, para continuar el desarrollo del estado del arte o para transferencia de tecnología y llevarlos al campo de las aplicaciones reales. En este sentido el trabajo contribuye en:

- El planificador SAYPHI, que integra en su sistema dos heurísticas (h^{FF} y h^{diff}) y una librería con los principales algoritmos de búsqueda heurística. SAYPHI es el planificador base para CABALA, ROLLER, REPLICA y SAYPHI-RULES, cuatro de los competidores en la sección de aprendizaje de la IPC-6. Además, SAYPHI participa en la transferencia de tecnología, al ser el planificador utilizado en varios proyectos de investigación que intentan aplicar técnicas de planificación a dominios reales como automatizar procesos de minería de datos, resolver problemas de logística intermodal o generar cursos adaptados al estudiante.

- El dominio *Portcrane*, que puede ser incluido en la colección de dominios de evaluación que utiliza la comunidad. El dominio y un problema de ejemplo se encuentran en el Apéndice B.

Capítulo 9

Líneas Futuras

La eficiencia de las técnicas de planificación heurística sobre el modelo de planificación clásica sigue siendo un problema abierto, por lo que la evolución de ésta y otras técnicas de aprendizaje plantean nuevos trabajos de investigación. Por otro lado, las limitaciones encontradas al sistema CBR que se ha desarrollado, también plantean oportunidades de trabajos futuros, para conseguir mejoras adicionales a la eficiencia del planificador. Las propuestas para trabajos futuros a corto plazo son:

- Enriquecer el lenguaje utilizado para la representación de las secuencias de tipo, para recoger episodios de circunstancias iterativas, y para almacenar transiciones de fallo, que puedan ser utilizadas a la hora de un seguimiento más dependiente de las secuencias y menos dependiente de la heurística.

- Estudiar qué nuevas claves de recuperación pueden utilizarse para mejorar el proceso, y poder brindar a la fase de seguimiento el conocimiento correcto, que en ocasiones sí está presente en la base de casos.

- Desarrollar un modelo de preferencias por tipo de secuencias, que pueda ser incluido como una meta-información de la base de casos de un dominio, de modo que el seguimiento tenga en cuenta el valor que tienen las secuencias de cada tipo.

- Integrar el conocimiento de control aportado en las secuencias a nuevos algoritmos de búsqueda. La propuesta puede ser la aplicación de una selección directa a una profundidad determinada antes de evaluar nuevamente el estado. Esta sería la contrapartida de las ordenaciones de los *lookahead*, pero en lugar de partir de las acciones del plan relajado, se construirían los estados con recomendaciones CBR encadenadas.

A estas propuestas se pueden añadir ciertas líneas de interés que pueden pensarse como trabajos futuros a más largo plazo:

- Utilizar las secuencias de tipo enriquecidas como técnica de almacenamiento de preferencias de estado. Ampliaciones al modelos de planificación clásica, ya recogidas en las últimas versiones de PDDL, intentan modelar la planificación como la construcción de planes en los que se prefiere transitar por ciertos estados. Un enfoque CBR puede con cierta facilidad, explotar la idea de almacenar las preferencias dentro de las posibles transiciones, para después reproducirlas en nuevos problemas.

- Integrar el enfoque CBR de las secuencias de tipo para mejorar la eficiencia de los planificadores que trabajan en entornos de ejecución. La incertidumbre que aparece en los entornos de ejecución se puede sopesar mediante un enfoque de replanificación que busque las transiciones más robustas en problemas pasados. Bajo la filosofía CBR, las transiciones de estado más robustas en problemas aprendidos pueden almacenarse para recomendar la elección de los caminos en los nuevos problemas.

Bibliografía

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7, no.1:39–59.

Bacchus, F. (2001). The AIPS'00 planning competition. *AI Magazine*, 22(3):47–56.

Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191.

Backstrom, C. and Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–655.

Bergmann, R. and Wilke, W. (1996). Paris: Flexible plan adaptation by abstraction and refinement. In Voss, A., editor, *ECAI (1996) Workshop on Adaptation in Case-Based Reasoning*. John Wiley & Sons.

Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297.

Blum, A. and Furst, M. (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, Montreal, Canada. Morgan Kaufmann.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33.

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.

Botea, A., Enzenberger, M., Müller, M., and Schaeffer, J. (2005). Macro-FF: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621.

Botea, A., Müller, M., and Schaeffer, J. (2007). Fast planning with iterative macros. In *Proceedings of the IJCAI-2007*.

Brafman, R. I. and Tennenholtz, M. (2002). R-max a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

Bresina, J., Jonsson, A., Morris, P., and Rajan, K. (2005). Activity planning for the mars exploration rovers. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2005)*. AAAI Press.

Bylander, T. (1994). The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204.

Carbonell, J. G., Blythe, J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Pérez, A., Reilly, S., Veloso, M. M., and Wang, X. (1992). PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University.

Castillo, L., Fernández-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 2006)*. AAAI Press.

Cocora, A., Kersting, K., Plagemann, C., Burgard, W., and Raedt, L. D. (2006). Learning relational navigation policies. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-06)*, Beijing, China.

Coles, A., Fox, M., and Smith, A. (2006). A new local search algorithm for forward-chaining planning. In Boddy, M., Fox, M., and Thiébaux, S., editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-2007)*, Providence, Rhode Island, USA.

Coles, A. and Smith, A. (2007). Marvin: A heuristic search planner with on-line macro-actions learning. *Journal of Artificial Intelligence Research*, 28:119–156.

Cox, M., Muñoz-Avlia, H., and Bergmann, R. (2005). Case-based planning. *Knowledge Engineering Review*, 20(3):283–287.

Cox, M. and Veloso, M. (1997). Supporting combined human and machine planning: An interface for planning by analogical reasoning. In Leake, D. and Plaza, E., editors, *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning*, pages 531–540, Berlin, Germany.

De la Rosa, T., Borrajo, D., and García-Olaya, A. (2006). Replaying type sequences in forward heuristic planning. In *Technical Report of the AAAI'06 Workshop on Learning for Search*, Boston, MA (USA). AAAI Press.

De la Rosa, T. and Fuentetaja, R. (2006). Integrating actions precondition difficulty within the relaxed plan heuristic measure. In *Technical Report of the AAAI'06 Workshop on Heuristic Search, Memory-based Heuristics and their Application*, Boston, MA (USA). AAAI Press.

De la Rosa, T., Jiménez, S., and Borrajo, D. (2008). Learning relational decision trees for guiding heuristic planning. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 08)*.

Dzeroski, S., Raedt, L. D., and Blockeel, H. (1998). Relational reinforcement learning. In *International Workshop on Inductive Logic Programming*, pages 11–22.

Edelkamp, S. (2001). Planning with pattern databases. In *Proceedings of the European Conference on Planning (ECP-01)*, pages 13–34, Toledo, Spain.

Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–301.

Fikes, R., Hart, P., and Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Fox, M. and Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:317–371.

Fox, M. and Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, pages 61–124.

Geffner, H. (2002). Perspectives on artificial intelligence planning. In *Proceedings of AAAI-2002*, pages 1013–1023. AAAI/MIT Press. Invited Talk.

Gerevini, A., Long, D., Haslum, P., Saetti, A., and Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence*, 173:619–668.

Gerevini, A., Saetti, A., and Serina, I. (2004). Planning with numerical expressions in lpg. In *Proceedings of the 16th European Conference on Artificial Intellience (ECAI-04)*, pages 667–671.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning, Theory and Practice*. Morgman Kaufmann.

Hammond, K. J. (1990). Case-based planning: A framework for planning from experience. *Cognitive Science*, 14(3):385–443.

Hanks, S. and Weld, D. (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Helmert, M., Haslum, P., and Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*. AAAI Press.

Hillis, D. W. (1985). *The Connection Machine*. MIT Press.

Hoffmann, J. (2003). The METRIC-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341.

Hoffmann, J. (2005). Where "ignoring delete lists" works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758.

Hoffmann, J. and Brafman, R. (2006). Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6–7):507–541.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Holte, R., Newton, J., Felner, A., Meshulam, R., and Furcy, D. (2004). Multiple pattern databases. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS-04)*, pages 122–131, Whistler, British Columbia, Canada.

Ihrig, L. H. and Kambhampati, S. (1996). Design and implementation of a replay framework based on a partial order planner. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 849–854, Portland, Oregon, USA. AAAI Press / The MIT Press.

Kambhampati, S. and Hendler, J. (1992). A validation structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258.

Kautz, H. A. and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363.

Kettler, B. P., Hendler, J. A., Andersen, W. A., and Evett, M. P. (1994). Massively parallel support for case-based planning. *IEEE Expert*, pages 8–14.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148.

Koehler, J. (1996). Planning from second principles. *Artificial Intelligence*, 87:148–187.

Koehler, J. and Hoffmann, J. (1999). Handling of inertia in a planning system. Technical Report 122, Albert Ludwigs University, Freiburg, Germany.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann.

Kolodner, J. and Leake, D. (1996). A tutorial introduction to case-based reasoning. In Leake, D., editor, *Case-Based Reasoning: Experiences, Lessons & Future Directions*, pages 167–183. AAAI Press / Mit Press.

Leckie, C. and Zukerman, I. (1991). Learning search control rules for planning: An inductive approach. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 422–426, Evanston, IL. Morgan Kaufmann.

Long, D. and Fox, M. (2003). The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59.

Macedo, L. and Cardoso, A. (2004). Cased-based, decision-theoretic, HTN-Planning. In Funk, P. and Calero, P. G., editors, *Advances in Case-Based Reasoning. Proceedings of 7th European Conference in CBR*, Madrid, Spain. Springer.

Manna, Z. and Waldinger, R. (1987). A theory of plans. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning about Actions and Plans*, pages 11–45. Kaufmann, Los Altos, CA.

Martin, M. and Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Appl. Intell*, 20:9–19.

McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 634–639, Menlo Park, California, USA.

McDermott, D. (1996). A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on AI Planning Systems*.

Mcdermott, D. (2000). The 1998 AI planning systems competition. *AI Magazine*, 21:35–55.

McGann, C., Py, F., Rajan, K., Ryan, H., and Henthorn, R. (2008). Adaptative control for autonomous underwater vehicles. In *Proceedings of the 23rd AAAI Conference*. AAAI Press.

Minton, S. (1988). *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Muñoz-Avila, H., Aha, D., Nau, D., Weber, R., Breslow, L., and Yaman, F. (2001). Sin: Integrating case-based reasoning with task decomposition. In *Proceedings of the IJCAI 2001*. AAAI Press.

Muñoz-Avila, H., Paulokat, J., and Wess, S. (1994). Controlling nonlinear hierarchical planning by case replay. In *Working papers of the Second European Workshop on Case-based Reasoning*, pages 195–203, Chantilly, France.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., and F.Yaman (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404.

Nebel, B., Dimopoulos, Y., and Koehler, J. (1997). Ignoring irrelevant facts and operators in plan generation. In *ECP-97*, pages 338–350.

Newton, H., Levine, J., Fox, M., and Long, D. (2007). Learning macro-actions for arbitrary planners and domains. In *Proceedings of the ICAPS 2007*.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, pages 103–114.

Perini, A. and Ricci, F. (1996). An interactive planning architecture. In Ghallab, M. and Milani, A., editors, *New Directions in AI Planning*. IOS Press.

Ram, A. and Francis, A. (1996). Multi-plan retrieval and adaptation in an experience-based agent. In Leake, D., editor, *Case-Based Reasoning: Experiences, Lessons & Future Directions*, pages 167–183. AAAI Press / Mit Press.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence, A Modern Approach*. Prentice Hall.

Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press, Cambridge, England.

Shahaf, D. and Amir, E. (2006). Learning partially observable action schemas. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'06)*, pages 667–671.

Spalazzi, L. (2001). A survey on case-based planning. *Artificial Intelligence Review*, 16:3–36.

Veloso, M. and Carbonell, J. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278.

Vidal, V. (2004). A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 150–160, Whistler, British Columbia, Canada.

Wang, X. (1994). Learning planning operators by observation and practice. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 335–340, Chicago, IL, USA. AAAI Press.

Warfield, I., Hogg, C., Lee-Urban, S., and Muñoz-Avila, H. (2007). Adaptation of hierarchical task network plans. In Dankel II, D., editor, *Proceedings of the 20th International FLAIRS Conference*, Key West, FL (USA). AAAI Press.

Watson, I. (1997). *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann Publisher.

Weberskirch, F. (1995). Combining snlp-like planning and dependency-maintenance. Technical Report LSA-95-10E, Centre for Learning Systems and Applications, University of Kaiserslautern.

Winner, E. and Veloso, M. (2003). Distill: Towards learning domain-specific planners by example. In *Proceedings of Twentieth International Conference on Machine Learning (ICML 03)*, Washington, DC, USA.

Xu, Y. and Fern, A. (2007). On learning linear ranking functions for beam search. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Yang, Q., Wu, K., and Jiang, Y. (2007). Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143.

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-2006)*, The Lake District, Cumbria, UK.

Yoon, S., Fern, A., and Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of ICAPS-2007*.

Yoon, S., Fern, A., and Givan, R. (2008). Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9:683–718.

Apéndice A

Publicaciones

A continuación se presenta una lista de las publicaciones en congresos o conferencias internacionales que están ligadas al desarrollo de esta tesis.

Título:	Learning Relational Decision Trees for Guiding Heuristic Planning
Autores:	Tomás de la Rosa, Sergio Jiménez y Daniel Borrajo
Publicación:	Proceedings of the 18th International Conference on Automated Planning and Scheduling
Fecha:	Septiembre 2008

Título:	Using Cases Utility for Heuristic Planning Improvement
Autores:	Tomás de la Rosa, Angel García Olaya y Daniel Borrajo
Publicación:	Proceedings of the 7th International Conference on Case-Based Reasoning
Fecha:	Agosto 2007

Título:	Case-based Search Control for Heuristic Planning
Autores:	Tomás de la Rosa
Publicación:	ICAPS 2007 Doctoral Consortium
Fecha:	Septiembre 2007

Título:	Case-based Recommendation for Node Ordering in Planning
Autores:	Tomás de la Rosa, Angel García Olaya y Daniel Borrajo
Publicación:	Proceedings of the 20th International FLAIRS Conference
Fecha:	Mayo 2007

Título:	Replaying Type Sequences in Forward Heuristic Planning
Autores:	Tomás de la Rosa, Daniel Borrajo y Angel García Olaya
Publicación:	Technical Report of the AAAI'06 Workshop on Learning for Search
Fecha:	Julio 2006

Título:	Integrating Actions Precondition Difficulty within the Relaxed Plan Heuristic Measure
Autores:	Tomás de la Rosa y Raquel Fuentetaja
Publicación:	Technical Report of the AAAI'06 Workshop on Heuristic Search, Memory-based Heuristics and their Application
Fecha:	Julio 2006

Apéndice B

El Dominio Portcrane

Este apéndice muestra la definición del dominio y un ejemplo de problema en el dominio *Portcrane*, utilizado como dominio de evaluación en esta tesis. La tarea del dominio consiste en utilizar grúas para desembarcar contenedores de un barco, y colocarlos en sus respectivas zonas de destino en el puerto. Las grúas que desembarcan del barco, llamadas *portainers* dejan los contenedores en una zona de transito, desde la que a continuación, las grúas movibles o *transtainers* los llevan a la zona de destino. Algunos contenedores son normales (*dryvan*), pero otros necesitan un tratamiento especial. Los contenedores especiales pueden venir congelados, o con carga fijada con seguros (por tener la abierta la tapa superior) y estas características son importantes para elegir el tipo de acción que ejecuta la grúa. Una acción no especializada de la grúa implicaría que al colocar el contenedor tiene que recuperarse la propiedad con una acción de reparación, por ejemplo `freeze-cargo` para congelar nuevamente el contenedor.

El dominio fue desarrollado para mostrar que características de los objetos, presentes en el dominio, son útiles para diferenciar las tipologías de secuencias que se generan, permitiendo una mejor precisión a la hora de recuperar o seguir una secuencia de tipo. Al igual que otros dominios utilizados para investigación, este dominio es una simplificación de la problemática real de desembarcar los contenedores de un barco.

B.1. Portcrane en PDDL

```
(define (domain port-terminal)
 (:requirements :typing)
 (:types shiparea groundarea - area
 container crane)
 (:predicates
   (at-crane ?x - crane ?y - area)
   (located ?x - container ?y - groundarea)
   (reachable ?x - crane ?y - area)
   (available ?x - crane)
   (on-ship ?x - container)
   (lifting ?x - crane ?y - container)
```

```
   (locked ?x - container)
   (frozen ?x - container)
   (dryvan ?x - container)

   (reefer-crane ?x - crane)
   (opentop-crane ?x - crane)

   (reefer-connected ?x - container)
   (opentop-locked ?x - container))

(:action Unload-ship
         :parameters (?x - crane ?y - container ?z - shiparea)
         :precondition (and (on-ship ?y) (at-crane ?x ?z) (available ?x))
         :effect (and (not (on-ship ?y)) (not (available ?x))
                      (lifting ?x ?y)))

(:action Unload-ship-freeze
         :parameters (?x - crane ?y - container ?z - shiparea)
         :precondition (and (on-ship ?y) (at-crane ?x ?z) (available ?x)
                            (reefer-crane ?x))
         :effect (and (not (on-ship ?y)) (not (available ?x))
                      (lifting ?x ?y) (reefer-connected ?y)))

(:action Unload-ship-lock
         :parameters (?x - crane ?y - container ?z - shiparea)
         :precondition (and (on-ship ?y) (at-crane ?x ?z) (available ?x)
                            (opentop-crane ?x))
         :effect (and (not (on-ship ?y)) (not (available ?x))
                      (lifting ?x ?y) (opentop-locked ?y)))

(:action Move-crane
         :parameters (?x - crane ?a1 ?a2 - area)
         :precondition (and (reachable ?x ?a1) (reachable ?x ?a2)
                            (at-crane ?x ?a1))
         :effect (and (not (at-crane ?x ?a1)) (at-crane ?x ?a2)))

(:action Drop-yard
         :parameters (?x - crane ?y - container ?a - groundarea)
         :precondition (and (lifting ?x ?y) (at-crane ?x ?a))
         :effect (and (not (lifting ?x ?y)) (available ?x)
                      (not (frozen ?y)) (not (locked ?y))
                      (not (reefer-connected ?y))
                      (not (opentop-locked ?y))
                      (located ?y ?a)))

(:action Drop-yard-freeze
         :parameters (?x - crane ?y - container ?a - groundarea)
         :precondition (and (lifting ?x ?y) (at-crane ?x ?a)
                            (reefer-crane ?x) (reefer-connected ?y))
         :effect (and (not (lifting ?x ?y)) (available ?x)
                      (not (reefer-connected ?y)) (not (locked ?y))
                      (located ?y ?a)))

(:action Drop-yard-lock
         :parameters (?x - crane ?y - container ?a - groundarea)
         :precondition (and (lifting ?x ?y) (at-crane ?x ?a)
                            (opentop-crane ?x) (opentop-locked ?y))
         :effect (and (not (lifting ?x ?y)) (available ?x)
                      (not (opentop-locked ?y)) (not (frozen ?y))
                      (located ?y ?a)))

(:action lift-yard
```

```
         :parameters (?x - crane ?y - container ?a - groundarea)
         :precondition (and (located ?y ?a) (available ?x)
                            (at-crane ?x ?a))
         :effect (and (lifting ?x ?y) (not (available ?x))
                      (not (located ?y ?a))))

(:action lift-yard-freeze
         :parameters (?x - crane ?y - container ?a - groundarea)
         :precondition (and (located ?y ?a) (available ?x) (frozen ?y)
                            (at-crane ?x ?a) (reefer-crane ?x))
         :effect (and (lifting ?x ?y) (not (available ?x))
                      (not (located ?y ?a)) (reefer-connected ?y)))

(:action lift-yard-lock
         :parameters (?x - crane ?y - container ?a - groundarea)
         :precondition (and (located ?y ?a) (available ?x) (locked ?y)
                            (at-crane ?x ?a) (opentop-crane ?x))
         :effect (and (lifting ?x ?y) (not (available ?x))
                      (not (located ?y ?a)) (opentop-locked ?y)))

(:action Freeze-cargo
         :parameters (?c - container ?a - groundarea)
         :precondition (and (located ?c ?a))
         :effect (and (frozen ?c)))

(:action Lock-cargo
         :parameters (?c - container ?a - groundarea)
         :precondition (and (located ?c ?a))
         :effect (and (locked ?c)))
)
```

B.2. Problemas en el Portcrane

Los problemas en el *Portcrane* describen la situación inicial en la que todos los contenedores, con sus características respectivas, están en el barco, y las grúas están en una zona aleatoria de sus zonas accesibles. La meta de los problemas es tener a los contenedores en una zona del puerto, con las mismas características con las que estaban en el barco. El siguiente ejemplo muestra un problema con dos contenedores.

```
(define (problem portcrane-prob-4)
  (:domain portcrane)
  (:objects
     ship - shiparea
     transit zone1 zone2 - groundarea
     portainer1 transtainer1 transtainer2 - crane
     c1 c2 c3 - container)

 (:init
    (reachable portainer1 ship) (reachable portainer1 transit)
    (reachable transtainer1 transit) (reachable transtainer2 transit)
    (reachable transtainer1 zone1) (reachable transtainer2 zone1)
    (reachable transtainer1 zone2) (reachable transtainer2 zone2)

    (reefer-crane portainer1) (opentop-crane portainer1)
```

```
    (reefer-crane transtainer1)
    (opentop-crane transtainer2)

    (at-crane portainer1 ship) (available portainer1)
    (at-crane transtainer1 zone1) (available transtainer1)
    (at-crane transtainer2 transit) (available transtainer2)

    (on-ship c1) (dryvan c1)
    (on-ship c2) (frozen c2)
    (on-ship c3) (locked c3))
 (:goal
  (and (located c1 zone1)
       (located c2 zone2) (frozen c2)
       (located c3 zone1) (locked c3))))
```

Apéndice C

Resultados Adicionales

En este apéndice se muestran los resultados adicionales de los experimentos realizados, que por fluidez del documento no fueron incluidos en las secciones de resultados de los capítulos correspondientes.

C.1. Resultados Adicionales para h^{diff}

C.1.1. Escalada con h^{diff}

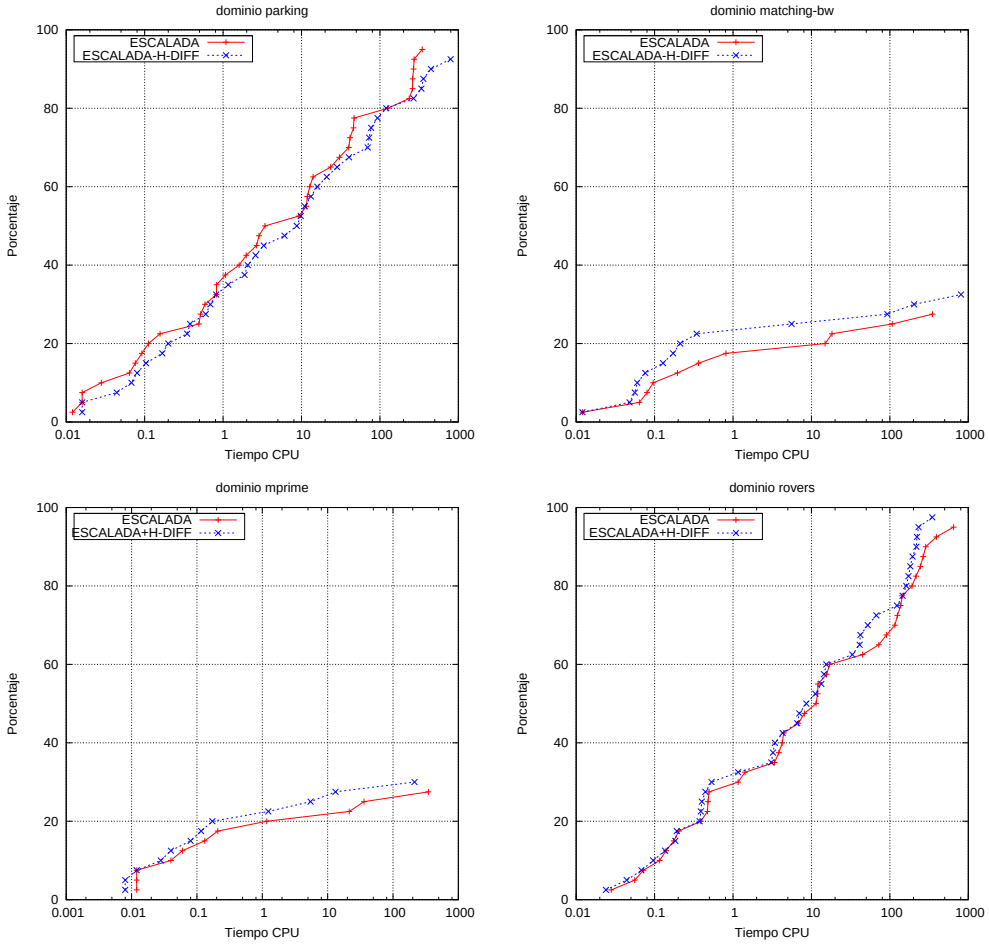


Figura C.1: Distribución acumulada de tiempo de CPU para experimentos con Escalada y h^{diff}.

C.1.2. EHC con h^{diff}

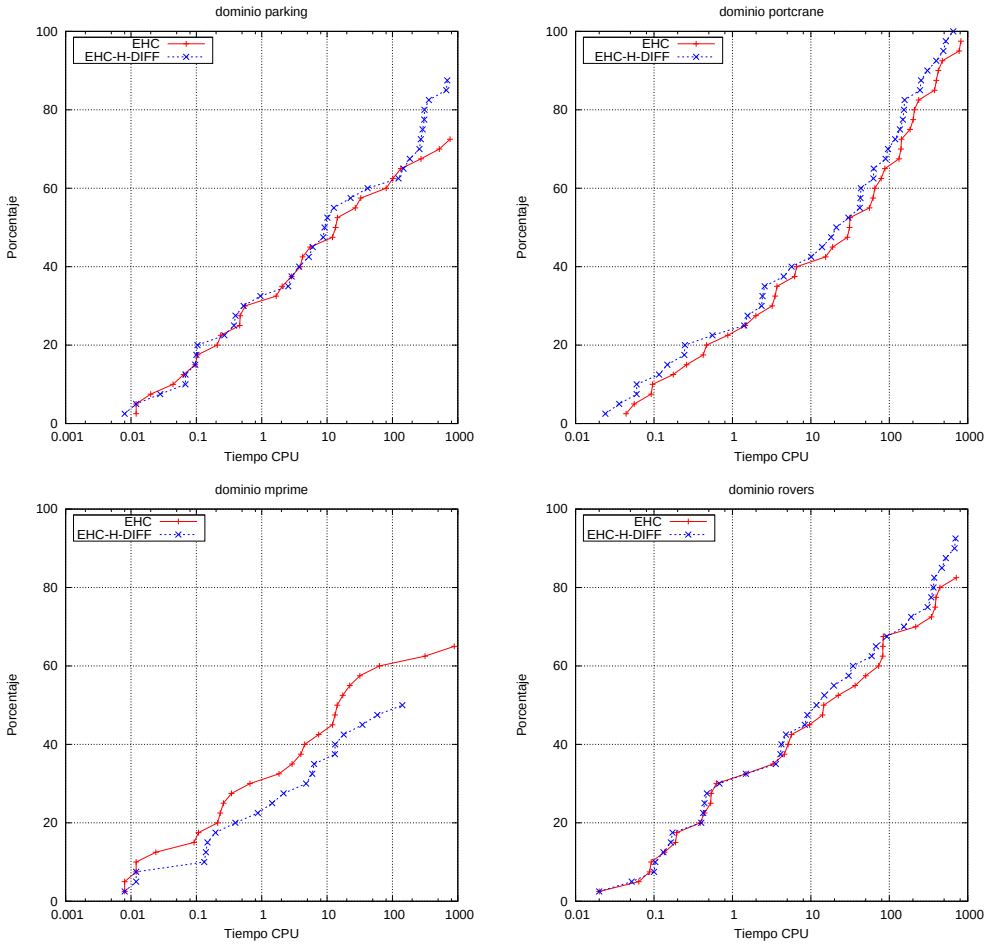


Figura C.2: Distribución acumulada de tiempo de CPU para experimentos con EHC y h^{diff}.

C.1.3. WBFS con h^{diff}

Dominio	WBFS	WBFS con h^{diff}
Blocksworld (23)	38.26	**37.22**
Matching-bw (25)	**30.08**	30.40
Parking (30)	23.90	**23.87**
Logistics (19)	76.37	**72.47**
Mystery' (23)	**27.57**	27.74
Portcrane (12)	**73.08**	73.67
Satellite (18)	27.94	**27.11**
Rovers (13)	**37.38**	37.85

Tabla C.1: Promedio de longitud de los planes para problemas resueltos en ambas configuraciones del experimento de WBFS con h^{diff}.

C.2. Resultados Adicionales para Algoritmos CBR

C.2.1. Escalada+CBR

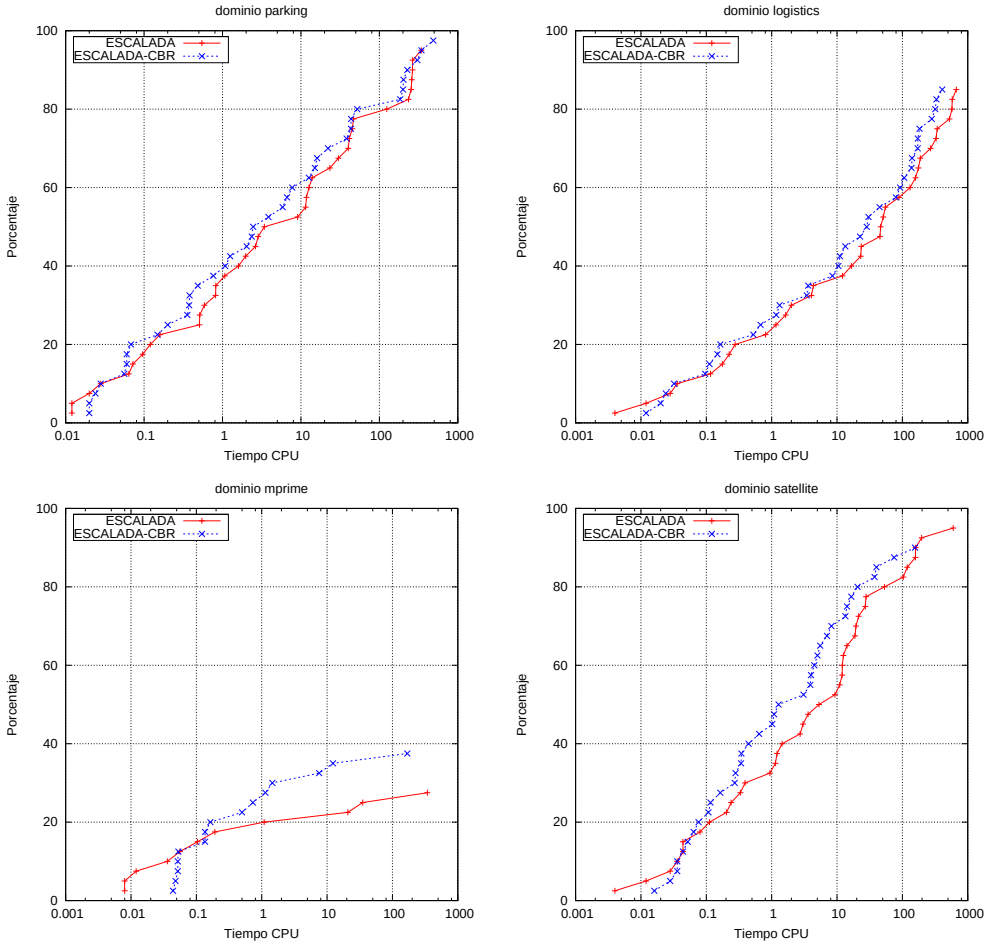


Figura C.3: Distribución acumulada de tiempo de CPU para experimentos con Escalada y CBR.

C.2.2. EHC+CBR

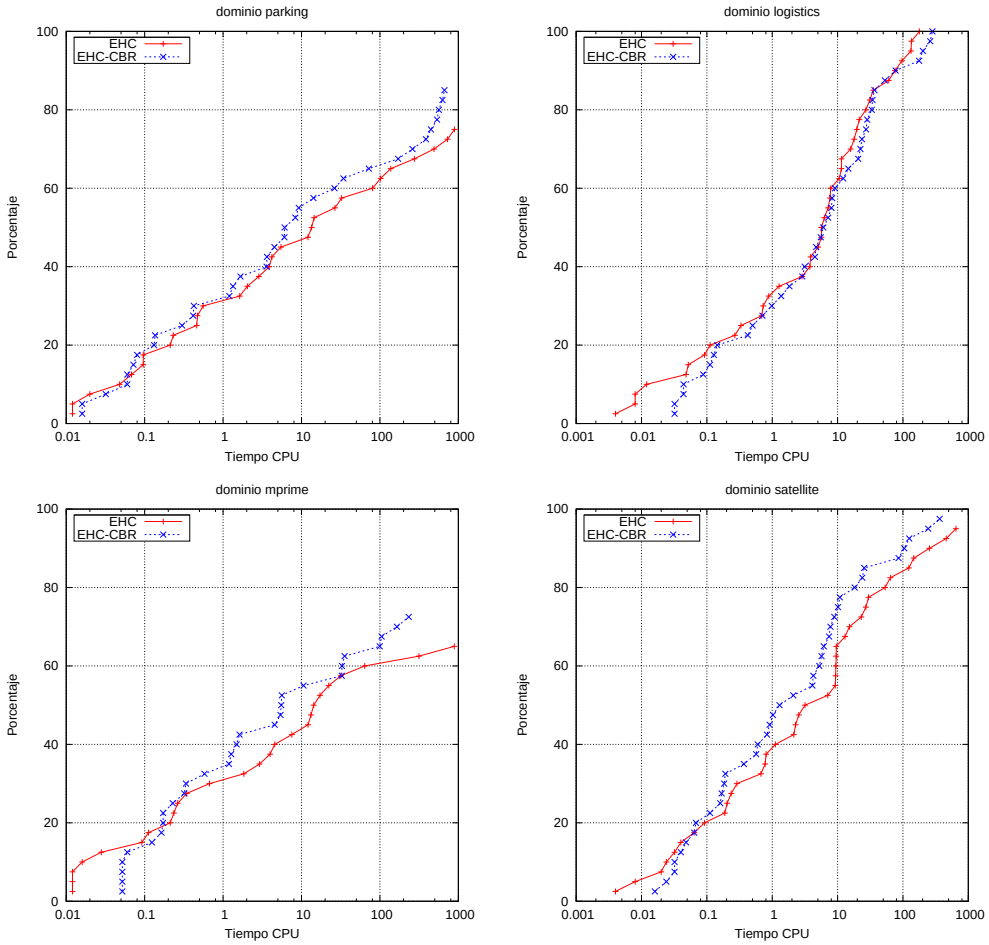


Figura C.4: Distribución acumulada de tiempo de CPU para experimentos con EHC y CBR.

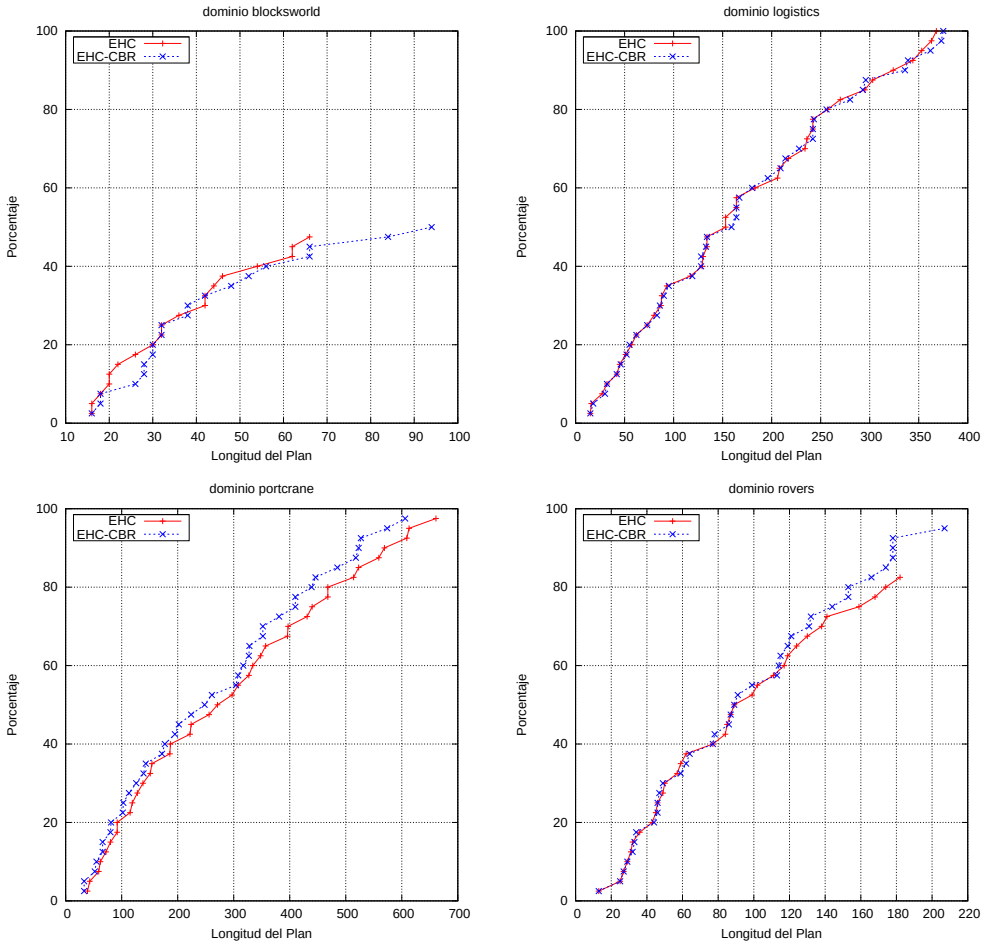


Figura C.5: Distribución acumulada de longitud del plan para experimentos con EHC y EHC+CBR.

C.2.3. WBFS+CBR

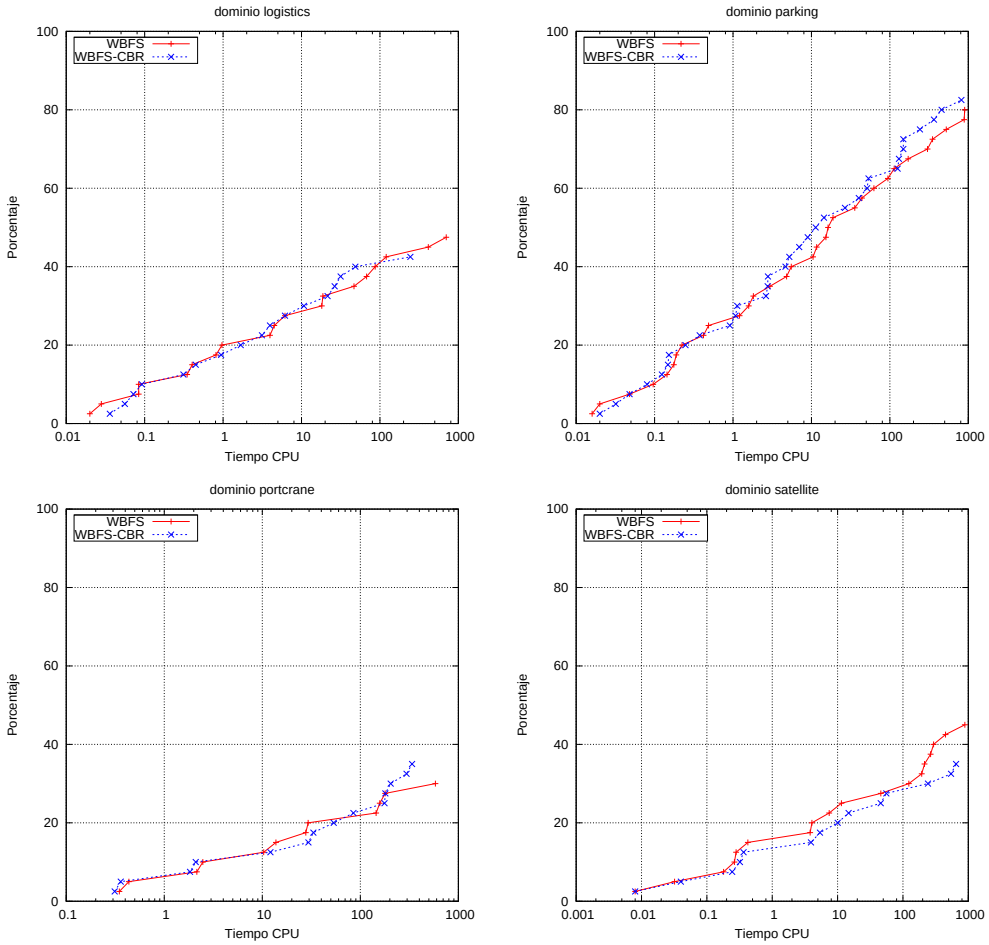


Figura C.6: Distribución acumulada de tiempo de CPU para experimentos con WBFS y CBR.

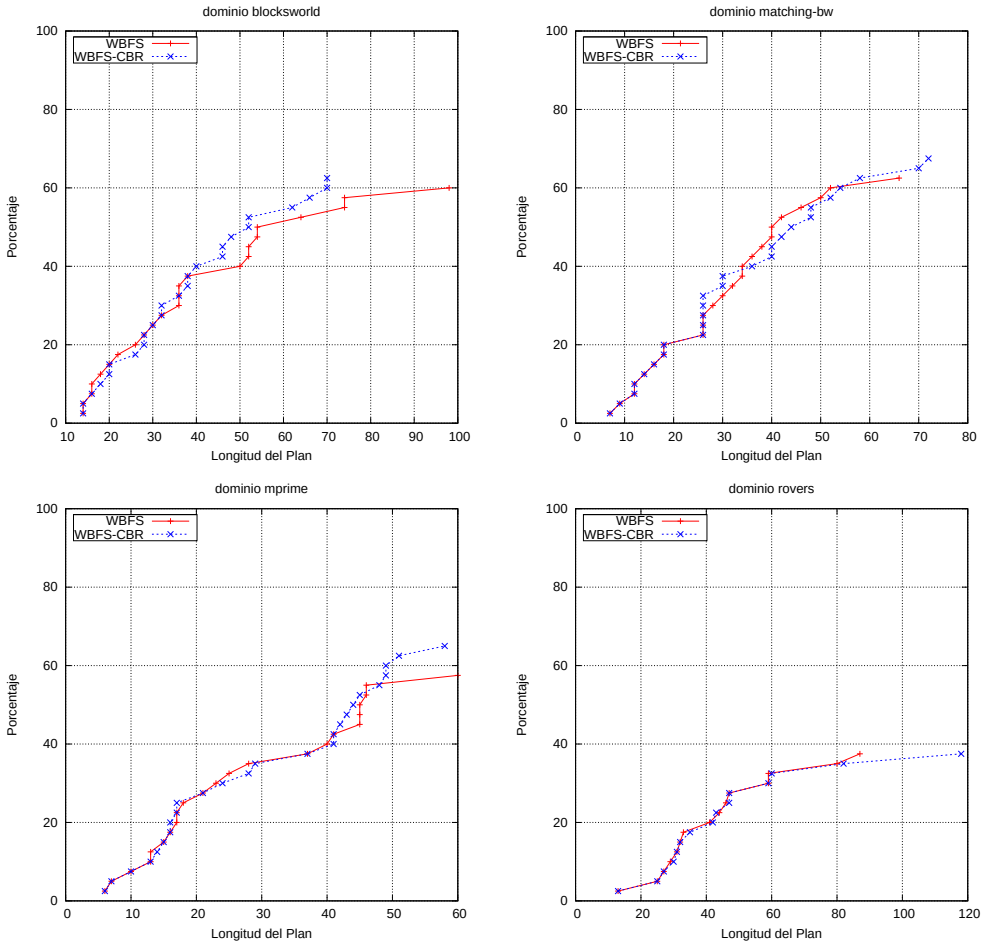


Figura C.7: Distribución acumulada de longitud del plan para experimentos con WBFS y WBFS+CBR.

C.3. Evaluaciones Complementarias

Evaluación de la Generación de las bases de Casos (Escalada+CBR)

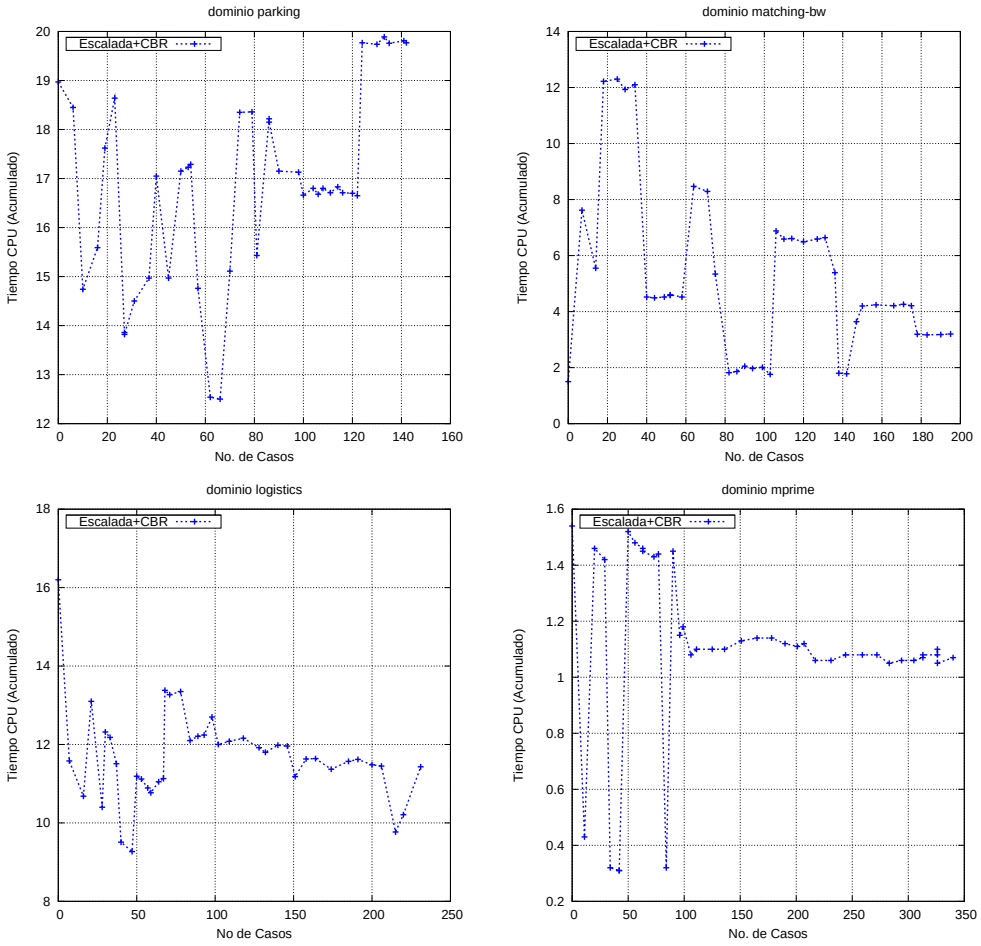


Figura C.8: Tiempo acumulado de Escalada+CBR en resolver los problemas durante las iteraciones de generación de las bases de casos.

Iteración	blo	mbw	par	log	mpr	pcr	rov	sat
0	0	0	0	0	0	0	0	0
1	4	7	6	7	11	6	8	8
2	7	14	10	16	20	11	16	13
3	7	18	16	21	29	18	23	18
4	10	25	19	28	34	23	30	23
5	11	29	23	30	42	28	37	23
6	11	34	27	33	42	33	45	25
7	13	40	27	37	50	34	52	27
8	14	44	31	40	56	37	57	28
9	15	49	37	47	63	40	64	33
10	16	52	40	50	63	43	71	35
11	17	52	45	53	73	43	76	40
12	19	58	50	57	77	46	85	44
13	22	64	53	59	84	47	93	48
14	23	71	54	64	90	47	101	52
15	24	75	57	67	96	47	105	57
16	25	82	62	68	99	47	115	58
17	25	86	66	71	106	50	121	64
18	25	90	70	78	111	50	130	69
19	25	94	74	84	125	50	140	73
20	27	99	79	89	136	50	149	77
21	27	103	81	93	151	55	160	84
22	27	106	86	98	165	61	171	85
23	27	110	86	102	178	68	180	85
24	27	114	90	109	190	73	193	85
25	27	120	98	118	201	74	202	90
26	27	127	100	128	207	81	212	93
27	27	131	104	132	217	87	224	95
28	27	136	106	140	231	92	233	96
29	27	138	108	146	244	97	243	101
30	27	142	111	151	259	101	253	103
31	27	147	114	158	272	104	263	107
32	28	150	116	164	283	109	277	110
33	28	157	120	174	294	112	291	116
34	28	166	122	185	305	115	306	119
35	28	171	124	191	313	116	321	125
36	28	175	130	200	313	119	338	129
37	28	178	133	206	326	123	350	131
38	28	183	135	215	326	125	360	133
39	28	190	141	220	326	130	371	137
40	28	195	142	231	340	133	386	140

Tabla C.2: Número de casos generados en cada iteración para la evaluación de Escalada+CBR.

C.4. Resumen de Resultados

Algoritmo	blo	mbw	par	log	mpr	pcr	rov	sat	Total
Escalada	34	11	38	34	11	37	38	38	241
Escalada (hdiff)	33	13	37	37	12	33	**40**	39	244
Escalada+CBR	37	11	39	34	15	38	36	39	249
Escalada+CBR-V	36	12	38	39	15	37	**40**	39	256
Escalada+CBR (online)	**39**	12	**40**	34	13	**40**	38	**40**	256
EHC	19	4	29	**40**	26	39	38	33	228
EHC (hdiff)	23	6	35	**40**	20	40	**40**	37	241
EHC+CBR	20	3	34	**40**	29	39	39	38	242
EHC+CBR-V	20	7	34	**40**	30	39	**40**	38	248
EHC+CBR (online)	20	5	36	39	30	37	**40**	38	245
WBFS	24	25	31	19	23	12	18	15	167
WBFS (hdiff)	27	27	32	20	27	14	23	13	183
WBFS+CBR	25	27	33	17	26	14	14	15	171
EHC-Lookahead	26	6	**40**	**40**	22	**40**	**40**	**40**	254
EHC-Lookahead+CBR	28	7	37	**40**	22	**40**	**40**	**40**	254
WBFS-Lookahead	32	**30**	39	**40**	33	**40**	38	**40**	**292**
WBFS-Lookahead+CBR	28	28	**40**	**40**	**34**	**40**	39	**40**	289

Tabla C.3: Problemas resueltos por los diferentes algoritmos de evaluación.

Algoritmo	blo	mbw	par	log	mpr	pcr	rov	sat	Total
Escalada	0,25	0,54	0,62	0,38	0,84	0,46	0,85	0,74	0,59
Escalada (hdiff)	0,45	0,65	0,49	0,57	0,91	0,68	0,93	0,84	0,69
Escalada+CBR	0,33	0,55	0,52	0,42	0,74	0,76	0,74	0,88	0,62
Escalada+CBR-V	0,22	0,34	0,43	0,54	0,63	0,69	0,66	0,90	0,55
Escalada+CBR (online)	0,30	0,49	0,47	0,50	0,73	0,72	0,71	0,87	0,60
EHC	0,86	0,91	0,85	0,97	0,86	0,87	0,89	0,97	0,90
EHC (hdiff)	0,79	0,78	0,65	0,96	0,81	0,89	0,85	0,90	0,83
EHC+CBR	0,79	0,93	0,83	0,96	0,90	0,97	0,91	**0,98**	0,91
EHC+CBR-V	0,84	0,64	0,82	0,95	0,89	0,97	0,91	0,97	0,87
EHC+CBR (online)	0,84	0,89	0,83	0,96	0,91	0,89	0,92	0,97	0,90
WBFS	0,95	**0,96**	**0,97**	0,94	**0,94**	0,98	0,97	0,99	0,96
WBFS (hdiff)	**0,98**	0,95	0,94	**0,99**	**0,94**	0,97	**0,99**	**0,98**	**0,97**
WBFS+CBR	0,96	0,94	0,93	0,96	**0,94**	**0,99**	0,97	0,97	0,96
EHC-Lookahead	0,65	0,81	0,77	0,84	0,79	0,89	0,87	0,89	0,81
EHC-Lookahead+CBR	0,72	0,83	0,78	0,82	0,76	0,88	0,87	0,90	0,82
WBFS-Lookahead	0,54	0,73	0,83	0,84	0,83	0,89	0,92	0,86	0,80
WBFS-Lookahead+CBR	0,65	0,78	0,81	0,82	0,84	0,88	0,94	0,89	0,83

Tabla C.4: Puntuaciones de los algoritmos de evaluación según la métrica de calidad de la IPC-6.

Apéndice D

Secuencias de Tipo Generadas

En este apéndice se muestran las secuencias de tipo generadas que son de particular interés para algunos dominios.

D.1. Secuencias de tipo PACKAGE en el *logistics*

```
(((CASE-NO 1)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (((IN1 . 1)) ((AT1 . 1))))
  (SEQUENCE
           (((AT1) (1) NIL 0 (0 . 0))
            ((IN1) (1) (LOAD-AIRPLANE . 468) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-AIRPLANE . 949) 1 (0 . 0)))))

 ((CASE-NO 2)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (((IN1 . 1)) ((AT1 . 1)) ((IN1 . 1)) ((AT1 . 1)) NIL NIL NIL))
  (SEQUENCE
           (((AT1) (1) NIL 0 (0 . 0))
            ((IN1) (1) (LOAD-TRUCK . 234) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-TRUCK . 718) 1 (0 . 0))
            ((IN1) (1) (LOAD-AIRPLANE . 438) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-AIRPLANE . 917) 1 (0 . 0)))))

 ((CASE-NO 3)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (NIL ((IN1 . 1)) ((AT1 . 1)) ((IN1 . 1)) ((AT1 . 1)) ((IN1 . 1)) ((AT1 . 1))))
  (SEQUENCE
           (((AT1) (1) NIL 0 (0 . 0))
            ((IN1) (1) (LOAD-TRUCK . 202) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-TRUCK . 686) 1 (0 . 0))
            ((IN1) (1) (LOAD-AIRPLANE . 422) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-AIRPLANE . 903) 1 (0 . 0))
            ((IN1) (1) (LOAD-TRUCK . 199) 1 (0 . 0))
```

```
           ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
           ((AT1) (1) (UNLOAD-TRUCK . 675) 1 (0 . 0)))))

 ((CASE-NO 4)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (NIL ((IN1 . 1)) ((AT1 . 1)) NIL NIL NIL NIL))
  (SEQUENCE
           (((AT1) (1) NIL 0 (0 . 0))
            ((IN1) (1) (LOAD-TRUCK . 284) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-TRUCK . 760) 1 (0 . 0)))))

 ((CASE-NO 5)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (((IN1 . 1)) ((AT1 . 1)) ((IN1 . 1)) ((AT1 . 1))))
  (SEQUENCE
           (((AT1) (1) NIL 0 (0 . 0))
            ((AT1) (1) (NO-OP . 0) 0 (0 . 0))
            ((IN1) (1) (LOAD-AIRPLANE . 406) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-AIRPLANE . 885) 1 (0 . 0))
            ((IN1) (1) (LOAD-TRUCK . 181) 1 (0 . 0))
            ((IN1) (1) (NO-OP . 0) 0 (0 . 0))
            ((AT1) (1) (UNLOAD-TRUCK . 657) 1 (0 . 0))))))
```

D.2. Secuencias de tipo CONTAINER en el *portcrane*

```
(((CASE-NO 1)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (NIL ((LIFTING2 . 1)) ((LOCATED1 . 1)) ((LIFTING2 . 1)) ((LOCATED1 . 1))))
  (SEQUENCE
     (((FROZEN1 ON-SHIP1) (1 1) NIL 0 (0 . 0))
      ((REEFER-CONNECTED1 FROZEN1 LIFTING2) (1 1 1) (UNLOAD-SHIP-FREEZE . 9) 1 (0 . 0))
      ((REEFER-CONNECTED1 FROZEN1 LIFTING2) (1 1 1) (NO-OP . 0) 0 (0 . 0))
      ((FROZEN1 LOCATED1) (1 1) (DROP-YARD-FREEZE . 65) 1 (0 . 0))
      ((REEFER-CONNECTED1 FROZEN1 LIFTING2) (1 1 1) (LIFT-YARD-FREEZE . 137) 1 (0 . 0))
      ((REEFER-CONNECTED1 FROZEN1 LIFTING2) (1 1 1) (NO-OP . 0) 0 (0 . 0))
      ((FROZEN1 LOCATED1) (1 1) (DROP-YARD-FREEZE . 76) 1 (0 . 0)))))

 ((CASE-NO 2)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (NIL ((LIFTING2 . 1)) ((LOCATED1 . 1)) ((LIFTING2 . 1)) ((LOCATED1 . 1))))
  (SEQUENCE
     (((DRYVAN1 ON-SHIP1) (1 1) NIL 0 (0 . 0))
      ((DRYVAN1 LIFTING2) (1 1) (UNLOAD-SHIP . 3) 1 (0 . 0))
      ((DRYVAN1 LIFTING2) (1 1) (NO-OP . 0) 0 (0 . 0))
      ((DRYVAN1 LOCATED1) (1 1) (DROP-YARD . 49) 1 (0 . 0))
      ((DRYVAN1 LIFTING2) (1 1) (LIFT-YARD . 121) 1 (0 . 0))
      ((DRYVAN1 LIFTING2) (1 1) (NO-OP . 0) 0 (0 . 0))
      ((DRYVAN1 LOCATED1) (1 1) (DROP-YARD . 54) 1 (0 . 0)))))

 ((CASE-NO 3)
  (MERGED T)
  (UTILITY (0 . 0))
  (RX-KEY (NIL ((LIFTING2 . 1)) ((LOCATED1 . 1)) ((LIFTING2 . 1)) ((LOCATED1 . 1))))
```

```
(SEQUENCE
   (((LOCKED1 ON-SHIP1) (1 1) NIL 0 (0 . 0))
    ((OPENTOP-LOCKED1 LOCKED1 LIFTING2) (1 1 1) (UNLOAD-SHIP-LOCK . 16) 0 (0 . 0))
    ((OPENTOP-LOCKED1 LOCKED1 LIFTING2) (1 1 1) (NO-OP . 0) 0 (0 . 0))
    ((LOCKED1 LOCATED1) (1 1) (DROP-YARD-LOCK . 86) 1 (0 . 0))
    ((OPENTOP-LOCKED1 LOCKED1 LIFTING2) (1 1 1) (LIFT-YARD-LOCK . 152) 1 (0 . 0))
    ((OPENTOP-LOCKED1 LOCKED1 LIFTING2) (1 1 1) (NO-OP . 0) 0 (0 . 0))
    ((LOCKED1 LOCATED1) (1 1) (DROP-YARD-LOCK . 90) 1 (0 . 0))))))
```

Apéndice E

El Planificador SAYPHI

El planificador SAYPHI fue desarrollado originalmente para la integración de las técnicas CBR expuestas en esta tesis. En la actualidad es el planificador base para diferentes técnicas de aprendizaje que se aplican en el grupo de Planificación y Aprendizaje de la Universidad Carlos III de Madrid. SAYPHI se distribuye bajo la licencia GPL y está accesible en la web en la dirección

`www.plg.inf.uc3m.es/~trosa/software/sayphi.tgz`

Se ejecuta bajo un entorno LISP (SBCL preferiblemente). A continuación, un ejemplo de los pasos a seguir para resolver un problema de planificación en SAYPHI.

1. Cargar el planificador (desde el directorio raiz):
   `>(load "loader")`
2. Cargar un fichero de dominio PDDL: (carpeta de dominio y fihero PDDL):
   `>(say-domain "satellite" "satellite.pddl")`
3. Cargar un fichero de problema:
   `>(prob "pfile01.pddl")`
4. Ejecutar la búsqueda (EHC y h^{FF} por defecto):
   `>(plan)`
   o según el algoritmo
   `>(plan :algorithm 'hill-climbing)`