UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

APRENDIZAJE DE CONOCIMIENTO DE CONTROL PARA PLANIFICACIÓN DE TAREAS

TESIS DOCTORAL

Susana Fernández Arregui
Leganés, 2006

Departamento de Informática

Escuela Politécnica Superior
Universidad Carlos III de Madrid

APRENDIZAJE DE CONOCIMIENTO DE CONTROL PARA PLANIFICACIÓN DE TAREAS

AUTOR: Susana Fernández Arregui
DIRECTORES: Ricardo Aler Mur
Daniel Borrajo Millán

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día ..... de ..... de 2006.

Presidente: D. .....

Vocal: D. .....

Vocal: D. .....

Vocal: D. .....

Secretario: D. .....

Realizado el acto de defensa y lectura de la Tesis el día ..... de ..... de 2006 en .....

Calificación: .....

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A Oscar y mis hijos, por el espacio que llenan sus presencias.
A Rodrigo, por el vacío que causa su ausencia.

ITACA
(1911)

Si vas a emprender el viaje hacia Itaca,
pide que tu camino sea largo,
rico en experiencias, en conocimiento.
A Lestrigones y a Cíclopes,
o al airado Poseidón nunca temas,
no hallarás tales seres en tu ruta
si alto es tu pensamiento y limpia
la emoción de tu espíritu y tu cuerpo.
A Lestrigones ni a Cíclopes,
ni al fiero Poseidón hallarás nunca,
si no los llevas dentro de tu alma,
si no es tu alma quien ante ti los pone.

Pide que tu camino sea largo.
Que numerosas sean las mañanas de verano
en que con placer, felizmente
arribes a bahías nunca vistas;
detente en los emporios de Fenicia
y adquiere hermosas mercancías,
madreperla y coral, y ámbar y ébano,
perfumes deliciosos y diversos,
cuanto puedas invierte en voluptuosos y delicados perfumes;
visita muchas ciudades de Egipto
y con avidez aprende de sus sabios.

x

Ten siempre a Itaca en la memoria.
Llegar allí es tu meta.
Mas no apresures el viaje.
Mejor que se extienda largos años;
y en tu vejez arribes a la isla
con cuanto hayas ganado en el camino,
sin esperar que Itaca te enriquezca.

Itaca te regaló un hermoso viaje.
Sin ella el camino no hubieras emprendido.
Mas ninguna otra cosa puede darte.

Aunque pobre la encuentres, no te engañará Itaca.
Rico en saber y en vida, como has vuelto,
comprendes ya qué significan las Itacas.

Konstantino Kavafis

Agradecimientos

Muchas son las personas que han hecho de esta tesis un hermoso viaje a Itaca. Empezando por esa primera entrevista con Araceli, embarazadísima como estabas de tu hijo *Juancito*, Pedro, José Manuel y Daniel que me brindastéis la oportunidad de emprender el viaje. Continuando por todos los que me habéis acompañado en el largo camino. Oscar, amado y abnegado esposo de una doctoranda con dos hijos. Javier, que siempre has tenido más fe en mi tesis que yo misma. Agapito, mi incondicional compañero de despacho. Fernando, excelente colega y amigo. Ricardo e Inés, mi primera publicación internacional. Juan Manuel, mi primer compañero de prácticas. Rafaela, siempre solucionando los problemas más complicados. Todos los que antes que yo, Gonzalo, Agustín, Germán, Susana, Ana, Ana Bel ..., habéis vencido a Lestrigones y a Cíclopes y al airado Poseidón enseñándome a no temerles y a disfrutar del recorrido. A todos mis compañeros del grupo PLG que hacéis tan agradable el trabajo diario. Y por supuesto, a los que a más bahías nunca vistas me habéis hecho arribar, hermosas mercancías adquirir y ciudades de Egipto visitar, mis directores Daniel y Ricardo. Gracias, gracias a todos de todo corazón.

Resumen

Esta tesis doctoral se centra en dos disciplinas de la Inteligencia Artificial: la planificación de tareas y el aprendizaje automático.

La planificación en IA desarrolla sistemas de computación que resuelvan problemas cuya solución consista de un conjunto de acciones, total o parcialmente ordenadas, denominado plan que, mediante su ejecución, consiguen pasar de una situación inicial a otra situación en que se hacen ciertas una serie de metas o fines perseguidos. Debido al enorme tamaño del espacio de búsqueda que hay que explorar, se considera un problema *PSPACE-complete* [Bylander, 1991]. Normalmente es necesario definir algún tipo de conocimiento o heurística que permita obtener eficientemente el plan.

Por otro lado, se dice de un programa de ordenador que aprende a partir de la experiencia E, con respecto a alguna clase de tarea T y una medida de rendimiento P, si su rendimiento en la tarea T, medido mediante P, mejora con la experiencia E [Mitchell, 1997].

El objetivo de esta tesis es el desarrollo de sistemas de aprendizaje automático aplicables a planificación, para extraer el conocimiento de control (o heurísticas) adecuado que incremente la eficiencia (disminuya el tiempo y la memoria consumidos por el ordenador) en resoluciones de nuevos problemas. También se busca definir una metodología de aprendizaje adaptable a distintos paradigmas de planificación capaz de obtener heurísticas. Por último, se estudia la transferencia de dicho conocimiento, primero entre dos planificadores distintos; es decir, se extrae el conocimiento de una técnica de planificación específica, utilizando la metodología diseñada, para aplicarlo en otra diferente; y segundo, a los sistemas de aprendizaje: cómo le afecta al rendimiento del sistema de aprendizaje la utilización de un conocimiento previo proveniente de una fuente externa. Para poder realizar estas transferencias se necesita haber definido previamente un lenguaje de representación del conocimiento de control.

Abstract

The research presented in this dissertation focuses on two topics in Artificial Intelligence: planning and machine learning. The aim of classical planning is to find plans to solve problems. A problem is made of a starting situation and a set of goals. Given a set of possible actions, a planner must find a sequence of actions (a plan) that, starting from the initial state, fulfills all goals. Planning has a high computational complexity. In fact, it is PSPACE-complete [Bylander, 1991]. Usually, in order to make the search process more efficient, some heuristics are needed.

On the other hand, a computer program is said to learn from experience E, with respect to some task T and a performance measure P, if its performance on T according to P, improves with experience E [Mitchell, 1997].

This thesis aims to show that machine learning methods can improve efficiency (time and memory) in modern planners, like graphplan-based planning or agent-based hierarchical partial-order planners. From this experience, a general methodology is developed to apply machine learning to different planning paradigms. Finally, this thesis explores the idea of transferring heuristics learned in one planner to a different planner. In particular, we transfer heuristics from a graphplan-based planner to a bidirectional one.

Índice general

Índice de figuras

Índice de tablas

Parte I

Introducción general

Capítulo 1

Introducción

Una de las características de la inteligencia humana es la capacidad de planificar tareas para obtener un fin determinado. Se conoce el objetivo, considerado como un conjunto de metas a satisfacer, y es necesario encontrar las acciones que llevan hasta él partiendo de una situación inicial concreta. Normalmente, las acciones que se pueden escoger para obtener esas metas perseguidas no son únicas, sino alternativas.

El ser humano está constantemente planificando, desde los razonamientos que hace sobre cómo se va a organizar el día, qué tareas tiene que hacer y cómo las va a acometer, hasta el establecimiento de planes específicos de cualquier actividad profesional. Por ejemplo, las empresas elaboran su plan de acción en el que se define la programación de actividades para satisfacer las metas empresariales fijadas; en manufacturación es necesario determinar las acciones que transforman las materias primas en los productos elaborados; los abogados requieren de un plan de defensa de sus clientes, etc.

En general, planificar suele ser una tarea difícil debido principalmente a las siguientes cuestiones: nuestra visión del mundo es incompleta (racionalidad limitada), el mundo cambia constantemente (dinamismo), nuestro modelo del mundo falla muchas veces (incertidumbre), las acciones tardan en ejecutarse (razonamiento temporal), algunas metas pueden tener efectos contrapuestos (dependencia entre metas), los planes no siempre son válidos (ejecución y replanificación) y no todos los planes tiene la calidad deseada. Sin embargo, estas dificultades pueden ser paliadas en parte gracias a la capacidad de aprendizaje del hombre que hace posible una continua adaptación al mundo. Aprender es un proceso cognitivo complejo que abarca numerosos aspectos. Ha sido ampliamente estudiado por disciplinas diferentes, como psicología, pedagogía, biología, filosofía . . . , y existen multitud de definiciones de aprendizaje. Una de ellas es la de Herbert Simon que lo define como “cambios en un sistema que permiten llevar a cabo la misma tarea, u otra tarea en la misma población, de un modo más eficiente y eficaz” [Simon, 1983].

La Inteligencia Artificial (IA), como rama de la ciencia de la computación que se ocupa de la automatización de la conducta inteligente [Luger and Stubblefield, 1993], ha tratado de automatizar ambas capacidades, la de planificación y la de aprendizaje. Fue en los años 60 cuando Alan Newell y Herbert Simon lograron crear un programa llamado GPS (*General Problem Solver*: solucionador general de problemas) que es considerado como el precursor de los sistemas de planificación actuales [Newell and Simon, 1963]. El objetivo de este sistema era obtener de manera automática la solución a cualquier problema expresado en un lenguaje de alto nivel. Desde entonces hasta nuestros días la planificación de tareas ha sido abordada desde perspectivas muy diferentes y, a pesar de todas las dificultades que conlleva, se está llegado a resultados importantes que cada vez nos aproximan más a la resolución de problemas del mundo real. Por ejemplo, la planificación no lineal hace posible resolver problemas en que hay dependencia entre metas, la planificación temporal aquéllos en los que las acciones tienen duración y utilizan recursos, o la planificación con incertidumbre que permite resolver problemas a pesar de que el modelo del mundo falle.

La planificación en IA, en sentido amplio, se puede definir como la búsqueda de un conjunto de acciones, denominado *plan*, que resuelve un problema determinado dentro de un *dominio*. En la definición del problema se especifican tanto el *estado inicial* del que se parte como las *metas* u objetivos que se quieren conseguir. Un *estado* o situación es la descripción instantánea del sistema. El dominio contiene la descripción de todas las acciones posibles que se pueden ejecutar a través de los denominados *operadores*. La solución (plan) determina un conjunto de acciones u operadores (del dominio) necesarias para pasar del estado inicial a un estado en que estén presentes todas las metas. Dichas acciones pueden estar totalmente ordenadas, constituyendo una secuencia única de operadores, o parcialmente ordenadas, en cuyo caso el plan está constituido tanto por secuencias de acciones que necesitan estar ordenadas como por otras acciones que no importa el orden en que se ejecuten. La representación del dominio suele ser mediante predicados lógicos de primer orden, los operadores se definen mediante pares precondición-efecto y normalmente se define algún tipo de conocimiento que permita obtener eficientemente el plan, heurísticas.

Planificar requiere razonar acerca de las consecuencias de las acciones que se van a ejecutar. En general se dispone de un gran número de acciones posibles y cada una de ellas puede dar lugar a múltiples consecuencias, lo que lleva a un complejo problema computacional debido al enorme tamaño del espacio de búsqueda que es preciso explorar y máxime aún si se quiere encontrar la solución óptima. A pesar de los enormes avances conseguidos, todavía se considera intratable la resolución de problemas por medio de planificadores independientes del dominio [Bylander, 1994]; se considera un problema *PSPACE-complete* [Bylander, 1991]. Por esta razón, una de las líneas de investigación que se ha seguido en planificación es la de mejorar los planificadores base utilizando técnicas de aprendizaje automático. Cuando dichas técnicas se utilizan en planificación se pueden perse-

guir tres tipos de objetivos: mejorar la eficiencia y eficacia del planificador, es decir, incrementar el número de problemas resueltos por el planificador y reducir el tiempo y la memoria consumidos en la resolución; intentar mejorar la calidad de los planes obtenidos; y mejorar la propia teoría del dominio. La calidad de un plan es posible calcularla utilizando diferentes criterios tales como: número de operadores en el plan, tiempo que tarda en ejecutarse el plan, coste económico del plan, etc. Se habla de *métricas de calidad* y constituyen una entrada más del planificador.

Capítulo 2

Objetivos de la tesis doctoral

El objetivo general de esta tesis doctoral es la aplicación y desarrollo de técnicas de aprendizaje automático que extraigan conocimiento de control del proceso de planificación para incrementar la eficiencia, disminuyan el tiempo y la memoria, en las resoluciones futuras de nuevos problemas. También se pretende estudiar la posibilidad de generar un marco común de representación de conocimiento de control para poder intercambiar dicho conocimiento entre distintos planificadores. Para ello se pretende analizar las decisiones comunes realizadas por los diferentes algoritmos de planificación para aprender, de la propia experiencia del planificador, las decisiones correctas que llevaron a la solución de los problemas y buscar una forma común de representarlas. Esto implica definir un lenguaje de representación del conocimiento de control que sea directamente utilizable por distintos planificadores o que sea fácilmente adaptable a las particularidades de cada técnica de planificación mediante traductores automáticos. Una vez definido un lenguaje general de representación del conocimiento de control, se persigue definirar una metodología de aprendizaje adaptable a los distintos paradigmas de planificación. En tercer lugar, se desea encontrar las características que deben tener los planificadores para hacer factible la utilización del método de aprendizaje y estudiar el tipo de conocimiento de control susceptible primero de ser extraído y segundo de ser transferido de unos planificadores a otros. Por último, se busca estudiar la influencia de transferir conocimiento externo a los propios sistemas de aprendizaje en planificación: cómo le afecta al rendimiento del sistema de aprendizaje la utilización de un conocimiento previo proveniente de una fuente externa.

Entre los objetivos específicos cabe señalar:

- Definición de un lenguaje común para la representación del conocimiento de control.
- Tomar como referencia HAMLET, el sistema multi-estrategia deductivo-inductivo, que aprende reglas de control para el planificador no lineal de orden total

PRODIGY4.0 y desarrollar un sistema similar aplicable a un planificador híbrido que aplica técnicas de planificación jerárquica y de planificación de orden parcial. Centrado en incrementar la eficiencia de resolución de problemas del planificador, más que en mejorar la calidad de los planes que obtiene.

- Desarrollar un sistema similar al anterior pero aplicado a planificación basada en grafos de planes.
- Estudiar las diferencias y similitudes de los tres sistemas de aprendizaje para definir una metodología de diseño y desarrollo de sistemas de aprendizaje de conocimiento de control basada en la explicación.
- Determinar los ámbitos de aplicabilidad del método de aprendizaje respecto a las características de los planificadores y el conocimiento extraído.
- Estudiar hasta qué punto es posible reutilizar conocimiento de control aprendido en un planificador basado en grafos de planes en otro que utilice otra técnica, como es planificación bidireccional.
- Estudiar las posibilidades de mejorar los propios sistemas de aprendizaje de conocimiento de control mediante la utilización de un conocimiento inicial o "a priori", obtenido de experiencias previas de resolución de problemas, que facilite la tarea de aprendizaje al sistema. Es decir, investigar de qué forma se puede integrar el conocimiento proporcionado por otras fuentes ajenas al propio sistema de aprendizaje.

El documento se ha estructurado en cuatro partes. El resto de esta primera parte presenta el estado del arte, la tecnología utilizada y el lenguaje de conocimiento de control de partida. La segunda parte describe los dos sistemas de aprendizaje implementados en esta tesis doctoral: HEBL y GEBL y la metodología de aprendizaje de conocimiento de control definida. En la tercera parte se explica el estudio experimental realizado sobre la transferencia de conocimiento de control, primero entre distintos paradigmas de planificación y luego a los sistemas de aprendizaje. En la última parte se hace un resumen de las aportaciones de la tesis, las conclusiones finales, los trabajos futuros y las publicaciones realizadas sobre los contenidos de esta tesis. Por último, en los apéndices se describen los dominios de planificación utilizados y los resultados detallados de los experimentos realizados. Con este esquema se pretende separar las aportaciones propias de la tesis doctoral, todo lo descrito a partir de la segunda parte, de los trabajos previos en los que se apoya, lo descrito en esta primera parte.

Capítulo 3

Estado del arte

En este capítulo se presentan las investigaciones relacionadas con el tema de esta tesis doctoral. Primero se hace una introducción sobre los conceptos más importantes en planificación, así como de las principales investigaciones llevadas a cabo sobre el tema. Se ha seguido el enfoque de Geffner en cuanto a la división de los distintos tipos de planificación y en la distinción de tres elementos fundamentales de todo proceso de planificación: lenguajes, modelos matemáticos y algoritmos [Geffner, 2002].

La revisión de la literatura comienza con una introducción a la planificación, sección 3.1. Los siguientes apartados presentan las investigaciones más relevantes según la división entre planificación clásica, sección 3.2, planificación con incertidumbre y realimentación, sección 3.3 y planificación temporal y *scheduling*, sección 3.4. Posteriormente, se presentan las investigaciones relacionadas con la aplicación de aprendizaje automático a la planificación, organizándolos según una de las formas tradicionales de clasificar el aprendizaje en deductivo, sección 3.6, inductivo, sección 3.7 y mixto (utilizan ambas técnicas), sección 3.8. La última sección muestra las conclusiones derivadas del análisis de estos trabajos.

3.1. Introducción a la planificación

La planificación en IA tiene el objetivo de, dado cualquier problema expresado como una situación o estado inicial y unas metas a conseguir, obtener un conjunto de acciones, denominado plan, que mediante su ejecución permitan llegar desde el estado inicial a otro estado en que todas las metas del problema se satisfacen. En este contexto se pueden distinguir tres elementos: lenguajes de representación para describir convenientemente los problemas, modelos matemáticos de la clase de problemas considerada y algoritmos para resolver dichos problemas.

Diferentes suposiciones sobre la naturaleza de las acciones y problemas, y la

presencia de sensores que realimentan los sistemas con sus mediciones, llevan a varias formas de planificación. La **planificación clásica** se centra en problemas cuya solución es un conjunto de acciones deterministas y se tiene una información completa del problema. La **planificación con incertidumbre** resuelve problemas en donde no se tiene una información completa del problema y las acciones no son deterministas. Dentro de ella, se distingue la denominada **planificación conformante** [Smith and Weld, 1998] en que no es posible utilizar sensores para medir el estado actual. La **planificación con tiempo y recursos** o "scheduling" que trata problemas en los que las acciones tienen duración y utilizan recursos. concurrentemente.

Los modelos matemáticos que subyacen a la planificación clásica son los llamados modelos de estados en que las acciones, de forma determinista, transforman un estado en otro y la tarea de planificación consiste en encontrar la secuencia de acciones que pasen del estado inicial a un estado en que estén presentes las metas del problema. Por otro lado, en planificación conformante, las acciones transforman un estado en un conjunto de posibles estados sucesores y la tarea de planificación consiste en encontrar la secuencia de acciones que lleve a conseguir las metas por una cualquiera de las posibles transiciones, desde el estado inicial. Cuando hay incertidumbre en la transición de estados o en el propio estado inicial, la realimentación pasa a ser un factor importante afectando drásticamente la forma de los planes. En ausencia de realimentación, como en planificación clásica y conformante, un plan es una secuencia fija de acciones; sin embargo, en presencia de realimentación, las acciones dependen del estado observado y los planes se convierten en funciones (políticas) que transforman estados en acciones.

3.2. Planificación clásica

Los planificadores desarrollados en este modelo general de planificación asumen que el conocimiento representado en el dominio es un conocimiento completo y correcto. Además, las acciones son deterministas, es decir, sus resultados son predecibles y no hay otros agentes que cambien el entorno.

De los tres elementos requeridos para la resolución de problemas (lenguajes, modelos matemáticos y algoritmos), en planificación clásica el modelo matemático es fijo. Es el denominado modelo de estados y lo que varía entre unos planificadores y otros son los lenguajes de representación y los algoritmos utilizados.

Los lenguajes de planificación se basan en la lógica de predicados haciendo algunas extensiones para poder expresar: el conjunto de acciones posibles, las condiciones bajo las que son aplicables dichas acciones, los efectos que producen, el estado inicial y los estados meta. Uno de los lenguajes más antiguos, simple y más usado en planificación es el definido para el planificador STRIPS [Fikes and Nilsson, 1971]. Un problema en STRIPS es una tupla $< A, O, I, G >$ donde

- A: es el conjunto de literales
- O: es el conjunto de todos los operadores
- I: $\subseteq A$ representa el estado inicial
- G: $\subseteq A$ representa los estados meta

Los operadores $o \in O$ se representan por tres listas:

- $Pre(o)$: $\subseteq A$ precondiciones necesarias para poder ejecutar el operador
- $Add(o)$: $\subseteq A$ literales que pasan a ser verdaderos por la ejecución del operador
- $Del(o)$: $\subseteq A$ literales que pasan a ser falsos por la ejecución del operador

Un problema $P =< A, O, I, G >$ determina un modelo de estados de la siguiente forma:

1. Los estados $s \in S$ se representan mediante los literales de A
2. El estado inicial s_0 es I
3. Los estados meta $s \in S_G$ son aquellos tales que $G \subseteq s$
4. Las acciones $A(s)$ son los $op \in O$ que cumplen $Pre(op) \subseteq s$
5. El estado resultante al aplicar op en s es: $s' = s - Del(op) + Add(op)$ (función de transición de estados)
6. El coste $c(op, s)$ es siempre 1

Una solución para el problema de planificación P es una secuencia aplicable de acciones $a_i, i = 0, ..., n$ que llevan desde el estado inicial s_0 a algún estado meta $s \in S_G$. Es decir, $s_{n+1} \in S_G$ y para $i = 0, ..., n$ $s_{i+1} = f(a_i, s_i)$ y $a_i \in A(s_i)$. La solución óptima es aquélla que minimiza el coste total $\sum_{i=0}^{n} c(a_i, s_i)$

A partir de STRIPS se han desarrollado diferentes lenguajes de planificación, destacando entre todos el lenguaje PDDL debido a que se ha constituido como estándar para la codificación de dominios de planificación. La primera versión fue desarrollada por Drew McDermott, con la ayuda del comité de la competición internacional de planificación de 1998 [McDermott, 2000].

Los algoritmos para resolver modelos de estados son los llamados algoritmos de búsqueda que exploran total o parcialmente el estado de espacios tratando de encontrar un camino (óptimo) que lleve de s_0 a algún $s \in S_G$. Se distingue entre:

1. Búsqueda ciega o algoritmos de fuerza bruta: la meta tiene un papel pasivo en la búsqueda. Por ejemplo, búsqueda en amplitud, en profundidad, hacia atrás, bidireccional ....

2. Algoritmos de búsqueda heurística: se utiliza una función $h(s)$ que estima la "distancia" (coste) desde el estado s hasta S_G para guiar la búsqueda. Por ejemplo, búsqueda en escalada, técnicas de mejor-primero (A*, IDA*) ....

Algunos de los trabajos de investigación realizados sobre problemas de planificación clásica se enumeran a continuación en orden cronológico:

- Algoritmos Strips (años 70): **planificación de orden total** partiendo desde las metas y que realizan encadenamiento hacia atrás. Hacen búsqueda en un **espacio de estados**. Destacan HACKER [AIS, 1974], ABSTRIPS [Sacerdoti, 1974] y PRODIGY [Carbonell *et al.*, 1990]. Los planificadores NOAH [Sacerdoti, 1975] y NONLIN [Tate, 1976] constituyen un punto de inflexión en planificación porque son el origen de las técnicas de **planificación de orden parcial (POP)**, que van construyendo planes parciales definiendo una relación de orden entre sus acciones.

- Planificación de orden parcial (años 80): trabajan en varias submetas a la vez y hacen búsqueda en un **espacio de planes** [Barrett and Weld, 1994, Pednault, 1991]. Introducen los conceptos de enlaces causales y amenazas. Destacan TWEAK [Chapman, 1987], UCPOP [Penberthy and Weld, 1992] (utiliza el lenguaje ADL [Pednault, 1989]), SNLP [McAllester and Rosenblitt, 1991] y UNPOP [McDermott, 1999].

- Planificación basada en grafos de planes (1995): construyen un grafo con todos los planes paralelos posibles hasta una cierta longitud; generan el plan buscando en el grafo desde la meta hacia atrás. Destaca GRAPHPLAN [Blum and Furst, 1995] y alguno de sus descendientes como IPP [Koehler *et al.*, 1997]. La sección 4.5 explica con más detalle este tipo de planificadores.

- Planificación basada en resolución SAT (1996): convierten un problema de planificación en un problema SAT [Kautz and Selman, 1996] y lo resuelven con cualquiera de las estrategias para resolver problemas SAT. Un problema SAT consiste en dada una fórmula en lógica proposicional obtener las asignaciones de valor de verdad a las proposiciones que hacen posible que la fórmula se haga verdad. Aunque los problemas SAT sean intratables (son problemas NP-completos), en la práctica se puede encontrar eficientemente una solución para muchos de ellos. Como ejemplo, el planificador BLACKBOX unifica planificación SAT y planificación basada en grafos de planes [Kautz and Selman, 1999].

- Planificación con búsqueda heurística (1997): hacen una búsqueda en el espacio de estados utilizando una función heurística h que se extrae del problema P. El espacio de estados se puede explorar hacia adelante (planificación progresiva) o hacia atrás (planificación regresiva). Usan algoritmos de búsqueda, como escalada, A^*, IDA^* ... La función heurística es el componente crítico de estos sistemas y se puede extraer automáticamente de la representación. Se tienen los planificadores HSP [Bonet and Geffner, 2001b], GRT [Refanidis and Vlahavas, 1999], FF [Hoffmann, 2001], MIPS [Edelkamp and Helmert, 2000] y STAN [Fox and Long, 2001] (los dos últimos también usan otras técnicas). El sistema ALTALT combina planificación basada en grafos y búsqueda heurística [Nigenda *et al.*, 2000].

Otra línea de investigación importante en planificación clásica la constituyen los **planificadores jerárquicos**. Con la definición de un modelo de planificación jerárquica se pretende conseguir dos objetivos: adaptar el proceso de planificación a la manera en que los humanos resolvemos la mayoría de los problemas complejos y reales, y reducir la complejidad computacional. En esta línea destacan los trabajos sobre Redes de Tareas Jerárquicas (HTN) [Erol *et al.*, 1994a, Knoblock, 1996, Wilkins, 1984], con los planificadores O-PLAN [Currie and Tate, 1991] y UMCP [Erol *et al.*, 1994b]; Jerarquías de Abstracción [Knoblock, 1993, Sacerdoti, 1974, Yang *et al.*, 1996], como el planificador ABSTRIPS [Bacchus and Yang, 1994]; descomposición basada en orden parcial (descomposición-POP) [Fox, 1997, Young *et al.*, 1994], como el planificador SHOP [Nau *et al.*, 1999] y sistemas de planificación en aplicaciones reales [Knoblock, 1996, Tate *et al.*, 1994, Wilkins, 1984], como los planificadores HYBIS [Castillo *et al.*, 2000] y PANDA [Biundo and Schattenberg, 2001, Schattenberg and Biundo, 2002], ambos sistemas híbridos HTN-POP, y el planificador SIADEX [Castillo *et al.*, 2006]. Hay aproximaciones que resuelven problemas HTN convirtiéndolos a problemas SAT [Mali, 1999, Mali and Kambhampati, 1998] y otros combinan HTN con GRAPHPLAN [Lotem *et al.*, 1999].

Por último, mencionar la **planificación deductiva** que concibe la planificación como programación con la construcción de fórmulas y teoremas [Green, 1969, Manna and Waldinger, 1980].

Es importante resaltar que todas estás técnicas de planificación se han combinado entre sí generando planificadores difíciles de encasillar en sólo una de las categorías anteriores.

3.3. Planificación con incertidumbre y realimentación

La planificación con incertidumbre se aplica para resolver problemas de control en los que se tienen unas metas que se quieren cumplir y unos agentes capaces

de realizar acciones. Son problemas en los que no se tiene un conocimiento determinista "a priori" de las acciones de los agentes; una acción no siempre produce el mismo efecto en el sistema y hasta que no se ejecuta la acción no se sabe el efecto que producirá. Cuando para conocer el estado del sistema hay sensores y las acciones ya no sólo dependen del estado actual, sino también de las medidas de dichos sensores, se habla de planificación con realimentación.

Los modelos matemáticos usados en estos problemas son:

- Procesos de decisión de Markov (MDP) [Howard, 1990]: son modelos no deterministas, donde los resultados de las acciones no son completamente predecibles, pero sus estados son totalmente observables. Una característica fundamental de los modelos de Markov es que las acciones del agente dependen sólo del estado actual del entorno, y no de acciones pasadas. Son una generalización de los modelos de estados en que la función de transición pasa a ser probabilística y representa la probabilidad de que al aplicar una acción en un estado se pase a otro estado.

- MDP parcialmente observables (POMDP) [Astrom, 1965]: son modelos no deterministas y sólo son observables parcialmente. Son una generalización de los MDP en los que los estados, al no ser completamente observables, pasan a ser estados de creencia. Los POMDP requieren además del modelo de sensores, esto es, el conjunto de observaciones después de cada acción con una probabilidad asociada. Las soluciones, en vez de ser secuencias de acciones, son funciones (políticas) que determinan la acción a realizar basándose en el último estado de creencia. Las políticas óptimas minimizan los costes esperados.

Los sistemas de planificación con incertidumbre o con conocimiento incompleto se clasifican según el grado de observación del entorno por parte de los agentes, dando lugar a tres clases:

- Planificación con observabilidad total: dentro de estos sistemas se puede incluir la denominada **planificación reactiva**, donde no es necesaria una representación simbólica del mundo externo; basta con diseñar procesos simples, cada uno con su propio bucle de percepción-reacción, y combinar su ejecución adecuadamente. Los trabajos iniciales sobre reacción [Georgeff, 1987, Godefroid and Kabanza, 1991, Schoppers, 1987] han servido para establecer las bases de los sistemas multiagentes o de inteligencia artificial distribuida [Kokkinaki and Valavanis, 1995, Wooldridge and Jennings, 1994].

- Planificación con observabilidad parcial: asumen la existencia de conocimiento generado en el exterior que debe ser detectado. Según la técnica seguida para llevar a cabo esta detección, se distinguen tres aproximaciones generales: **planificación y ejecución entrelazadas (PEE)**

[Ambros-Ingerson and Stell, 1996, Golden *et al.*, 1994, Olawsky and Gini, 1990, Olawsky *et al.*, 1995], se tienen los planificadores IPEM [Ambros-Ingerson and Stell, 1996], BUMP [Olawsky and Gini, 1990], XII [Golden *et al.*, 1994], SAGE [Knoblock, 1995]; **replanificación asíncrona** [Cross and Walker, 1994, Currie and Tate, 1985, Wilkins, 1984] y **planificación contingente o condicional** con los planificadores WARPLAN-C [Warren, 1974], SNLP [McAllester and Rosenblitt, 1991], SENSEP [Etzioni *et al.*, 1992], BURIDAN [Kushmerick *et al.*, 1995],C-BURIDAN [Draper *et al.*, 1993] y CASSANDRA [Pryor and Collins, 1996].

- Planificación con observabilidad nula: constituye la ya mencionada planificación *conformant*. Se aplica en sistemas complejos en los que no se puede conocer el estado exacto del sistema porque no es posible la observación y los efectos de las acciones pueden ser muy diferentes y altamente indeseables si se ejecutan en un estado u otro. Por ejemplo, en las naves espaciales, los sensores para medir el estado interno son mínimos debido a las restricciones de energía y peso producidas en el espacio. Uno de los primeros planificadores de este tipo es POSPLAN y su versión jerárquica POSPLAN* [Pereira *et al.*, 1997]. Otros sistemas que tratan este tipo de planificación son CGP [Smith and Weld, 1998], CMBP [Cimatti and Roveri, 2000], GPT [Bonet and Geffner, 2001a], HSCP [Bertoli *et al.*, 2001] y el trabajo de Kurien, Nayak y Smith [Kurien *et al.*, 2002].

3.4. Planificación con tiempo y recursos (scheduling)

La planificación con tiempo y recursos se aplica para resolver problemas cuyas acciones pueden tener duración y recursos. Se pueden tratar como problemas Strips, añadiendo a las acciones a una **duración** $D(a)$ y unos **recursos** $R(a)$; es decir, un recurso puede estar siendo utilizado por una acción o puede estar libre. Dos acciones a y a' son **mutex** si necesitan un recurso común o interactúan destructivamente (los efectos de una borran una de las precondiciones de la otra). Un *schedule* es un conjunto de acciones junto con los instantes de tiempo en que empiezan y terminan cada una. Por *scheduling* se entiende la asignación de recursos y tiempos de inicio de las actividades, obedeciendo a las restricciones temporales de las actividades y las limitaciones de capacidad de los recursos compartidos. Scheduling es también una tarea de optimización donde recursos limitados se disponen a lo largo del tiempo entre actividades que se pueden ejecutar en serie o en paralelo (plan paralelo) de acuerdo con el objetivo de, por ejemplo, minimizar el tiempo de ejecución de todas las actividades, *makespan*.

Los modelos matemáticos usados son:

- Problema de satisfacción de restricciones (CSP) [Tsang, 1993]. Modelos ba-

sados en variables y restricciones. Un CSP se puede definir como:

$$CSP =< Variables, Dominios, Restricciones >$$

Una solución es la asignación de valores del dominio a las variables, de manera que satisfagan todas las restricciones. Un CSP es consistente si tiene una o varias soluciones.

- Problema de optimización de restricciones (COPs) [Dechter, 2003]. Iguales que los anteriores pero con una función de optimización.
- Problema de satisfacción de restricciones dinámico (DCSP). Generalización de los CSP en los que las variables tienen etiquetas de actividad. Inicialmente sólo están activas un subconjunto de variables y el objetivo es encontrar una asignación válida para las variables activas. También contiene un conjunto de *restricciones de actividad* de la forma: si una variable x toma el valor v_x, entonces las variables y, z, w ... pasan a estar activas.

El lenguaje PDDL2.1 es una extensión de PDDL para poder representar las características temporales de los dominios [Fox and Long, 2002b]. En el nivel 3 se incorporan acciones durativas sin efectos permanentes y en el nivel 4 acciones durativas con efectos permanentes.

Entre la literatura, existen varios planificadores capaces de razonar con algún modelo de tiempo. El planificador IXTET (*Indexed Time Table*) genera un plan de forma similar a un planificador de orden parcial pero añadiendo restricciones temporales, como la mínima y máxima duración esperada de las acciones o la sincronización entre los eventos externos [Ghallab and H., 1994]. Representa el tiempo como un punto, que simboliza un instante, en contraste con HSTS que lo representa como un intervalo, con un punto inicial y un punto final, representando duración [Muscettola, 1994]. Otros planificadores de orden parcial capaces de manejar diversos tipos de restricciones temporales son ZENO [Penberthy and Weld, 1994] y PARCPLAN [Liatsos and Richards, 1999]. En planificación en el espacio de estados tenemos los sistemas TLPLAN [Bacchus and Ady, 2001], SAPA
[Do and Kambhampati, 2001b] y TP4 [Haslum and Geffner, 2001] que incorpora heurísticas para mejorar su eficiencia, utilizando IDA* como algoritmo de búsqueda. A partir de GRAPHPLAN se han desarrollado varios trabajos que incorporan tratamientos temporales, como TGP [Smith and Weld, 1999], RIPP [Koehler, 1998] y TPSYS [Garrido and Long, 2004, Garrido and Onaindía, 2003]. El sistema GP-CSP convierte automáticamente un problema de GRAPHPLAN en un problema CSP y usa *resolvedores* estándar de CSP para encontrar la solución
[Do and Kambhampati, 2001a]. Para mejorar su eficacia incorpora técnicas de aprendizaje deductivo que se explican en la siguiente sección. Esta idea también se aplica en CPLAN; trata problemas de planificación complejos, que puedan tener incertidumbre, restricciones y acciones concurrentes, transformándolos en un problema CSP [van Beek and Chen, 1999]. Tiene el inconveniente de que requiere de

mucha información que debe ser introducida a mano por un humano. El planificador LPSAT trata restricciones temporales combinando SAT y programación lineal [Wolfman and Weld, 1999].

La NASA, motivada por la idea de crear robots que puedan explorar el espacio, ha llevado a cabo varias investigaciones sobre planificación temporal, a través de su *Remote Agent Experiment* (RAX) [Muscettola, 1994, Muscettola *et al.*, 1998]. También, la Agencia Espacial Europea tiene el sistema OPTIMUN AIV que aplica planificación temporal para la nave espacial AIV [Aarup *et al.*, 1994].

El sistema TALPLANER [Kvarnström and Doherty, 2000] está inspirado en los trabajos previos de Bacchus&Kabanza [Bacchus and Kabanza, 1996] y Weld&Etzioni [Weld and Etzioni, 1994] donde las metas son expresadas como fórmulas temporales. Utiliza TAL, una lógica lineal temporal [Doherty *et al.*, 1998], para razonar sobre las acciones y los cambios en dominios dinámicos.

El planificador IPSS integra planificación y *scheduling* en un único sistema [R-Moreno, 2003]. En IPSS el razonamiento se divide en dos niveles. El planificador se encarga de la selección de acciones (puede optimizar según una métrica de calidad diferente al tiempo o recursos) y el *scheduler* se encarga de la asignación del tiempo y los recursos. Durante el proceso de búsqueda, cada vez que el planificador decide aplicar un operador, consulta al *scheduler* para comprobar su consistencia temporal y de recursos. Si es inconsistente el planificador vuelve hacia atrás y genera otra secuencia de acciones. Otros trabajos que integran planificación y *scheduling* son [Garrido and Barber, 2001, Muscettola, 1994, Myers *et al.*, 2001].

3.5. Aprendizaje aplicado a planificación

En los sistemas de planificación hay, al menos, tres oportunidades para aprender:

1. Aprender antes de que empiece la planificación.
2. Aprender durante la ejecución de un plan.
3. Aprender después del proceso de búsqueda de un plan válido.

Los trabajos que automáticamente extraen conocimiento del dominio a través del análisis de los dominios entran dentro de la primera aproximación. La mayoría de los trabajos de aprendizaje en planificación se enmarcan en la tercera categoría, aprendizaje posterior al propio proceso de planificación. Hacen uso de su propia experiencia resolviendo problemas de planificación para mejorar tanto la eficiencia como la calidad de los planes generados en futuras ejecuciones. El segundo tipo de aprendizaje se aplica en sistemas que tratan problemas del mundo real donde hay incertidumbre e información incompleta y necesitan la iteración con los humanos.

Las técnicas de aprendizaje automático se pueden clasificar de muchas maneras. Quizá la clasificación más extendida y general sea:

- Aprendizaje inductivo: se pretende aprender un concepto meta o función a partir de un conjunto de ejemplos, D, que pueden ser positivos, si satisfacen el concepto, o negativos, en caso contrario. Una de las cuestiones más importantes es el lenguaje utilizado para representar cada instancia; suele ser en forma de tupla con sus atributos y unos valores aunque se ha generalizado a representaciones basadas en lógica de predicados. Hay que obtener una de las hipótesis h de todo el espacio de hipótesis H que sea consistente con los ejemplos.
- Aprendizaje deductivo: además del espacio de hipótesis y del conjunto de entrenamiento del aprendizaje inductivo, se tiene una teoría del dominio B que puede usarse para explicar los ejemplos de entrenamiento. La salida deseada es la hipótesis h de H que sea consistente tanto con el conjunto de ejemplos de entrenamiento D, como con la teoría del dominio B.

A continuación se comentan aquellos trabajos relacionados con el aprendizaje de conocimiento para planificación y resolución de problemas. Una clasificación general de los trabajos más relevantes se puede encontrar en [Zimmerman and Kambhampati, 2003].

3.6. Aprendizaje deductivo en planificación

Los métodos deductivos explican y analizan un único ejemplo de un concepto para aprender descripciones más eficientes del concepto. Una vez que se tiene una explicación de porqué una instancia es un ejemplo de un concepto se realiza una generalización para poder aplicarla a la clasificación eficiente de futuras instancias diferentes de ese concepto. Requieren normalmente un conocimiento completo y correcto del dominio.

En planificación, las aproximaciones más utilizadas han sido: aprendizaje de macro-operadores, aprendizaje de secuencias de sub-objetivos, aprendizaje basado en la explicación (EBL), y aprendizaje de la teoría del dominio (aunque este último, no necesariamente se deba realizar sólo mediante aprendizaje deductivo).

3.6.1. Aprendizaje de macro-operadores

Los intentos iniciales para mejorar la eficiencia de la resolución de problemas se basaban en aprender macro-operadores [Fikes *et al.*, 1972, Iba, 1989, Korf, 1985]. Los macro-operadores son secuencias de operadores de planificación que pueden ser reutilizados en futuros casos de resolución de problemas, con la esperanza de

que permitirán alcanzar la solución de manera más rápida. Estas macro-secuencias pueden ser útiles en aquellos problemas que puedan ser descompuestos en submetas no serializables. Las submetas pueden ser: independientes, si la solución óptima se obtiene al concatenar soluciones óptimas de las submetas; serializables, si existe un cierto orden de las submetas que resuelven el problema sin violar una submeta resuelta con anterioridad; y no serializables, si se necesitan violar soluciones a metas anteriores para resolver las nuevas metas. La utilización de macro-operadores reduce la profundidad del árbol de búsqueda pero hace que crezca el factor de ramificación por el incremento de operadores del dominio. Esto hace que el beneficio de los macro-operadores decaiga al incrementar su número.

3.6.2. Aprendizaje de secuencias de sub-objetivos

STEPPINGSTONE aprende secuencias de sub-objetivos que actúan como hitos entre el estado inicial y las metas [Ruby and Kibler, 1989]. A diferencia de los macro-operadores, consigue decrecer la distancia entre el estado inicial y las metas sin incrementar el factor de ramificación. Aprende cuando el planificador base (medios-fines) falla. Los autores también proponen el sistema EAS como una generalización de STEPPINGSTONE, en la que se aprenden episodios (casos) de aquellos problemas que, además de haber sido solucionados, se podría mejorar la calidad de la solución encontrada [Ruby and Kibler, 1992].

3.6.3. Aprendizaje basado en la explicación (EBL)

Existen métodos de aprendizaje que aprenden heurísticas que permiten a un planificador independiente del dominio explorar de manera eficiente en su espacio de búsqueda. Estas heurísticas ayudan al planificador a tomar mejores decisiones (por ejemplo, a elegir un operador). El conocimiento contenido en las heurísticas es local, restringido a un punto de decisión particular. Las heurísticas intentan que el planificador llegue antes a la solución, ordenando las alternativas del árbol de búsqueda de planificación de una manera eficiente, o realizando podas del árbol de búsqueda. La ventaja frente a los macro-operadores es que no incrementan el factor de ramificación, pero tienen la desventaja de que no decrementan el número de pasos que hay desde el estado inicial a las metas. EBL intenta aprender a partir de uno o pocos ejemplos, basándose en observaciones sobre el aprendizaje humano [Mitchell *et al.*, 1986]. Las entradas a EBL constan de: el concepto que se quiere aprender, el ejemplo de aprendizaje, la teoría del dominio y un criterio de operatividad

Existe una gran variedad de sistemas que utilizan EBL para aprender conocimiento de control para planificadores. Por ejemplo, el sistema PRODIGY4.0-EBL de aprendizaje para el planificador PRODIGY2.0 usa EBS (*Explanation Based Specialization*), una variedad de EBL [Minton, 1988]. El sistema aprende reglas de

control en respuesta a decisiones correctas e incorrectas del planificador, siguiendo el mecanismo descrito en los párrafos anteriores. Dicho trabajo introdujo además el importante concepto de **utilidad**: en ocasiones añadir más conocimiento de control puede ser contraproducente. La razón es que este conocimiento de control hay que equipararlo en cada nodo que se expande para comprobar si es aplicable. Puesto que cada trozo del conocimiento de control será aplicable sólo en algunas ocasiones, puede ocurrir que el coste de equiparación del conocimiento de control en todos los nodos expandidos supere a los beneficios de aplicar el conocimiento (en aquellos casos en que sea aplicable). Un problema similar ocurre con los macro-operadores [Tambe *et al.*, 1990]. Minton intenta resolver este problema de dos maneras distintas. La primera, denominada compresión, transforma las reglas de control sin modificar su significado, pero haciendo su equiparación más eficiente. En segundo lugar, olvida aquellas reglas cuya utilidad es negativa. El sistema ULS utiliza evaluación empírica de las reglas obtenidas para determinar si algunas condiciones son frecuentemente redundantes [Chase *et al.*, 1989]. Aquellas que lo son, se podan, lo que hace que las reglas obtenidas sean menos costosas de aplicar.

Otros trabajos muestran que si el conocimiento de control se organiza de manera adecuada, el problema de la utilidad es mucho menos grave [Doorenbos and Veloso, 1993]. Existen también extensiones a EBL para aprender conceptos recursivos e iterativos [Shavlik, 1990]. El sistema FAILSAFE aprende reglas de control a partir de fallos, en lugar de sólo ejemplos de resolución de problemas exitosos [Bhatnagar, 1992, Bhatnagar and Mostow, 1994, Mostow and Bhatnagar, 1987]. Uno de los problemas de EBL es que en dominios complejos puede ser muy complicado obtener trazas de problemas resueltos. FAILSAFE intenta resolver este problema generando planes sin tener en cuenta ciertas restricciones que impone el dominio y comprobando posteriormente las restricciones e identificando porqué falla el plan. Finalmente, se utiliza EBL para identificar y evitar esa clase de fallos en el futuro. Estos trabajos sólo incrementan la eficiencia y eficacia de los planificadores, en contraste con QUALITY [Pérez, 1995] y el sistema de aprendizaje propuesto por Iwamoto [Iwamoto, 1994] encaminados a mejorar la calidad de las soluciones encontradas por el planificador PRODIGY.

Los sistemas anteriores aprendían conocimiento de control para planificadores de orden total (búsqueda en el espacio de estados o de problemas). UCPOP+EBL aprende reglas de control gracias al análisis de fallos previos durante el proceso de planificación para planificadores de orden parcial [Kambhampati *et al.*, 1996]. Este sistema analiza los caminos de la traza de un problema de planificación. Estudia tanto aquellos caminos que no se pudieron continuar como aquellos que superaron una profundidad límite. Para ello usa conocimiento sobre el dominio adicional basado en leyes físicas del dominio. SNLP+EBL también utiliza EBL con un planificador de orden parcial [Katukam and Kambhampati, 1994]. Otros ejemplos de sistemas que utilizan EBL en POP para incrementar la calidad de los planes son PIPP [Upal, 1999] y PYRRHUS [Williamson and Hanks, 1994].

Los sistemas basados en EBL necesitan teorías completas, correctas y tratables, pero no siempre es posible cumplir estas condiciones al mismo tiempo. El caso del ajedrez presenta una teoría que es completa pero intratable: aunque disponemos de todos los posibles movimientos, generar demostraciones en ajedrez es un problema combinatoriamente intratable, aunque sólo sea porque se desconoce cual va a ser el movimiento del contrario (con lo que, habría que probar todos los posibles movimientos). LEBL (*Lazy Explanation Based Learning*) intenta solucionar este problema, aprendiendo conocimiento que es sólo válido de manera aproximada, y refinándolo posteriormente cuando se presenten fallos, tras aprender el fallo por medio de EBL [Tadepalli, 1989].

El sistema GP-CSP [Do and Kambhampati, 2001a] resuelve un problema de planificación convirtiéndolo automáticamente a un problema CSP y utilizando la librería creada por Beek [Beek, 1994] para resolver problemas CSP. Además, utiliza EBL para ayudar a explicar los fallos encontrados durante la búsqueda para evitar incurrir en ellos en futuras ejecuciones. Los fallos se almacenan como asignaciones de variable-valor con el significado semántico de que no pueden formar parte de la solución. Para decidir cuántas de estas explicaciones se deben almacenar se pueden usar dos técnicas: aprendizaje basado en el tamaño [Frost and Dechter, 1994] o aprendizaje basado en la relevancia [Bayardo and Miranker, 1996]. En el aprendizaje basado en el tamaño de grado k se ignoran todas las explicaciones que contengan más de k variables y valores. En el aprendizaje basado en la relevancia de grado k se ignoran aquellos esquemas que difieran de la asignación parcial actual en más de k asignaciones.

3.6.4. Aprendizaje por análisis del dominio

Existen sistemas que realizan un análisis previo de la teoría del dominio para extraer conocimiento. Entre ellos se encuentra STATIC que obtiene reglas de control tras analizar la descripción del dominio (los operadores) sin necesidad de utilizar ejemplos [Etzioni, 1993]. Este sistema también usa EBL para analizar las relaciones entre precondiciones y efectos de los operadores que son luego convertidos en reglas de control para ordenar las decisiones relativas a objetivos de PRODIGY4.0. Sin embargo, al no utilizar ejemplos, parte del conocimiento puede ser inútil si nunca va a ser utilizado en la distribución de problemas que se presenten al planificador. En otras palabras, puede aparecer el problema de la utilidad. Por ello, se diseñó un nuevo sistema (DYNAMIC) que combina las ventajas de EBL y STATIC [Pérez and Etzioni, 1992]. Otro sistema de este tipo es STAN que analiza los dominios buscando simetrías [Fox and Long, 2001]. Estas simetrías pueden ser de dos tipos: de los objetos del dominio y de las acciones. Un ejemplo sencillo de la utilidad de las simetrías en planificación es que si se dispone de una caja con 500 chinchetas, el que se use una u otra chincheta es irrelevante. Si coger una chincheta provoca que el plan falle, cualquier otra chincheta tendrá el

mismo efecto. STAN aprovecha la simetrías de esta manera para el planificador GRAPHPLAN, evitando elegir una acción simétrica de otra que falló previamente. STAN se ve ayudado por otra herramienta denominada TIM que permite encontrar automáticamente tipos de objetos (esto es, conjuntos de objetos del dominio que se comportan de manera similar) [Long and Fox, 1999]. Esta información es útil por sí misma, y sirve además para establecer simetrías no sólo de objetos sino también de tipos de objetos. El sistema SYMMETRICSTAN es una extensión de STAN que tiene en cuenta el hecho de que en muchos problemas de planificación las simetrías no se presentan en el estado inicial sino que pueden aparecer durante el proceso de búsqueda [Fox and Long, 2002a]. Huffman, Pearson y Laird [Huffman *et al.*, 1992] hacen una análisis de la teoría del dominio basándose en la premisa de que casi todas las descripciones de dominios son aproximaciones debido a la dificultad para describir completamente las precondiciones y efectos de los operadores [Huffman *et al.*, 1992]. Detectan diferentes imperfecciones en las definiciones de los operadores: precondiciones demasiado generales o demasiado específicas, efectos incompletos o extraños y operadores que faltan y deberían existir. RIFO es un método que puede detectar automáticamente información irrelevante en planificadores basados en el espacio de estados [Nebel *et al.*, 1997]. Una información se considera irrelevante si no es necesaria para obtener el plan que constituye la solución. El sistema se deshace de esta información en un pre-proceso inicial, antes de empezar a resolver el problema.

3.7. Aprendizaje inductivo en planificación

Desgraciadamente, los sistemas deductivos requieren teorías del dominio correctas, completas y tratables (aunque existen trabajos basados en EBL que aprenden a partir de teorías incompletas e intratables [Tadepalli, 1989]). Construir dichas teorías no siempre es fácil. Lo que es peor, aunque las teorías sean completas, pueden carecer de información heurística. Por ejemplo, pueden no contener información acerca de qué decisión es más eficiente tomar en un determinado momento, de entre un conjunto de decisiones correctas. Unido a lo anterior, puesto que los sistemas deductivos aprenden de tan sólo unos pocos ejemplos, tienden a producir reglas demasiado específicas, con muchas precondiciones, que sufren del problema de la utilidad [Etzioni and Minton, 1992, Minton, 1988]. Los sistemas inductivos intentan resolver estos problemas aprendiendo a partir de un conjunto de ejemplos. Dichos ejemplos son instancias de decisiones correctas e incorrectas tomadas por un planificador al resolver un determinado conjunto de problemas. Los sistemas inductivos no requieren de teorías completamente correctas, puesto que en ocasiones aprender una heurística incorrecta no significa que el planificador tenga que fallar, sino que simplemente puede llevarle por un camino incorrecto por el que tenga que retroceder posteriormente. Esto podría compensarse si en otras muchas ocasiones, la aplicación de la heurística es correcta [Leckie, 1993]. Los sistemas deductivos no

podrían aprender este tipo de conocimiento, puesto que sólo aprenden heurísticas correctas. Además, los sistemas inductivos pueden aprender reglas muy generales, puesto que examinan muchos ejemplos. La mayoría de los trabajos de investigación que utilizan aprendizaje inductivo en planificación se combinan con alguna otra técnica de aprendizaje deductivo dando lugar a los sistemas multi-estrategia que se detallan en la siguiente sección. A continuación, se detallan algunas de las investigaciones que utilizan sólo aprendizaje inductivo.

3.7.1. Aprendizaje basado en el Espacio de Versiones (EV)

Como se dijo anteriormente, el aprendizaje inductivo trata de obtener la hipótesis h de todo el espacio de hipótesis H que sea consistente con los ejemplos proporcionados, tanto positivos como negativos. El espacio de versiones es el conjunto de todas las hipótesis que son consistentes con los datos. En el contexto de planificación es el conjunto de todas las posibles explicaciones del concepto que se quiere aprender. Un primer algoritmo para encontrar el EV a partir de unos ejemplos consiste en generar todas las posibles hipótesis y luego borrar aquéllas que describan algún ejemplo negativo o no describan algún ejemplo positivo. El problema de este algoritmo es que es muy poco eficiente por la cantidad de posibles hipótesis que se pueden crear. Por eso se han desarrollado diferentes algoritmos para resolver este problema. Entre ellos se encuentra el de eliminación de candidato (*candidate elimination*), introducido en [Mitchell, 1977]. El concepto a aprender se representa mediante dos conjuntos: el conjunto G que contiene los predicados más generales posibles que explican el concepto y el conjunto S con los más específicos. Los conceptos contenidos entre estos dos límites pertenecen al EV y son posibles explicaciones para el concepto a aprender. Con un número suficiente de ejemplos se puede llegar a que estos dos conjuntos sean iguales, siguiendo un algoritmo incremental. Cada vez que llega un ejemplo nuevo se adaptan G y S para cubrirlo. El sistema CAMEL utiliza este algoritmo para aprender métodos de expansión en un planificador HTN [Ilghami *et al.*, 2002], aplicado a dominios militares. CAMEL aprende de manera incremental las condiciones de los métodos. La ventaja de que sea incremental es que no es necesario almacenar todos los ejemplos de entrenamiento. En cuanto recibe uno, adapta el espacio de versiones y se deshace de él. Cada método tiene su correspondiente EV y cada miembro del EV de un método es una posible precondición para dicho método. Para generar los ejemplos de entrenamiento se requieren trazas de problemas resueltos. Cuando el método utilizado llega a la solución se considera un ejemplo positivo; en caso contrario un ejemplo negativo. El planificador utilizado es SHOP con una pequeña modificación. Si hay más de un método aplicable a la situación actual SHOP siempre escoge el primero. En CAMEL se cambia este comportamiento para que elija uno cualquiera aleatoriamente.

3.7.2. Aprendizaje basado en árboles de decisión

Los árboles de decisión son un método de aprendizaje inductivo que intenta obtener las características relevantes de un conjunto de ejemplos conocidos para hacer una distribución en clases, produciendo reglas de tipo *if-then* capaces de clasificar una nueva instancia desconocida en una clase. Son muy robustos al ruido, capaces de aprender bien con datos incompletos. Hay muchas variantes de algoritmos de árboles de decisión, como ID3 [J.R., 1983] y su continuación C4.5 [Quinlan, 1993], ITI [Utgoff *et al.*, 1997] ....

En planificación se ha utilizado este método de aprendizaje para determinar las acciones que debe seguir un robot móvil, Pioneer, en un dominio parcialmente observable ya que su sistema de sensores no es completo ni preciso [Schmill *et al.*, 2000]. Pretende adquirir de forma dinámica el modelo de operadores aplicables al robot, basándose en su propia experiencia. Los datos recogidos por sus sensores, primero se distribuyen en clases según las características de los patrones recibidos. Luego se utiliza un algoritmo de árboles de decisión para determinar las condiciones iniciales de cada una de las clases. Estas clases se incluyen en el planificador para que el agente pueda determinar de qué forma van a cambiar las lecturas de los sensores y de qué forma reaccionará el modelo con patrones desconocidos. El dominio parcialmente observable de Pioneer hace que el modelo que pueda aprender no sea completo. En general, no puede aprender todas las precondiciones necesarias y suficientes para cada uno de sus operadores.

3.7.3. Programación Lógica Inductiva (ILP)

El sistema GRASSHOPPER aprende reglas para seleccionar metas, operadores y argumentos de operadores utilizando FOIL [Quinlan and Cameron-Jones, 1995] como sistema inductivo [Leckie and Zukerman, 1991]. Este sistema busca decisiones similares en trazas de ejemplos resueltos por el planificador para construir de manera inductiva las reglas de control. El sistema EXEL también utiliza ILP aplicado a planificadores basados en el espacio de estados [Reddy and Tadepalli, 1997]. Considera que el aprendizaje de conocimiento de planificación es equivalente al aprendizaje de cláusulas de Horn, sin recursividad ni funciones. Utiliza un algoritmo recursivo e interactivo similar a CLINT [Raedt and Bruynooghe, 1992]. Se empieza por la clausula vacía H, por medio de preguntas de equivalencia se encuentra un ejemplo e no incluido en H y se modifica H para incluirlo. Mientras H no sea equivalente al concepto objetivo C, se repite este proceso con otros ejemplos. Los literales irrelevantes se descartan de la hipótesis por preguntas de relación.

3.7.4. Otros métodos inductivos

- Aprendizaje de *políticas generales*: una política general es una función que opera sobre todas las instancias de un determinado dominio y es capaz de determinar la acción que se debe aplicar basándose tanto en el estado del sistema como en las metas actuales. Ejemplos de políticas generales en el mundo de los bloques son: 'levantar un bloque mal puesto si está libre', 'poner el bloque actual en el destino si el destino está bien situado y libre', etc. A partir de ejemplos resueltos en un mismo dominio se pueden aprender estas políticas generales en planificadores basados en el espacio de estados. Este problema fue tratado inicialmente por Roni Khardon [Khardon, 1999]. Khardon representa una política general usando listas ordenadas de reglas y aprende las políticas generales de forma supervisada a partir de ejemplos resueltos, utilizando una variación del algoritmo de Rivest [Rivest, 1987] de listas de decisión. Similar a este trabajo es el de Martín y Geffner [Martin and Geffner, 2000], pero en vez de utilizar un lenguaje de reglas, representa las políticas generales en un lenguaje de conceptos [Brachman and Levesque, 1984, Geffner, 1999]. El lenguaje de conceptos tiene la misma expresividad que la lógica de predicados de primer orden, pero además tiene una sintaxis que permite razonar con clases de objetos. Por ejemplo, la clase de objetos 'bien situado' en el mundo de los bloques, se puede definir de manera compacta en términos de los predicados originales.

  El aprendizaje por refuerzo también está relacionado con el aprendizaje de políticas pero tienen dos diferencias importantes: la primera es que estas aproximaciones son algoritmos de aprendizaje supervisado, frente a RL que no lo es y la segunda diferencia es que, en RL, las políticas toman la forma de funciones de valor que les hace difícil adaptarse a los cambios en el tamaño del espacio de estados, lo cual impide que aprendan políticas generales. La programación genética se ha utilizado para buscar políticas generales. Koza en [Koza, 1994] y Baum en [Baum, 1995] desarrollaron algoritmos evolutivos para encontrar políticas generales en el mundo de los bloques. Koza las representa como programas Lisp mientras que en la aproximación de Baum se representan como una colección de reglas. Igual que en RL son métodos no supervisados pero sí que pueden aprender políticas generales. Otros sistemas basados en programación genética no aprenden conocimiento de control, sino que intentan aprender un programa planificador ya adaptado al dominio [Spector, 1994]. Es decir, buscan en el espacio de planificadores que funcionan bien en un dominio determinado. Hasta el momento, esta alternativa sólo ha sido ensayada en dominios muy simples.

- Aprendizaje de reglas en planificación por reescritura (*Plan Rewriting PbR*): La idea básica de PbR [Ambite and Knoblock, 2001] es, primero generar un plan inicial y después, reescribirlo iterativamente, usando una serie de reglas, hasta que la calidad del plan sea aceptable o se alcance el límite de un recurso

determinado. Este tipo de planificación tiene muchas ventajas con respecto a la escalabilidad, la calidad de los planes y el tiempo de resolución de problemas, pero requiere de tres entradas adicionales que no precisan otros planificadores independientes del dominio: un generador de planes inicial, un conjunto de reglas y una estrategia de búsqueda. Aunque las reglas están descritas en un lenguaje declarativo de alto nivel, su diseño y selección requieren un gran esfuerzo por parte del diseñador. Por eso, se ha creado un método capaz de aprender estas reglas a partir de ejemplos [Ambite *et al.*, 2000]. En un dominio determinado se resuelven un conjunto de problemas de entrenamiento, con un planificador óptimo y con el generador de planes. El sistema compara ambas soluciones y genera las reglas que transformarían la solución inicial en la óptima.

- Redes Neuronales [J.W. *et al.*, 1991]: El sistema UCPOP-NN utiliza redes neuronales para aprender en un planificador de orden parcial [Zimmerman, 1996]. En [Fernández *et al.*, 2002] se describe un método para aprender las acciones a realizar para resolver problemas del mundo de los bloques.

3.8. Aprendizaje mixto inductivo-deductivo

Sin embargo, los sistemas inductivos también tienen sus problemas. Por ejemplo, necesitan un gran número de ejemplos para aprender conocimiento de control eficiente y sus operadores de generalización a menudo requieren de un gran esfuerzo computacional. Para solventar los problemas de los métodos inductivos y deductivos, algunos investigadores los han combinado en un sistema multi-estrategia, los cuales pretenden superar las desventajas de los sistemas deductivos con las características de los inductivos, y viceversa. A continuación, se explican los sistemas e investigaciones más relevantes que entran en esta categoría.

3.8.1. Aprendizaje basado en casos (CBR) y por Analogía

Surgieron otros métodos que utilizaban analogía o CBR [Hammond, 1989, Hanks and Weld, 1995, Kambhampati, 1989, Kettler *et al.*, 1994, Veloso, 1994b], aplicados tanto a planificación basada en el espacio de estados [Avesani *et al.*, 2000, Bergmann and Wilke, 1996, Ihrig and Kambhampati, 1996b, Veloso and Carbonell, 1993], como a planificación de orden parcial
[Ihrig and Kambhampati, 1996a, Ihrig and Kambhampati, 1997]. La idea básica de esta línea de investigación es utilizar problemas previamente resueltos, sus soluciones (planes) e información asociada, para resolver nuevos problemas que son similares a ellos. Para tener una buena visión general del tema y de los principales sistemas de CBR aplicados a la planificación se puede consultar [Bergmann *et al.*, 1998].

Las cuestiones que CBR debe resolver en el caso de la planificación son las siguientes:

- Selección de un conjunto de casos (problemas) similares a los que se pretende resolver
- Adaptación de los casos al nuevo problema
- Validación de la solución obtenida
- Almacenamiento y organización de casos nuevos

La segunda cuestión, la adaptación del caso al nuevo problema, adquiere una dimensión importante en planificación. En CBR se distinguen dos tipos de adaptación: transformacional y generativa. La transformacional utiliza heurísticas dependientes del dominio para transformar los planes de problemas similares al que se quiere resolver [Carbonell, 1983]. La adaptación generativa utiliza el/los casos almacenados para dirigir a un planificador. Es decir, se intenta resolver cada nuevo problema con el planificador, pero utilizando un caso similar a dicho problema para orientar al sistema de planificación.

Un sistema representativo de CBR aplicado a planificación es PRODIGY4.0-ANALOGY que utiliza analogía derivacional [Veloso, 1994a]. PRODIGY4.0/ANALOGY almacena las decisiones tomadas por el planificador al resolver problemas junto con sus justificaciones. Esto permite identificar qué transformaciones es necesario realizar a las decisiones tomadas por el sistema de planificación para resolver aquellos problemas que sean similares pero no idénticos. A diferencia de ANALOGY, CAPLAN/CBC utiliza un planificador que busca en el espacio de planes [Muñoz-Avila and Huellen, 1995].

Otro sistema, PARIS, almacena cada caso en diferentes niveles de abstracción [Bergmann and Wilke, 1996]. Para resolver nuevos problemas, se equipara el problema con algún caso de la jerarquía de abstracciones y se utiliza el planificador para refinar el caso abstracto y adaptarlo al nuevo problema (relativo a la planificación utilizando abstracción existen también sistemas, como ALPINE, que pueden aprender automáticamente las abstracciones de un dominio de planificación [Knoblock, 1991]). ABALONE utiliza planificación y analogía en el campo de la demostración de teoremas [Reiser and Kaindl, 1994]. PRIAR sigue la filosofía de la analogía derivacional, utilizando las dependencias entre los diferentes operadores de un plan, para reutilizar los casos [Kambhampati and Hendler, 1989]. MRL utiliza representaciones lógicas, y plantea la recuperación de los casos de la base como preguntas lógicas (*logic query language*) [Koehler, 1994]. PLAGIATOR usa CBR para la demostración de teoremas, pero sigue una filosofía puramente transformacional [Kolbe and Brauburger, 1997].

El sistema HICAP (*Hierachical Interactive Case-based Architecture for Planning*) [Muñoz-Avila *et al.*, 1999] combina el planificador SHOP con un sistema

CBR llamado NACODAE [Breslow and Aha, 1997]. Cuando no se tiene una definición completa del dominio y no se tiene toda la información sobre el estado actual resulta difícil decidir qué método de refinamiento jerárquico utilizar. Los métodos de refinamiento o de expansión son las diferentes maneras en que una acción de un nivel jerárquico se transforma en acciones del siguiente nivel. NACODAE interactúa con el usuario por medio de preguntas y respuestas que le permiten extraer algún caso similar que tuviese almacenado que resuelva el problema. El sistema CASEADVISOR [Carrick *et al.*, 1999] también integra razonamiento basado en casos y planificación jerárquica. Utiliza planes almacenados con anterioridad para obtener información de cómo resolver una tarea en vez de para elegir el método de refinamiento. El aprendizaje CBR también se ha aplicado en planificación temporal, en dominios complejos de control de emergencias en los que el tiempo de respuesta es crítico [Avesani *et al.*, 1998].

3.8.2. Aprendizaje por refuerzo (RL)

El aprendizaje por refuerzo consiste en aprender políticas que permitan decidir, ante una situación o estado, qué acción es la más adecuada para lograr un objetivo o meta por medio de técnicas de prueba y error. En cada experiencia, se prueba a ejecutar una acción recibiendo del dominio un refuerzo positivo o negativo según el resultado de dicha acción. En la mayoría de los sistemas se pretende maximizar la suma de los refuerzos, como en Q-LEARNING [Watkins and Dayan, 1992] y en ADAPTIVE REAL-TIME DYNAMIC PROGRAMMING (ARTDP) [Barto *et al.*, 1995], pero existen otros sistemas donde se optimiza la media de los refuerzos [Mahadevan, 1996].

La teoría matemática que se aplica en su desarrollo se basa en los trabajos de programación dinámica de Bellman y la teoría de control óptimo que se comenzó a desarrollar a mediados de los 50
[Bellman, 1957, Bellman and Kalaba, 1965, Kumar, 1986]. Casi todos los algoritmos de RL están basados en la aproximación de las funciones de valor que estiman lo bueno que es para un agente estar en un determinado estado, o lo bueno que es ejecutar una determinada acción desde un determinado estado, que es equivalente a aprender una política de acción. En planificación con incertidumbre resultan muy apropiadas estas técnicas de aprendizaje, tanto si se conoce el modelo del dominio como si no. Cuando se conoce el modelo, se aplica programación dinámica para decidir qué acción aplicar en cada estado (la política) [Mahavedan and Connell, 1992, Sutton, 1990, Watkins and Dayan, 1992]. Cuando no se conoce el modelo, se tienen dos posibilidades: métodos basados en el modelo, que primero aprenden el modelo y luego ya aplican programación dinámica para aprender la política; y los métodos libres de modelos o aprendizaje por refuerzo directo, que directamente aprenden las políticas sin aprender el modelo.

El sistema LOPE [García-Martínez and Borrajo, 1998] aprende el modelo del

dominio observando los efectos que produce cada acción. El aprendizaje se lleva a cabo integrando tres técnicas: por observación, creando operadores directamente de la experiencia y observando sus consecuencias en el dominio; generalizando por medio de heurísticas operadores incorrectos; y por medio de aprendizaje por refuerzo que genera refuerzos positivos y negativos según la exactitud con que se adecuan al dominio. Cuando el espacio de estados es continuo, es decir, no existe un conjunto de estados discretos, se requieren técnicas de aprendizaje por refuerzo diferentes, como por ejemplo, las utilizadas en [Fernández and Borrajo, 2002]. El sistema H-LEARNING utiliza métodos de aprendizaje por refuerzo basándose en el modelo, optimizando la media de los refuerzos. Se ha aplicado al dominio de los vehículos automáticos guiados [Tadepalli and Ok, 1998], similar al sistema DYNA [Sutton, 1991]. Por último, se han hecho algunos trabajos para extender RL a POMDPs [Littman *et al.*, 1995, Parr and Russel, 1995].

3.8.3. Otros sistemas inductivos-deductivos

Algunos sistemas iniciales basados en este principio son LEX-2 [Mitchell *et al.*, 1982] y META-LEX [Keller, 1987]. AXA-EBL combina dos técnicas: EBL + inducción para aprender heurísticas aproximadamente correctas (es decir, cercanas a las que aprendería EBL sólo) [Cohen, 1990]. En primer lugar, AXA-EBL aprende reglas de control utilizando EBG. A partir de esas reglas, se construye un conjunto de reglas de control eliminando k o menos condiciones de la regla. A continuación se evalúan dichas reglas para encontrar las reglas de mínimo tamaño que cubran el mayor número de ejemplos positivos y el menor número de ejemplos negativos (inducción). AXA-EBL tiene el inconveniente de que sólo puede utilizar valores pequeños de k, puesto que el conjunto de aproximaciones a la regla original crece exponencialmente al incrementar k.

DOLPHIN (*Dynamic Optimization of Logic Programs through Heuristic INduction*) es una extensión de AXA-EBL que utiliza FOIL como sistema inductivo que es más potente [Zelle and Mooney, 1993]. HAMLET también combina inducción y deducción de una manera incremental [Borrajo and Veloso, 1997]. Utiliza dos módulos. El primero está basado en EBL y aprende reglas de control que son aproximadamente correctas y normalmente demasiado específicas. El segundo módulo usa inducción para generalizar y especializar las reglas. El sistema EVOCK combina el sistema de aprendizaje deductivo-inductivo de conocimiento de control basado en el espacio de versiones de HAMLET, con el paradigma de programación genética [Aler *et al.*, 2002]. Con ello se pretende superar las limitaciones de HAMLET, como son: la dependencia del lenguaje que usa para representar las hipótesis, dependencia de sus operadores de búsqueda de los ejemplos de aprendizaje y la dificultad de añadir sesgos que enfoquen el sistema no sólo a aprender conocimiento correcto sino también conocimiento eficaz.

Otro sistema multi-estrategia es SCOPE [Estlin and Mooney, 1997]. SCOPE com-

bina EBL e inducción de una manera diferente. El módulo principal está basado en ILP (en concreto usa el sistema FOIL) y como todo método inductivo, aprende a partir de ejemplos de problemas de planificación (decisiones correctas e incorrectas del planificador UCPOP). Usa EBL para obtener literales que son usados por ILP para construir nuevas cláusulas. De esta manera se reduce el espacio en el que ILP busca cuando construye nuevas cláusulas. Tanto HAMLET como SCOPE combinan las características de los métodos inductivos y deductivos para reducir sus problemas: EBL ya no necesita utilizar teorías completas del dominio con el objetivo de producir reglas completamente correctas, puesto que las reglas pueden ser corregidas por el sistema inductivo. Además, el que EBL produzca reglas demasiado específicas ya no es problemático, puesto que la inducción se encargará de generalizarlas. Desde el punto de vista de la inducción, ya no es necesario un número ingente de ejemplos para aprender, puesto que el aprendizaje está sesgado por el conocimiento aproximadamente correcto aprendido por el módulo deductivo. El sistema BLACKBOX/FOIL [Huang *et al.*, 2000] también utiliza EBL+ILP en planificación basada en el espacio de estados.

Existen sistemas híbridos, como DERNSLP+EBL, el cual combina EBL para planificadores de orden parcial y CBR. Este sistema aprende de los fallos que se producen tras recuperar un caso e intentar re-generarlo para adaptarlo a un nuevo problema [Ihrig and Kambhampati, 1996a]. TALUS intenta combinar el aprendizaje de macro-operadores, CBR y EBL dentro de un mismo esquema genérico [Allen *et al.*, 1992]. ALPINE combina aprendizaje de jerarquías de abstracción y EBL [Knoblock *et al.*, 1991]. X-LEARN, en lugar de aprender reglas de control, aprende un nuevo planificador basado en descomposición. X-LEARN utiliza ILP en combinación con técnicas EBL [Reddy and Tadepalli, 1997]. Para cada problema de aprendizaje, construye una regla de descomposición que resuelve ese problema. A continuación, se combina esa regla con las reglas aprendidas previamente de manera que se encuentre la generalización menos general. El sistema EBRL [Dietterich and Flann, 1997] utiliza EBL combinado con RL para poder tratar con teorías del dominio incompletas.

3.9. Conclusiones

Se ha presentado una serie de aproximaciones relacionadas con la planificación en IA y sistemas que aplican alguna de las múltiples formas de aprendizaje automática a los planificadores base independientes del dominio.

Referente a los sistemas de planificación la conclusión que se puede extraer es que no hay un planificador óptimo que destaque sobre todos los demás para todos los dominios y todos los problemas; cada técnica de planificación tiene sus ventajas e inconvenientes. Sin embargo, parece que los planificadores heurísticos y los basados en GRAPHPLAN son los que hoy en día tienen una mayor aceptación entre

la comunidad de planificación independiente del dominio por su rapidez. Por otro lado, los planificadores jerárquicos son los más utilizados en aplicaciones del mundo real. Muchos algoritmos de planificación, aunque las técnicas sean diferentes, comparten decisiones comunes como el operador a utilizar para alcanzar una meta o el orden de resolución de las metas pendientes.

En cuanto al aprendizaje, la conclusión derivada de estos trabajos es que el aprendizaje mejora los algoritmos de planificación y casi cualquiera de las técnicas clásicas de aprendizaje en IA es aplicable. La mayoría de estas investigaciones se dirigen a mejorar los recursos consumidos, como la memoria utilizada y el tiempo en la búsqueda de las soluciones, y muy pocas tratan de mejorar la calidad de los planes obtenidos.

En general, no hay un lenguaje estándar para representar el conocimiento de control, ni una metodología general de aprendizaje adaptable a diferentes paradigmas de planificación. Tampoco hay trabajos centrados en transferir conocimiento de control entre diferentes planificadores.

Capítulo 4

Tecnología utilizada

En esta sección se explican en detalle los planificadores y sistemas de aprendizaje desarrollados por otros investigadores que adquieren especial relevancia en la elaboración de la tesis.

La secciones 4.1 y 4.2 describen PRODIGY4.0 y HAMLET respectivamente, ya que los nuevos sistemas de aprendizaje propuestos en esta tesis están basado en HAMLET, sistema multi-estrategia deductivo-inductivo de aprendizaje de reglas de control para PRODIGY4.0. Además, en varios de los experimentos realizados para validar las ideas de la tesis intervienen ambos sistemas. La sección 4.3 explica el sistema de aprendizaje EVOCK que complementa a HAMLET y se utiliza para estudiar la transferencia de conocimiento de control entre sistemas de aprendizaje en planificación.

Los sistema de aprendizaje en planificación dependen estrechamente del planificador al que se aplica. Los sistemas aquí implementados utilizan los planificadores: HYBIS que se explica en la sección 4.4 y TGP, explicado en 4.6. TGP es una ampliación de GRAPHPLAN para tratar razonamiento temporal. La sección 4.5 explica en detalle el planificador GRAPHPLAN y los trabajos posteriores realizados para mejorarlo.

4.1. Planificador PRODIGY

PRODIGY4.0 es un planificador no lineal de orden total bidireccional en profundidad que hace búsqueda en el espacio de estados para encontrar el plan solución (ver [Veloso *et al.*, 1995] para los detalles del algoritmo). Empezando por las metas del problema va asignando operadores que las satisfagan, y según va encontrando operadores que se puedan ejecutar los ejecuta desde el estado inicial. Las entradas a PRODIGY4.0 son las estándar en cualquier planificador: la definición del dominio y el problema, más un conjunto de reglas de control o heurísticas que

especifican cómo tomar las decisiones durante el proceso de búsqueda. También se le puede controlar por un conjunto de parámetros como el límite de tiempo y nodos hasta encontrar la solución, el límite de profundidad del algoritmo de búsqueda, si debe encontrar múltiples soluciones .... La salida es un plan válido de orden total; una secuencia de operadores instanciados tales que, cuando se ejecutan, transforman el estado inicial del problema en un estado donde todas las metas del problema son ciertas (se han conseguido).

Todo el conocimiento se especifica usando el lenguaje PDL (PRODIGY DESCRIPTION LANGUAGE [Carbonell *et al.*, 1992, Minton *et al.*, 1989]) que es similar en términos de poder de representación al de ADL en la versión PDDL2.2 (incluyendo axiomas, llamados reglas de inferencia en PDL).[1].

Desde la perspectiva de adquisición de heurísticas, el ciclo de razonamiento del algoritmo de planificación implica distintos tipos de puntos de decisión:

1. *Seleccionar una meta* del conjunto de metas pendientes; es decir, precondiciones de operadores previamente seleccionados que no sean ciertas en el estado actual de planificación (por omisión, selecciona la meta sobre la que trabajar según el orden de la definición de las metas en el fichero del problema).

2. *Elegir un operador*, entre los definidos en el dominio, para conseguir la meta particular seleccionada (por omisión, se exploran según su posición en el fichero del dominio).

3. *Elegir los valores de las variables* para instanciar el operador elegido (por omisión, selecciona los valores que lleven a un operador instanciado con más precondiciones ciertas en el estado actual).

4. *Aplicar* el operador instanciado (decisión por omisión en la versión utilizada en esta tesis), o seleccionar otra meta no resuelta empezando un nuevo ciclo de razonamiento.

Cuando un operador instanciado es *aplicado* pasa a ser la siguiente acción del plan solución y el estado de planificación se actualiza según los efectos de dicho operador (bidireccionalidad)

En la figura 4.1 se muestra un ejemplo del árbol de búsqueda generado por PRODIGY al resolver un problema del dominio Blocks donde se observan los puntos de decisión anteriores. En este dominio se definen cuatro operadores: `pick-up`, cuando el brazo del robot coge un bloque de encima de la mesa; `put-down`, cuando se deja el bloque sostenido por el brazo sobre la mesa; `stack`, cuando el brazo del robot coge un bloque que está encima de otro bloque; y `unstack`, cuando se

[1] Se han definido traductores en las dos direcciones, PDL a PDDL y viceversa, cubriendo casi todas las características de representación.

deja el bloque sostenido por el brazo encima de otro bloque (para una descripción completa del dominio consultar el apéndice A.5). Inicialmente se tienen los bloques C,B y D sobre la mesa, y el bloque A encima del C. La meta del problema es que el bloque A esté sobre la mesa. El primer punto de decisión es seleccionar la meta sobre la que empezar a trabajar. En este caso sólo hay una, `on-table(A)`. Luego selecciona un operador, de entre todos los definidos en el dominio, que consiga dicha meta; es decir, que tenga entre sus efectos la meta. En este dominio sólo existe el operador `put-down`. El tercer punto de decisión es asignar las variables al operador elegido. En este caso, el operador sólo tiene una variable que queda fijada por la propia meta, el bloque A. Las precondiciones del operador, que no sean ciertas en el estado actual, pasan a la lista de metas pendientes y comienza un nuevo ciclo de razonamiento. En el ejemplo, la precondición de `put-down(A)` es `holding(A)`. El segundo ciclo de razonamiento empieza seleccionando la única meta pendiente `holding(A)`. Existen dos operadores del dominio que la añaden, `pick-up` y `unstack`. Selecciona el primer operador `pick-up`. El siguiente nodo asigna las variables al operador seleccionado `pick-up(A)`. Las precondiciones de dicho operador son `arm-empty`, `clear(A)` y `on-table(A)` que era la meta que se estaba tratando de solucionar en primer lugar, con lo que se ha entrado en lo que se denomina un bucle de meta y el planificador etiqueta el nodo como de fallo. Retrocede y selecciona el operador `unstack`. Tiene dos variables: el bloque a levantar y el bloque desde el que se levanta. La primera queda fijada por la meta, `holding(A)`, pero la segunda variable puede ser cualquier bloque del problema. El planificador intenta todas las opciones hasta llegar a la correcta, `unstack(A,C)`. Todas sus precondiciones son ciertas en el estado de planificación y se llega al último punto de decisión posible, aplicar dicho operador instanciado o seguir resolviendo otra meta. En este ejemplo no quedan metas pendientes y aplica el operador instanciado `UNSTACK(A,C)`, con lo que ya se cumplen todas las precondiciones del operador `put-down(A)` y aplica el operador instanciado `PUT-DOWN(A)`. Si hubiese otras metas pendientes, en este punto el planificador podría seleccionar otra meta para resolver en vez de aplicar el operador. El plan solución se compone de los dos operadores instanciados: `UNSTACK(A,C)` y `PUT-DOWN(A)`.

PRODIGY permite definir conocimiento de control declarativo. En cada uno de los puntos de decisión se pueden disparar reglas que determinen el siguiente nodo a explorar. En el ejemplo anterior se pueden definir una regla para seleccionar el operador `unstack` y otra para seleccionar las variables de dicho operador tal como se muestra en la figuras 4.2 y 4.3 respectivamente.

En la siguiente sección se explica en detalle la sintaxis de estas reglas ya que va a ser el modelo de partida para definir el lenguaje común de representación del conocimiento de control.

El planificador no incorpora heurísticas independientes del dominio ya que el objetivo de PRODIGY no es construir el planificador más eficiente, sino estudiar

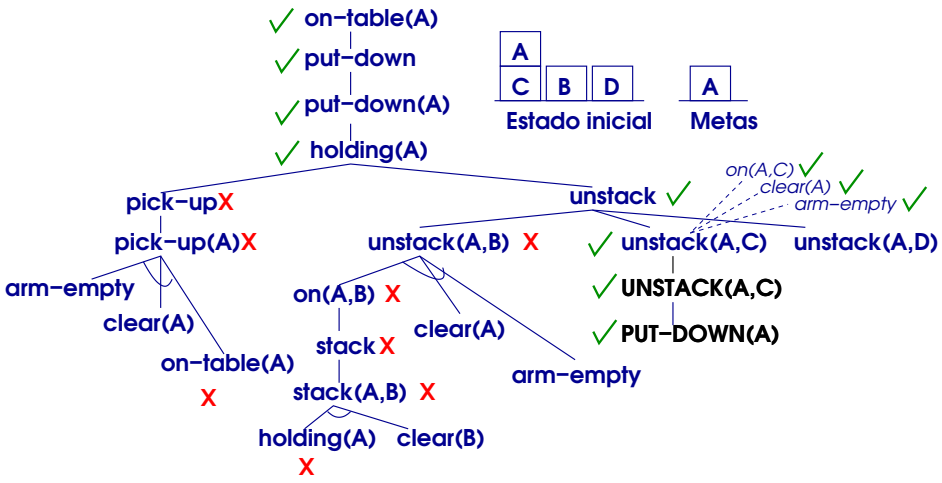

Figura 4.1: Ejemplo de árbol de búsqueda genérico en PRODIGY.

```
(control-rule select-op-unstack-for-holding
   (if (and (current-goal (holding <x>))
            (true-in-state (on <x> <y>))))
   (then select operator unstack))
```

Figura 4.2: Ejemplo de regla de control de PRODIGY para seleccionar el operador `unstack` en el dominio de los bloques.

técnicas de aprendizaje automático para la resolución de problemas. La idea era que la arquitectura pudiera incrementalmente aprender conocimiento (como heurísticas, casos, modelos del dominio . . .) a través de la interacción entre el planificador y la técnica de aprendizaje, de forma que este conocimiento haga a PRODIGY eficiente.

4.2. Sistema de aprendizaje HAMLET

HAMLET es un sistema de aprendizaje mixto, combinando técnicas de aprendizaje deductivo e inductivo, que aprende automáticamente reglas de control para el planificador PRODIGY. A continuación se explica un resumen de sus características principales. Para una completa descripción del sistema consultar [Borrajo and Veloso, 1997].

La figura 4.4 muestra un esquema de su arquitectura. Recibe como entrada un conjunto de problemas de aprendizaje de un dominio. Por cada problema, PRODIGY lo resuelve y guarda el árbol de búsqueda generado hasta encontrar la solución. El módulo deductivo de HAMLET explora el árbol aplicando EBL para dar una

```
(control-rule select-bindings-unstack-holding
   (if (and (current-goal (holding <x>))
            (current-operator unstack)
            (true-in-state (on <x> <y>))))
   (then select bindings
            ((<ob> . <x>) (<underob> . <y>))))
```

Figura 4.3: Ejemplo de regla de control para seleccionar las variables del operador `unstack` en el dominio de los bloques.

explicación de la decisiones correctas de PRODIGY que llevaron a la solución y genera un conjunto de reglas de control que son aproximadamente correctas. Dichas reglas pueden ser demasiado generales o demasiado específicas. El módulo inductivo usa aprendizaje inductivo para corregir dichas reglas, generalizándolas cuando encuentra nuevos ejemplos positivos de entre el resto de problemas de aprendizaje o especializándolas si el ejemplo es negativo. Además del dominio y los problemas HAMLET tiene como entradas diversos parámetros de aprendizaje del módulo deductivo. Los más importantes son:

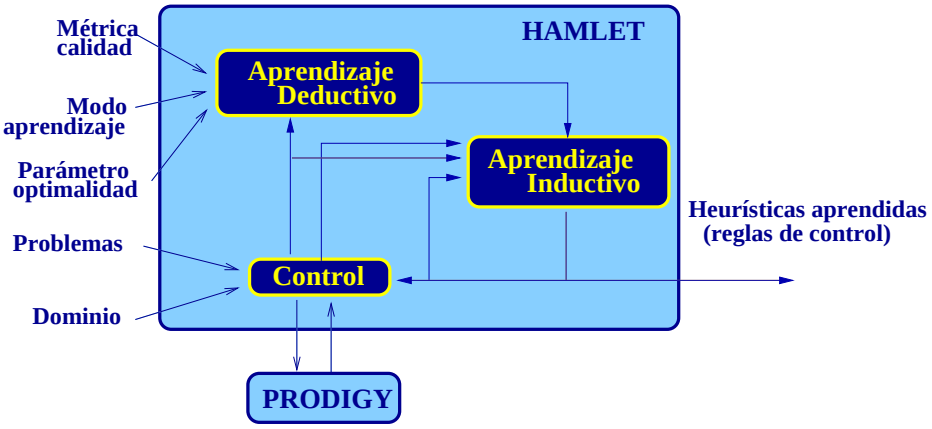

Figura 4.4: Esquema arquitectura de HAMLET.

- Métrica de calidad: determina el criterio o métrica a utilizar para medir la calidad de las soluciones encontradas por PRODIGY. El conocimiento de control generado tratará de maximizar la calidad. Ejemplos de métricas pueden ser la longitud del plan, el tiempo tardado en encontrar la solución, el coste económico del plan . . . o incluso combinaciones de más de un criterio.

- Modo de aprendizaje: HAMLET permite que el aprendizaje se haga sobre todos los puntos de decisión del camino solución durante la resolución del

problema o bien, sólo sobre las decisiones que no sean la opción por omisión del planificador.

- Parámetro de optimalidad: para que el aprendizaje tenga lugar es necesario que PRODIGY resuelva un problema y genere todo el árbol de búsqueda. En algunos dominios esto no es posible y HAMLET permite empezar el aprendizaje sobre un árbol de búsqueda no generado completamente.

La figura 4.5 describe un ejemplo de regla de control aprendida por HAMLET en el dominio Logistics (se describe en el apéndice A.1) para seleccionar el operador `unload-airplane`. La regla dice que si la meta es tener un paquete en una localidad, <`location1`>, y el paquete está actualmente dentro de un avión, entonces el planificador debería usar el operador `unload-airplane` en vez del operador `unload-truck` que también alcanza la misma meta (tener un paquete en una localidad dada). Sin esta regla, debido al razonamiento hacia atrás, PRODIGY intentaría satisfacer la meta utilizando el primer operador en la definición del dominio que entre sus efectos la añada (`unload-truck`) añadiendo en la lista de metas pendientes las precondiciones de dicho operador, incluida la precondición de que el paquete esté dentro de un camión. Intentaría buscar operadores para conseguir que el paquete esté dentro de un camión incrementando innecesariamente el árbol de búsqueda. Eligiendo el operador seleccionado por la regla se consigue alcanzar la meta aplicando un único operador.

HAMLET induce esta regla más general a partir de dos reglas más específicas aprendidas, en el módulo deductivo, de dos problemas de entrenamiento distintos. En el primer problema, el avión está en un aeropuerto distinto de <`location1`> y la primera regla aprendida tiene como precondición un `true-in-state` más indicando la localización del avión. En un segundo problema, el avión está en <`location1`> añadiéndose el correspondiente meta-predicado `true-in-state`. El módulo inductivo de HAMLET detecta que ambas reglas están subsumidas por la regla más general aprendida (las dos reglas más específicas se quitan).

```
(control-rule select-operators-unload-airplane
   (if (and (current-goal (at <package> <location1>))
            (true-in-state (inside <package> <airplane>))
            (different-vars-p)
            (type-of-object <package> package)
            (type-of-object <airplane> airplane)
            (type-of-object <location1> airport)))
   (then select operator unload-airplane))
```

Figura 4.5: Ejemplo de regla de control aprendida por HAMLET para seleccionar el operador `unload-airplane` en el dominio Logistics.

4.3. Sistema de aprendizaje EVOCK

EVOCK es un sistema de aprendizaje inductivo para aprender reglas de control basado en Programación Genética (ver [Aler *et al.*, 2002] para los detalles). Empieza con una población inicial formada por un conjunto de reglas (llamadas individuos) que pueden ser generadas aleatoriamente o proporcionadas externamente.

La figura 4.6 muestra un ejemplo de individuo en el dominio Blocks. En este caso, el individuo consta de dos reglas: `select-op-unstack-1` y `select-op-unstack-2`. Cada regla se compone de dos partes: la parte de precondiciones, representada en la figura debajo de `IF`, y la parte del consecuente (debajo de `THEN`). La parte de precondiciones, a su vez, se divide en meta-predicados, separando las variables de las constantes . Y la parte del consecuente se divide en tres: el tipo de regla (`select`, `prefer` o `reject`), la acción (`operator, goal, bindings`) y el argumento. Los individuos se dividen según se muestra en la figura porque los operadores genéticos actúan sobre cada parte.

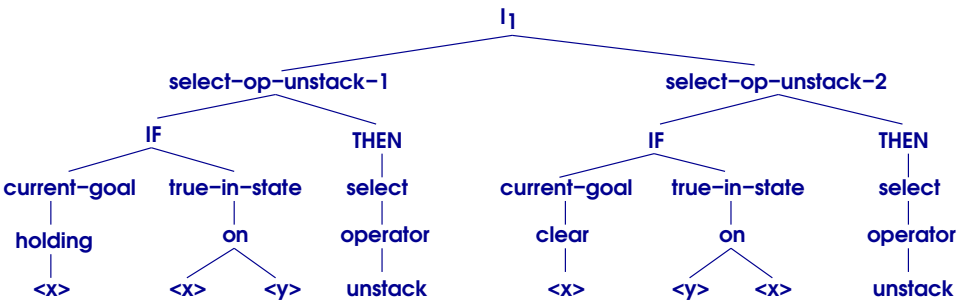

Figura 4.6: Ejemplo de individuo en EVOCK.

Después, sigue un tipo de búsqueda heurística en haz para encontrar un conjunto de reglas suficientemente buenas. Durante la búsqueda, los individuos se modifican mediante los operadores genéticos. Existe una gramática para generar solamente reglas sintácticamente correctas. Los operadores genéticos pueden añadir reglas (o partes de ellas), quitar reglas (o partes de ellas) y cruzar partes de unas reglas con partes de otras, igual que el operador estándar *crossover* de Programación Genética. EVOCK también incluye algunos operadores hechos a medida para modificar las reglas de control. La búsqueda en EVOCK está guiada por la función de *fitness* que evalúa los individuos entre otros criterios según el número de problemas del conjunto de entrenamiento que es capaz de resolver, el número de nodos expandidos, y el tamaño del individuo (son preferibles los más pequeños).

4.4. El planificador HYBIS

El diseño correcto y completo de un programa de control industrial es muy complejo, incluso para un humano. Tradicionalmente, los ingenieros de control han usado diferentes metodologías, estándares y herramientas para llevar a cabo esta tarea. El estándar ISA-SP88 [ANS, 1995] constituye una de estas metodologías usada para hacer un diseño jerárquico de programas de control industrial. El planificador HYBIS mezcla técnicas de HTN y POCL (versión de POC) para aproximar la resolución de problemas de planificación a la manera en la que los ingenieros diseñan los programas de control en los sistemas de manufacturación en dos sentidos:

- representa una planta industrial como una composición jerárquica de dispositivos a distintos niveles de abstracción que aceptan el estándar SP88, lo cual facilita al ingeniero la representación de los dominios (entrada) y
- genera programas de control automáticamente a distintos niveles de detalle, cercano a la manera en que los humanos desarrollan un programa modular de control industrial, lo cual produce una salida más entendible.

Los dominios para el planificador son representados por una jerarquía de agentes donde la raíz (un agente ficticio) representa toda la planta industrial, los nodos hojas son los **agentes primitivos** que representan los dispositivos de la planta y los nodos intermedios son los **agentes agregados**. La estructura y comportamiento de los agentes agregados viene determinada por el conjunto de agentes de un nivel menor de abstracción del que estén compuestos. Cada agente agregado tiene conocimiento de las diferentes alternativas que hay para llevar a cabo las acciones del siguiente nivel. Esto es equivalente a los diferentes métodos usados en HTN para descomponer cada operador.

4.4.1. Ejemplo de dominio

Como ejemplo de descripción de un dominio se tiene el sistema de manufacturación ITOPS extraído de [Viswanathan *et al.*, 1998]. La figura 4.7 muestra el diagrama de la planta al más alto nivel. La descripción de la composición jerárquica de sus agentes se muestra en la figura 4.8.

Esta planta tiene los productos r1 a r5 inicialmente en los tanques S1 a S5. Los productos intermedios, obtenidos por la reacción de los iniciales, son I1 a I4, según el siguiente esquema:

- **Mix** r1, r2, y r3 producen I1.
- **Filter** I1 produce I3.

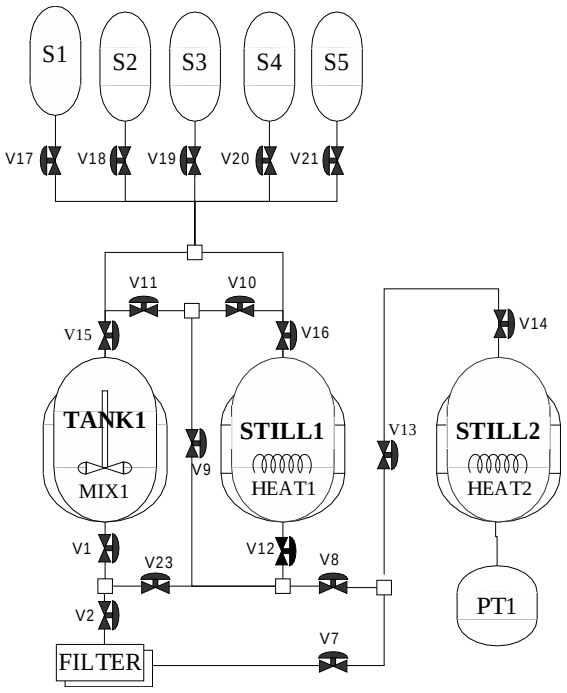

Figura 4.7: Planta ITOPS. Ejemplo dominio para HYBIS.

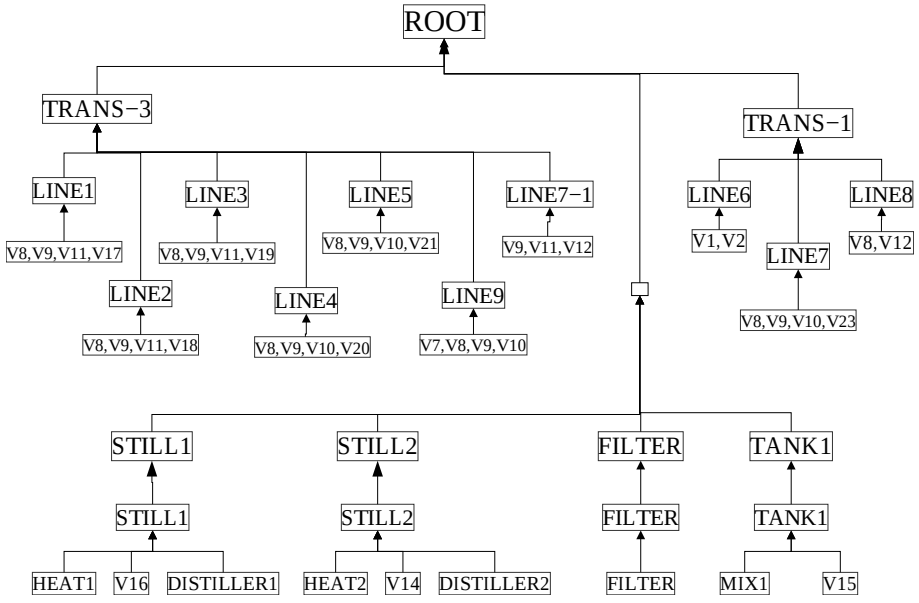

Figura 4.8: Planta ITOPS . Jerarquía de agentes.

- **Heat** r4, r5 e I3 producen I4.

El agente TANK1 es un tanque para contener los tres productos que vienen de los tanques S1 a S5, que son mezclados por el agente MIX1. Los agentes STILL1 y STILL2 compuestos por HEAT1 y HEAT2 respectivamente calientan unos productos y los transforman en otros. El agente FILTER sirve para filtrar. Los transportadores TRANS-3 y TRANS-1 son similares salvo que el primero puede transportar cualquier combinación de tres productos desde un punto hasta otro, siempre que estén conectados por cualquiera de los circuitos que lo componen. Y TRANS-1 se refiere a un sólo producto. Los agentes denominados LINE son circuitos compuestos por un número determinado de válvulas.

Este dominio tiene tres niveles de abstracción con los siguientes agentes (entre paréntesis el tipo):

- Nivel 1: TANK1-AGG(tank), STILL1-AGG(still), STILL2-AGG(still), FILTERAGG-AGG(filter-agg), TRANS-3(line-3prod), TRANS-1(line-1prod).
- Nivel 2: TANK1(tank), STILL1(still), STILL2(still), FILTERAGG(filter-agg), LINE1 a LINE9 y LINE7-1(valve-agg).
- Nivel 3: MIX1(mixer-3prod), HEAT1(heater-3prod), DISTILLER1(distiller), HEAT2(heater-3prod), DISTILLER2(distiller), FILTER(filter), V1, V2, V7 a V10(valve), V11,V12,V15 a V21, V23(valve-fwd).

4.4.2. Descripción de los agentes

Cada agente tiene un comportamiento similar al de un autómata finito: está en un estado que puede ser cambiado por una acción (operador) definida dentro del agente. Un agente se representa como g=<N,V,E,A> donde

- N(g) representa el nombre único del agente.
- V(g)=$v_1, ..., v_n$ es un conjunto de variables que pueden tomar su valor (son instanciadas) de objetos definidos en el dominio. Cada variable tiene restricciones de asignación y de no-asignación.
- E(g)=$s_1, ..., s_m$ es el conjunto de estados en los que un agente puede encontrarse.
- A(g)=$a_1, .., a_p$ es el conjunto de acciones que pueden ser ejecutadas por el agente. Cada acción produce un único cambio de estado en el agente.

Por ejemplo, en la planta ITOPS de la figura 4.7 se tiene el agente MIX1 que mezcla tres productos contenidos en el tanque TANK1, y viene definido de la siguiente forma:

```
N= MIX1
V= (?P1 ?P2 ?P3 ?RESULT ?LOC)
E= (OFF MIXING)
A= (MIX OFF-MIX)
```

Las variables de este agente son los tres productos que puede mezclar (?P1,?P2 y ?P3), el resultado obtenido de la mezcla (?RESULT) y el lugar donde están contenidos los productos (?LOC). Los estados son dos: OFF, cuando está parado y MIXING cuando está activo. Y las acciones también son dos: MIX (mezclar productos) y OFF-MIX (parar de mezclar los productos).

Una acción primitiva es una acción que puede llevar a cabo un agente primitivo y se representa mediante la tupla a_p=<N,Arg,Req,Efs> donde:

- N(a_p) es un símbolo identificativo de la acción dentro del agente al que pertenece.
- Arg(a_p) es un conjunto de variables o constantes que representan los argumentos de la acción.
- Req(a_p) es el conjunto de requisitos, representado cada uno como un literal, que deben ser ciertos para poder ejecutar la acción. Existen cuatro tipos distintos de requisitos: previos, deben cumplirse antes de que se ejecute a_p; simultáneos, deben cumplirse durante la ejecución de a_p; de consulta, se deben cumplir antes de ejecutarse a_p pero se interpretan como la respuesta a una consulta sobre la situación producida por la ejecución de otras acciones; y posteriores, condiciones que se deben cumplir después de ejecutar a_p.
- Efs(a_p) es el conjunto de efectos de la acción. Es un conjunto de literales que representan el cambio de estado producido en el agente por la ejecución de a_p, además del cambio producido en los objetos del dominio debido a la ejecución de la acción.

En el ejemplo anterior del agente MIX1, la acción MIX viene definida por:

```
N=MIX
Arg=(MIX1 ?RESULT)
Req= Previos:     ((TRANS-OP ?P1 ?P2 ?P3 ?RESULT)
                   (STATE MIX1 OFF)
                   (CONTAINS ?P1 ?LOC)
                   (CONTAINS ?P2 ?LOC)
                   (CONTAINS ?P3 ?LOC))
     Durante:     ((TRANS-OP ?P1 ?P2 ?P3 ?RESULT))
     Consulta:    NIL
     Posteriores: NIL
Efs=((STATE MIX1 MIXING)
```

```
(MIX ?RESULT ?LOC)
(CONTAINS ?RESULT ?LOC)
(NOT (STATE MIX1 OFF)))
```

La acción MIX que mezcla los tres productos requiere dos argumentos, el propio agente MIX1 y el producto resultado de la mezcla ?RESULT. Los requisitos necesarios previos a la ejecución de la acción son que el agente esté parado, (STATE MIX1 OFF), que la mezcla de los tres productos se transforme en el producto resultado, (TRANS-OP ?P1 ?P2 ?P3 ?RESULT) y que cada producto esté ubicado en la misma localización, (CONTAINS ?PX ?LOC). El único requisito necesario durante la ejecución de la acción es que los tres productos se puedan transformar en el producto resultado. No requiere ningún requisito ni de consulta ni posteriores. El efecto de esta acción es que cambia el estado del agente a MIXING y por tanto deja de estar en el estado OFF, y que el producto resultado ?RESULT queda mezclado en ?LOC.

Los requisitos y efectos de una acción se representarán como un conjunto de literales, donde un literal es la representación de un predicado que puede estar instanciado o no. Las acciones de los agentes agregados se representan igual que las de los primitivos pero con una propiedad más llamada **expansión** que especifica las diferentes maneras posibles de transformar una acción en un problema de otro nivel. Cada método de expansión se representa por un conjunto de literales, que pueden estar ordenados, representando un problema a resolver por agentes del siguiente nivel de abstracción. Por ejemplo, en la planta ITOPS de la figura 4.8 se tiene el agente TRANS-3 del nivel 1 que transporta tres productos desde un punto hasta otro. Para su acción agregada OFF se definen dos métodos de expansión adicionales:

```
((STATE LINE1 OFF) (STATE LINE4 OFF) (STATE LINE7-1 OFF)) y
((STATE LINE4 OFF) (STATE LINE5 OFF) (STATE LINE9 OFF))
```

Es decir, dicha acción se puede transformar en dos problemas del nivel 2 distintos. El primero consiste en que tres de los agentes del que está compuesto TRANS-3, las líneas de transporte 1, 4 y 7-1, se paren. El segundo problema es equivalente pero con las líneas 4, 5 y 9.

Las propiedades y comportamiento de cada agente agregado se relaciona con los agentes que lo componen por medio de la **interfaz**. La interfaz es una lista ordenada de reglas de herencia de restricciones que sirven como herramienta sintáctica para poder describir la asociación de abstracción entre el conocimiento que representa a un agente compuesto y el de sus componentes. Si un agente agregado no tiene definido un método de expansión, la función de expandir una acción devuelve una lista de literales resultado de aplicar la interfaz de su agente a los literales de sus efectos. El método por omisión es independiente del dominio pues sólo es una traducción de los efectos de una acción compuesta a literales (metas) del siguiente nivel que deberán resolverse por subacciones de la acción compuesta. Una misma acción compuesta podría descomponerse de formas distintas (en el sentido de las

subacciones del otro nivel) siguiendo el método por omisión, según el contexto del problema.

4.4.3. Algoritmo de planificación

El proceso de planificación es un algoritmo generativo y regresivo a distintos niveles de detalle. Cada plan en un nivel de abstracción es refinado en otro plan de menor nivel hasta que no quede ninguna acción primitiva en el plan de menor nivel de abstracción (y mayor nivel en la jerarquía). En cada nivel, el plan es generado por MACHINE [Castillo *et al.*, 2001], un Planificador de Orden Parcial. La entrada al planificador completo HYBIS es la descripción del dominio (jerarquía de agentes), el estado inicial que es el conjunto de literales representando las condiciones iniciales de los estados de todos los agentes del dominio y un objetivo al mayor nivel de abstracción a resolver. El procedimiento es el siguiente:

- Primero, por medio de un proceso generativo POP se obtiene la secuencia de acciones de control que deben ser resueltas por los agentes del mayor nivel de abstracción.
- Segundo, si la secuencia encontrada está compuesta sólo por acciones primitivas, entonces el problema está resuelto. Si no, la secuencia es refinada jerárquicamente; es decir, el algoritmo expande cada actividad agregada según la interfaz del agente o los métodos de expansión que tenga definido, para obtener un nuevo problema de un nivel menor.
- Tercero, el algoritmo se repite recursivamente para resolver el problema con los agentes del siguiente nivel.

Por tanto, el plan final obtenido por este algoritmo es la secuencia de control a diferentes niveles de modularidad. El algoritmo de planificación se muestra en la figura 4.9. Las funciones principales son:

PrimitivePlan?: devuelve verdadero si la secuencia obtenida está compuesta sólo por acciones primitivas.

DownwardRefine: este módulo expande cada actividad agregada según el interfaz del agente y los métodos de expansión definidos. Cada método es representado por un conjunto de literales, que pueden estar ordenados, representando un problema a resolver por los agentes del siguiente nivel de abstracción. En realidad es un proceso más complejo porque prueba distintas maneras de hacer la expansión eligiendo la mejor.

GENERATE extiende el algoritmo de **MACHINE** (que se describe a continuación) para que el plan en construcción preserve las restricciones heredadas de orden e intervalos y asegure la modularización correcta de las acciones.

```
HYBRID (Domain,Level,Agenda,H-Plan,H-Links)

 IF Agenda is EMPTY THEN
      IF PrimitivePlan?(H-Plan) THEN  return H-Plan
      ELSE  DownwardRefine
            Result=HYBRID (Domain,Level,Agenda,H-Plan,H-Links)
            IF Result FAIL THEN return FAIL
 ELSE
      Result=GENERATE (Domain,Level,Agenda,H-Plan,H-Links)
      IF Result=FAIL THEN return FAIL
      ELSE return HYBRID (Domain,Level,Agenda,H-Plan,H-Links)
```

Figura 4.9: Algoritmo de planificación de HYBIS.

```
MACHINE (Domain,Agenda,Plan,Links)

1. IF Agenda is EMPTY THEN return SUCCESS
2. Task <-- SelectTask (Agenda)
3. Choices <-- HowToDoIt? (Task, Domain, Plan)
4. Iterate over Choices until it is empty
    (a) How <-- ExtractFirst (Choices)
    (b) DoIt (How, Domain, Agenda, Plan, Links)
    (c) IF MACHINE (Domain, Agenda, Plan, Links) THEN
           return SUCCESS
5. Return FAIL
```

Figura 4.10: Algoritmo de planificación de MACHINE.

El proceso generativo que busca un plan con técnicas de POP en un mismo nivel se basa en el algoritmo de MACHINE de la figura 4.10. Empieza por un plan vacío con dos acciones, START y END que codifican el problema de planificación como en SNLP [McAllester and Rosenblitt, 1991] y UCPOP
[Penberthy and Weld, 1992]. Sobre este plan inicial, en cada paso se resuelve una tarea pendiente hasta que no quede ninguna por resolver o el problema no tenga solución. Las diferentes tareas a resolver pueden ser metas pendientes, amenazas, interferencias o inconsistencias en el orden. Todas ellas se incluyen en la Agenda. El proceso de búsqueda para resolver una tarea de la Agenda es primero en profundidad con respecto a las distintas maneras (*choices*) que hay para resolverla. Durante el proceso de planificación se utilizan cuatro estructuras de datos: la Agenda, el Plan, los Enlaces causales y el Dominio. El Plan es el conjunto de nodos parcialmente ordenados, donde cada nodo es una acción instanciada del dominio o una meta, junto con los enlaces causales existentes entre las acciones. La Agenda es el conjunto de todas las tareas pendientes. Los tres módulos básicos del algoritmo se describen a continuación:

- **SelectTask**: selecciona la primera tarea de la Agenda para resolverla. Las tareas en la agenda están ordenadas según el siguiente esquema: primero las inconsistencias de orden y luego las interferencias, amenazas y metas.

- **HowToDoIt?**: construye una lista con todas las posibles maneras para resolver la acción seleccionada de la agenda. Las inconsistencias de orden no tienen solución. Las interferencias y amenazas se resuelven por los métodos de *promotion* (adelantar) o *demotion* (retrasar). Las metas se pueden resolver de tres maneras: por los axiomas, añadiendo una acción nueva del dominio, o reutilizando una acción incluida en el plan parcial.

- **DoIt**: aplica una de las alternativas *choices* de la lista de *HowToDoIt*. La inclusión de una nueva acción para resolver una meta pendiente implica la inclusión de todas sus precondiciones como metas pendientes y la inclusión de un nuevo enlace causal. Si la acción es reusada se incluye el enlace causal.

4.5. Planificador GRAPHPLAN

En este apartado se expone brevemente el funcionamiento de GRAPHPLAN y los trabajos más notables que tratan de mejorar su eficacia con diversas técnicas de aprendizaje.

Graphplan alterna dos fases:

- Expansión del grafo: extiende el grafo del plan hasta lograr las condiciones necesarias (no suficientes) para que el plan exista.

- Extracción de la solución: realiza una búsqueda en encadenamiento hacia atrás dentro del grafo del plan buscando un plan válido. Si no existe, se realiza otro proceso de expansión y así sucesivamente.

El grafo de planificación tiene dos tipos de nodos colocados por niveles:

- proposiciones: efectos de las acciones del nivel anterior (niveles pares)
- acciones: cuyas precondiciones están presentes en el nivel anterior (niveles impares).

El nivel cero lo constituyen las proposiciones correspondientes al estado inicial del problema. El nivel uno contiene aquellas acciones cuyas precondiciones están presentes en el nivel cero (podrían ejecutarse en el estado inicial). El siguiente nivel de proposiciones se forma con los efectos de las acciones del nivel anterior y así sucesivamente. En todos los niveles de acciones, además, se consideran acciones ficticias `no-op` por cada una de las proposiciones existentes que tiene como única precondición y efecto dicha proposición. Cuando se añade la acción `no-op` se dice que la proposición persiste, ya que es una manera de que las proposiciones vayan pasando a niveles posteriores para representar el hecho de que no cambian por la ejecución de los operadores (relacionado con el problema del marco en planificación).

En el grafo del plan todas las acciones están representadas en paralelo. Sin embargo, no necesariamente todas se pueden realizar al mismo tiempo. Se define una relación de mutex entre nodos del mismo nivel de la siguiente forma:

- Entre acciones: dos acciones son mutex cuando una borra alguna precondición o efecto de la otra, o cuando alguna de las precondiciones de una es mutex con alguna de las precondiciones de la otra, en el nivel anterior.
- Entre proposiciones: dos proposiciones son mutex si todas las formas de alcanzarlas por las acciones del nivel anterior son mutex.

La figura 4.11 muestra un posible grafo del plan para un problema definido por el estado inicial, {l1, l2 y l3} y las metas {G1,G2,G3,G4}. También se representan gráficamente las relaciones de mutex que se pueden dar entre acciones y entre proposiciones.

GRAPHPLAN extiende el grafo hasta llegar a un nivel en donde todas las proposiciones meta están presentes y ninguna es mutex con otra (meta). Esta es la condición necesaria para que exista un plan y a partir de la cual, GRAPHPLAN pasa a la fase de extracción de la solución, realizando una búsqueda hacia atrás de meta en meta por niveles. Para cada meta busca la acción del nivel anterior a que la alcanza, incluyendo las acciones ficticias `no-op`.

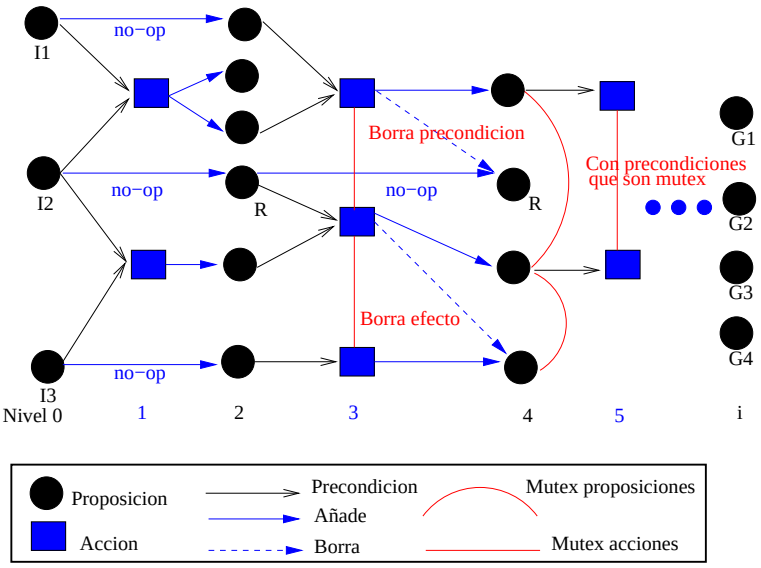

Figura 4.11: Grafo del plan generado por el algoritmo de GRAPHPLAN.

- Si a no es mutex con las acciones que ya se tienen en el plan parcial, se pasa a la siguiente meta. Cuando se tienen todas las acciones que satisfacen las metas en un nivel sin violar la relación de mutex, se hace lo mismo con las precondiciones de las acciones escogidas recursivamente. Si todas las precondiciones están en el estado inicial del problema, se ha encontrado el plan solución.
- Si a es mutex, se busca otra acción. Si todas las acciones posibles siguen siendo mutex, se retrocede a niveles anteriores de acciones para probar otra asignación. Si no hay más acciones posibles, se extiende el grafo con un nivel más de acciones y proposiciones y se repite el proceso.

Dado el alto grado de redundancia que hay en el grafo y los numerosos retrocesos del algoritmo, GRAPHPLAN introduce unas estructuras de datos llamadas memorizaciones (*memoizations*, *memos* o *nogoods*). Cada vez que se produce un fallo en la asignación de valores a un conjunto de metas en un nivel, esas metas se guardan en una tabla hash. Antes de intentar cualquier asignación de operadores en un nivel se consulta dicha tabla para evitar búsquedas innecesarias.

La única heurística que sigue GRAPHPLAN es la de persistir las metas siempre que se pueda; es decir, elegir la acción `no-op`. De esta forma evita introducir un nuevo operador en el plan. GRAPHPLAN encuentra el mejor plan paralelo, el de menor *makespan*, asumiendo que cada acción tarda una unidad de tiempo en realizarse.

A continuación se exponen los trabajos más notables de aplicación de alguna técnica de aprendizaje en GRAPHPLAN.

4.5.1. Mejoras de GRAPHPLAN por Kambhampati

El artículo [Kambhampati, 2000] resume todas las investigaciones realizadas por Kambhampati y sus colaboradores para mejorar la eficiencia de GRAPHPLAN visto como un CSP [Do and Kambhampati, 2001a]. Son las siguientes:

- Incorporación de técnicas de aprendizaje deductivo, EBL y retroceso dirigido por dependencias (*dependency directed backtracking* (DDB)). Cuando se produce un fallo en la asignación de valores a un conjunto de metas en un nivel, en lugar de almacenar todas las metas como un *memoization*, con técnicas de EBL da una explicación del fallo y restringe las metas del *memoization* a las que realmente hacen imposible una asignación válida de acciones (ver sección 3.6). Para hacer el retroceso de forma más eficiente utiliza DDB. La tabla hash originalmente utilizada por GRAPHPLAN para almacenar los *memoizations* deja de ser útil ya que ahora se buscan subconjuntos de metas; con que alguna de las metas sobre las que se está buscando una asignación forme parte de un *memoization* se puede asegurar que la búsqueda va a fallar. Utiliza los UB-Tree [Koehler *et al.*, 1997] que permiten de forma eficiente comprobar que un subconjunto de proposiciones está contenido en un conjunto.

- Ordenación dinámica de las variables: como se mencionó antes, la única heurística que introduce GRAPHPLAN es la de persistir las proposiciones antes de introducir una nueva acción, aunque no sea siempre lo más adecuado. Kambhampati propone una familia de heurísticas basadas en el nivel del plan en que se introducen por primera vez las acciones y proposiciones. Se basan en la tradicional ordenación de metas (*variable ordering heuristic*) en CSP, según las que tengan mayor número de restricciones y la ordenación de las acciones que pueden alcanzar una meta (*value ordering heuristic*) según tengan menos restricciones. Define el coste de una proposición como el nivel en el grafo en que se introduce por primera vez; intuitivamente las proposiciones con mayores restricciones aparecerán más tarde en el grafo. El coste de las acciones se define en función del coste de sus precondiciones estableciendo tres posibles criterios: el máximo, la suma, o el nivel en que por primera vez están presentes todas, que es equivalente, por la forma en que se construye el grafo, al nivel en el que se introduce la acción en el grafo menos uno. Las metas se ordenan de forma decreciente de mayor a menor coste y las acciones que satisfacen una meta en orden creciente de menor a mayor coste. Esta heurística puede ayudar a establecer un límite inferior para determinar el mínimo número de pasos que tendrá la solución; es decir, permite establecer en qué nivel del grafo empezar, como mínimo, a extraer la solución.

- También han estudiado otras formas de mejorar GRAPHPLAN en paralelo con EBL y DDB, aunque los resultados no han sido tan buenos, como la

utilización de *forward checking* y reinicialización aleatoria de la búsqueda cuando se alcanza un número determinado de retrocesos.

En un trabajo posterior Zimmerman y Kambhampati amplían estas ideas y presentan cuatro planificadores diferentes que construyen y explotan una traza del proceso de búsqueda [Zimmerman and Kambhampati, 2005]. La idea común es capturar los aspectos relevantes del primer episodio de búsqueda para que guié las búsquedas posteriores, actualizándolos dinámicamente durante todo el camino hasta la solución. Además de contemplar el proceso de búsqueda desde la perspectiva de los CSP, lo tratan como una búsqueda en el espacio de estados donde los estados son los subconjuntos de metas a los que hay que buscar una asignación válida de operadores que las satisfagan. La función generadora de nodos es el motor que encuentra acciones a las metas. Esto permite definir heurísticas y convertir el algoritmo original de búsqueda de GRAPHPLAN en un IDA* o búsqueda en haz. Las aproximaciones se pueden dividir en dos clases según la extensión de la experiencia de búsqueda capturada en la traza y según la perspectiva desde la que se contempla el proceso de búsqueda:

- Familia planificadores EGBG (Explanation Guided Backward search for Graphplan): la traza de búsqueda es completa y contemplan el proceso de búsqueda como un CSP. Construyen los planificadores EGBG y ME-EGBG en el que incorporan 6 métodos para reducir la demanda de memoria (ordenación de variables, ordenación de valores, EBL, DDB, análisis de invariantes del dominio y transiciones a otras partes del grafo del plan).
- Familia planificadores PEGG (Pilot Explanation Guided Graphplan): quitan de la traza de búsqueda toda la información relativa a mutex y contemplan el proceso de búsqueda desde la perspectiva de los IDA*. Crean los planificadores PEGG y la versión SO-PEGG que garantiza optimalidad.

4.5.2. Mejoras de GRAPHPLAN por Fox y Long

Como se expuso en la sección 3.6 Fox y Long en [Long and Fox, 1999], por medio de técnicas de análisis del dominio consiguen mejorar la velocidad de búsqueda de soluciones en GRAPHPLAN en varios órdenes de magnitud. Desarrollaron la herramienta denominada TIM que permite encontrar automáticamente tipos de objetos, es decir, conjuntos de objetos del dominio que se comportan de manera similar.

4.5.3. Mejoras de GRAPHPLAN por Afzal Upal

En el artículo [Upal, 2003] generaliza las *memoizations* de GRAPHPLAN para que puedan ser usadas en problemas distintos. Para ello hace un análisis del do-

minio antes de comenzar el proceso de planificación y aprende *memos* generales. Tiene en cuenta las metas, las condiciones iniciales y las acciones. Los primeros experimentos realizados hicieron que el tiempo en encontrar soluciones aumentase en un 20 % a 30 % debido al problema de la utilidad. Para reducirlo crearon un módulo previo que instancia todas las *memos* antes de empezar el proceso de planificación almacenándolas de la misma forma en que el algoritmo original de GRAPHPLAN lo hace. De esta forma consiguieron mejorar de un 5 a un 33 % el tiempo de búsqueda de la solución.

4.5.4. Mejoras de GRAPHPLAN por Huang et. al.

Aplicaron técnicas de aprendizaje sobre el planificador BLACKBOX en dos trabajos diferentes:

- En [Huang *et al.*, 1999] presentan un sistema de aprendizaje automático para sistemas de planificación basados en SAT que combina aspectos de aprendizaje supervisado e inductivo. El conjunto de entrenamiento es un número pequeño de problemas (alrededor de 10) junto con sus soluciones óptimas. Utiliza el planificador BLACKBOX para generar el plan de longitud paralela óptima. Dicho plan se pasa por un algoritmo de justificación que quita todas las acciones innecesarias [Fink and Yang, 1992] y por un algoritmo de inferencia de tipos que computa información de todos los operadores y objetos del dominio, TIM [Long and Fox, 1999]. El plan justificado y la información de tipos se pasan al módulo de aprendizaje, que lo utiliza de dos formas. Primero, verifica las reglas aprendidas hasta el momento con el nuevo plan para descartar inconsistencias. Luego, se extraen del plan un conjunto de ejemplos positivos y negativos que constituyen los datos de entrenamiento del módulo inductivo, que genera una o más reglas de control con un algoritmo parecido a FOIL. El proceso se repite con varios problemas, generando un conjunto de reglas en forma de conjunto de axiomas lógicos que pueden usarse tanto por el mismo planificador como por otros parecidos. Aprende reglas de selección y rechazo de una acción. Distinguen entre reglas estáticas, el cuerpo de la regla sólo depende del estado inicial y de las metas, y dinámicas, también depende del estado actual de planificación. Por ejemplo, en el dominio de logística una regla dinámica puede ser la de no mover un avión si dentro tiene un paquete cuyo destino es la ciudad en que se encuentra el avión. Una regla estática sería no descargar un paquete a no ser que esté en su destino final. BLACKBOX utiliza las reglas de rechazar acciones estáticas para podar la codificación booleana SAT. El resto de reglas se convierten en cláusulas proposicionales y se añaden a la codificación booleana SAT.

- En [Huang *et al.*, 2000] estudian la posibilidad de incorporar conocimiento de control de forma puramente declarativa en forma de restricciones adicionales en BLACKBOX. Utilizan las misma reglas de control de TLPLAN

[Bacchus and Kabanza, 2000]. En TLPLAN se incorpora conocimiento de control codificándolo directamente en los operadores del dominio en forma de fórmulas lógicas temporales. Es un planificador que hace encadenamiento de búsqueda de la solución hacia adelante, con lo cual, pasar esas reglas a planificación con encadenamiento hacia atrás no es trivial.

4.6. Planificador TGP

TGP es un planificador temporal que utiliza las técnicas de planificación basadas en grafos de planes. Sigue un algoritmo similar al de GRAPHPLAN aunque con algunas variaciones, que se explican en las secciones siguientes, para poder adaptar el razonamiento temporal.

4.6.1. Fase generación del grafo del plan

GRAPHPLAN genera el grafo del plan alternando niveles consecutivos de proposiciones y acciones. El primer nivel lo constituyen las proposiciones del estado inicial del problema a resolver. El siguiente nivel está formado por las acciones cuyas precondiciones están presentes en el nivel anterior. El siguiente nivel lo forman los efectos de las acciones del nivel anterior más todas las proposiciones que ya existían gracias a la acción ficticia `no-op` que permite la persistencia de metas. La expansión del grafo continúa hasta encontrar un nivel en que están presentes todas las proposiciones que representan las metas del problema y no son mutex entre sí. En el caso de TGP, al tener las acciones duración ya no se alternan los niveles de proposiciones y acciones exclusivas si no que en cada nivel pueden aparecer mezcladas. El nivel 0 y el nivel 1 del grafo se construye igual que en GRAPHPLAN, las proposiciones del estado inicial y las acciones cuyas precondiciones estén en el nivel cero. Cada acción del nivel 1 tiene una duración diferente con lo que sus efectos en lugar de añadirse al siguiente nivel se sitúan en el nivel $d + 1$, siendo d la duración de la acción. A partir del nivel 3 se van añadiendo acciones cuyas precondiciones estén presentes en el nivel anterior. Los efectos de dichas acciones en lugar de añadirlos en el siguiente nivel, se colocan en el nivel $d + i$ (siendo i el nivel de la acción). De esta forma, el nivel del grafo representa el instante de tiempo más temprano en que una acción puede empezar a ejecutarse o en el que se hace cierta por primera vez una proposición. Esto permite extender el lenguaje STRIPS de representación de acciones para hacer el tratamiento temporal:

- Las acciones tienen un tiempo de inicio, t_i y una duración, d.
- Para que una acción se ejecute, todas sus precondiciones deben ser ciertas en el instante t_i.

- Los efectos de las acciones quedan indefinidos hasta que se alcanza el instante $t_i + d$.

En TGP las relaciones de mutex definidas en GRAPHPLAN se siguen manteniendo aunque es preciso hacer algunas extensiones que se explican en el siguiente apartado. Los *nogoods* (o *memoizations*) de GRAPHPLAN siguen igual (conjuntos de metas que no pueden ser satisfechas en un nivel).

La representación del grafo del plan se puede simplificar teniendo en cuenta las siguientes observaciones:

- Las proposiciones y las acciones son monotónicamente crecientes: si aparecen en un nivel, siguen presentes en los niveles superiores.
- Los mutex son monotónicamente decrecientes: si dos nodos son mutex en un nivel, la relación de mutex se mantiene en los niveles anteriores.
- Los *nogoods* son monotónicamente decrecientes: si un conjunto de metas no se puede resolver en un nivel tampoco se podrá alcanzar en los niveles inferiores.

En vez de representar el grafo con sus niveles y los enlaces entre precondiciones y efectos de las acciones, cada acción, proposición, mutex y *nogood* se etiqueta con un número. En el caso de acciones y proposiciones representa el primer nivel del grafo en que aparecen; es decir, el primer instante en que la acción se puede ejecutar o la proposición se hace cierta. En el caso de mutex y *nogoods* representan el último nivel del grafo en que la relación se mantiene.

4.6.2. Ampliación de las relaciones de mutex

Amplia las relaciones de mutex en dos sentidos:

- Además de las consideradas por GRAPHPLAN entre acción-acción y entre proposición-proposición, define una relación más entre acción-proposición. Una acción es mutex con una proposición si alguna de las precondiciones o efectos de la acción es mutex con la proposición.
- Diferencia entre mutex eternos y condicionales. Los primeros existen siempre, en todos los niveles del grafo, frente a los *cmutex* que pueden desaparecer según crece el nivel del grafo del plan (al introducirse nuevas acciones y proposiciones).

Al tener duración las acciones, las condiciones para verificar los mutex se definen mediante relaciones de desigualdad referidas a:

1. Duración de una acción.
2. Tiempo en que una acción o proposición aparece por primera vez en el grafo del plan.
3. Tiempo teórico más temprano en que podría terminar una acción.
4. Instante de tiempo en que comienza la acción o una proposición se hace cierta.
5. Instante de tiempo en que termina realmente la acción.

Las relaciones de mutex que define TGP son:
Mutex eterno (*emutex*):

- Entre dos proposiciones P y Q: si P es la negación de Q.
- Entre acción-proposición, a y P: si noP es una precondición o efecto de a.
- Entre acción-acción, a y b: a o b borran precondiciones o efectos de la otra, o bien, a y b tienen precondiciones que son *emutex*.

Mutex condicionales (*cmutex*):

- Entre dos proposiciones: P y Q son *cmutex* cuando P es mutex con todas las acciones que soportan Q y viceversa. Una acción soporta P cuando entre sus efectos añade P y además termina antes de que P empiece.
- Entre acción-proposición: a y P son *cmutex* cuando P es mutex con alguna precondición de a, o bien, a es mutex con todas las acciones que soportan P.
- Entre acción-acción: a y b son *cmutex* cuando a es mutex con alguna precondición de b o viceversa.

La figura 4.12 presenta algunos ejemplos de relaciones de mutex definidas en GRAPHPLAN.

Los mutex que TGP definiría para el ejemplo de la figura serían:

- a es *emutex* con b porque a borra una precondición de b, R.
- b es *emutex* con c porque b borra un efecto de c.
- T es *cmutex* con U porque en el nivel i del grafo las únicas acciones que soportan ambas proposiciones son mutex. Sin embargo, podría ocurrir que en niveles del grafo superiores apareciera una nueva acción que tenga entre sus efectos T o U sin ser mutex con la otra acción que soporta a la otra proposición.

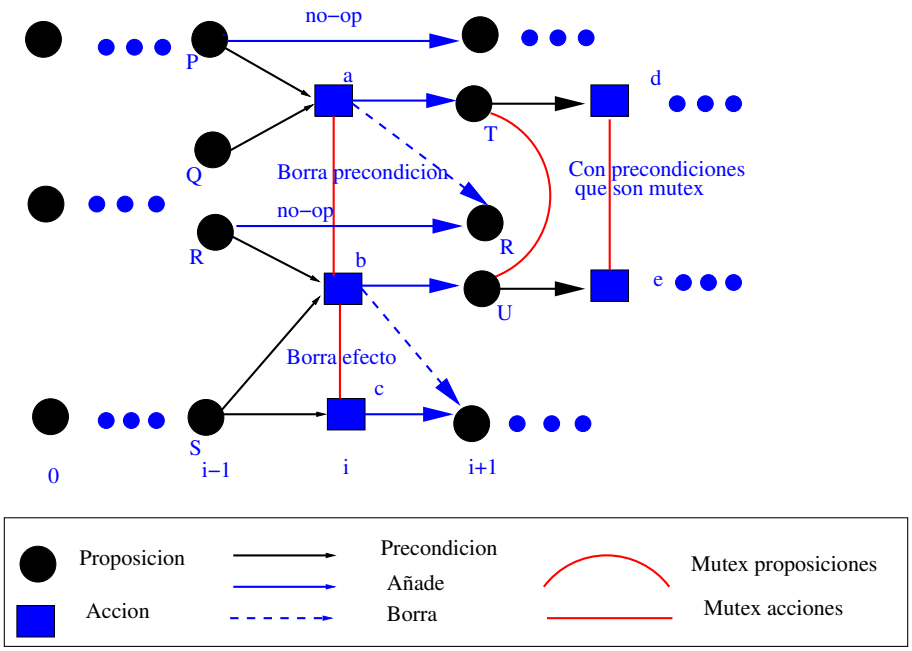

Figura 4.12: Relación de mutex en GRAPHPLAN.

- d es *cmutex* con e porque tienen precondiciones que son *cmutex* entre sí, T y U.
- d es *cmutex* con U porque d tiene una precondición que es *cmutex* con U, T.
- e es *cmutex* con T porque e tiene una precondición que es *cmutex* con T, U.

En la estructura que utiliza TGP para almacenar los mutex añade un campo para representar el tipo de mutex (PP entre proposiciones, AA entre acciones y AP entre acción-proposición) y otro para definir o explicar la relación, consistente en dos números $[x, y]$. Una relación de mutex entre dos nodos está activa si el instante de tiempo en que empieza la acción (o en que se hace cierta la proposición) del primer nodo de la relación es menor que x y el instante de tiempo en que empieza la acción (o en que se hace cierta la proposición) del segundo nodo de la relación es menor que y. En el caso de ser mutex eternos x o y tiene valor infinito.

4.6.3. Fase de extracción de la solución

Una vez que el grafo del plan se ha extendido hasta un nivel i en que están presentes todas las metas del problema sin mutex activos entre ellas, TGP realiza una búsqueda hacia atrás del plan solución.

Esta búsqueda está implementada utilizando dos estructuras de datos, G y A. G es una lista de pares $< g_i, t_i >$ donde g_i es una meta y t_i el instante de tiempo

en que aparece por primera vez en el grafo del plan. Inicialmente G tiene un par $< G_i, t_{Gi} >$ por cada meta del problema. La segunda estructura es A que almacena el plan en construcción como pares $< a_i, s_i >$, donde a_i representa una acción del plan y s_i el instante de tiempo en que empieza a ejecutarse. Inicialmente A está vacía. El algoritmo de extracción de la solución de TGP repite los siguientes pasos hasta que G esté vacía:

1. Saca $< g, t >$ de G.
2. Si $t = 0$ y $g \notin$ Estado Inicial, entonces falla (retrocede).
3. Hacer $P = \varnothing$, $G' = \varnothing$.
4. Si puede *persistir* g, $t < i$ y no es mutex con ninguna otra acción de A: añadir $< g, t >$ a P y a G'.
5. Si no, hacer $O =$ conjunto de operadores o_i que tienen como efecto g y son ciertas en un instante de tiempo $t_i <= t$. Está ordenado de forma creciente en el tiempo.
6. Elegir o de O que no sea mutex con ninguna otra acción en A ni con ninguna proposición en P. Añadir $< o, t - d >$ (siendo d la duración de o) a A, y para cada precondición $p_j o$ de o, añadir $< p_j o, t - d >$ a G'. Si no existe dicha o, retroceder.
7. Repetir en nivel (i-1) con $G = G'$.

Si no es posible encontrar una asignación válida para G, incrementa en un nivel el grafo del plan y repite el proceso.

El algoritmo, a un nivel más bajo de detalle, es el que se presenta en la figura 4.14. Las funciones básicas del algoritmo se describen a continuación:

- **PLAN**: mientras el tiempo no exceda el tiempo máximo de resolución establecido y no se encuentre un plan válido repite las siguientes acciones: extiende el grafo del plan y llama a *FIND-PLANS-IN-PG* para que encuentre un plan válido en ese nivel. Si encuentra la solución devuelve el plan y termina; si no, incrementa el tiempo y continua.
- **FIND-PLANS-IN-PG**: ordena todas las acciones que puedan satisfacer una meta en orden creciente en el tiempo (primero se prueban las acciones que están antes en el grafo del plan). Verifica que las metas del problema están presentes en el grafo del plan para el tiempo actual y no hay ningún mutex activo entre ellas. Si es así, llama a *FIND-PLAN-IN-PG1* para que encuentre un plan válido que satisfaga las metas del problema. Si lo encuentra, devuelve el plan solución; si no devuelve nil.

- **FIND-PLAN-IN-PG1**: si el tiempo es 0 o no quedan metas por satisfacer, genera un plan con las acciones que tenga el plan parcial y lo devuelve. Si no, comprueba si el conjunto de metas que se le pasa está almacenada como *nogood*, en cuyo caso devuelve nil. En caso contrario, llama a la función *ASSIGN-GOALS* para que encuentre las acciones válidas que satisfagan todas las metas. Si no encuentra tal asignación, almacena las metas como un nuevo conjunto *nogood* y continúa. Si encuentra una asignación válida, devuelve el plan.

- **ASSIGN-GOALS**: es una función recursiva que va buscando meta por meta una acción que la satisfaga. Mantiene dos listas: la de acciones que van constituyendo el plan parcial y la de persistencias; es decir, las metas a las que aplica el operador ficticio `no-op` para satisfacerlas (en el nivel actual no añade ningún operador que las satisfagan porque se asignará en un nivel inferior). Para cada meta, primero intenta persistir, verificando que no se viole ninguna relación de mutex entre el conjunto de acciones y persistencias que lleva. Si la persistencia no funciona, prueba con el primer operador que hay para satisfacer esa meta. Si el operador es mutex con alguna de las acciones o persistencias que ya hay, prueba con los siguientes operadores hasta encontrar uno que no viola ninguna relación de mutex. Dicho operador se añade a la lista de acciones y se repite el mismo proceso con el resto de metas. Una vez encontrada una lista de acciones y persistencias válidas para todas las metas se llama a la función *DESCEND*.

- **DESCEND**: genera un nuevo conjunto de metas con las precondiciones de las acciones elegidas más las persistencias y vuelve a llamar a la función FIND-PLAN-IN-PG1.

En la figura 4.13 se muestra un ejemplo de cómo actuaría el algoritmo en GRAPHPLAN que es prácticamente igual al de TGP salvo las diferencias explicadas en las secciones anteriores. Supóngase que hay cuatro metas G_1, G_2, G_3, G_4 en el nivel i. Se busca un conjunto de acciones que satisfaga todas ellas (y que no sean mutex) en el nivel de acciones $i-1$. Supóngase que encuentra $Plan = [A_1, A_2, A_3, A_4]$. Para cada una de esas acciones se buscan sus precondiciones y se intenta encontrar otro conjunto de acciones que satisfagan las nuevas metas en el nivel $i-2$. Es decir, se buscan acciones para satisfacer $P_1, P_2, P_3, P_4, P_5, P_6$. El algoritmo empieza con la primera meta P_1. Sólo hay una acción que la satisfaga, A_5, que se añade al plan ($Plan = [A_1, A_2, A_3, A_4, A_5]$). P_2 tiene dos posibles acciones que la soporta, A_6 y A_{11}. Prueba con la primera ($Plan = [A_1, A_2, A_3, A_4, A_5, A_6]$) y pasa a la meta P_3, añadiendo A_7 al plan ($Plan = [A_1, A_2, A_3, A_4, A_5, A_6, A_7]$). Al intentar buscar una acción para P_4, comprueba que las únicas acciones posibles son mutex con alguna acción del plan; A_8 es mutex con A_6 y A_9 es mutex con A_5. Entonces, tiene que retroceder a P_3, pero como no hay ninguna otra acción que la soporte, retrocede a P_2 y prueba con la acción A_{11} ($Plan = [A_1, A_2, A_3, A_4, A_5, A_{11}]$). Pasa a P_3 y la única acción posible es A_7,

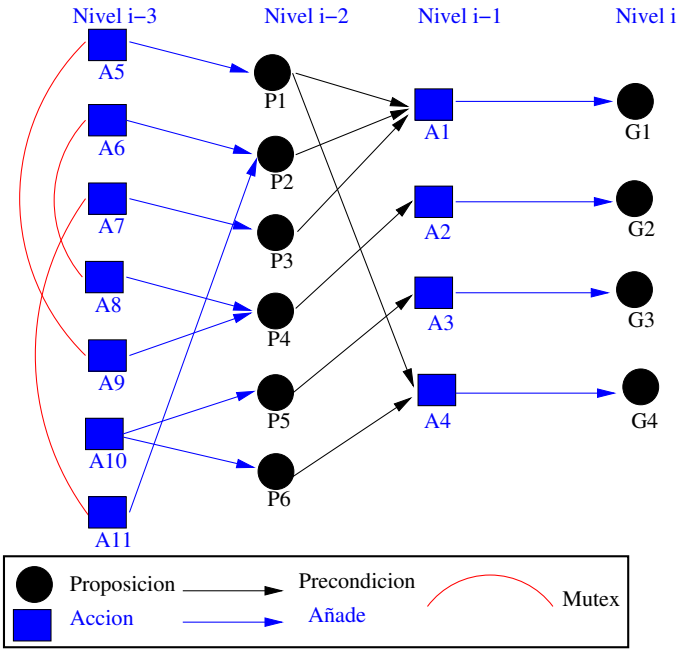

Figura 4.13: Ejemplo de un episodio de búsqueda hacia atrás en el grafo del plan realizado por GRAPHPLAN.

que es mutex con A_{11} con lo que tendría que volver a retroceder. Al no haber más acciones posibles, almacena como *nogood* el conjunto $\{P_1, P_2, P_3, P_4, P_5, P_6\}$ y retrocede al nivel i para buscar otra asignación válida para $\{G_1, G_2, G_3, G_4\}$, pero, como no hay más acciones posibles, almacena $\{G_1, G_2, G_3, G_4\}$ como un nuevo *nogood* y tendría que expandir el grafo del plan en un nivel más.

```
PLAN (problem)
   *plan-graph* = initialize-plan-graph(problem domain)
   MIENTRAS time < max_time
      update-support(*plan-graph* time) ;;extending the plan graph
      plans = FIND-PLANS-IN-PG(*plan-graph* time)
      IF   (plan-p plans) THEN return plans
      ELSE time = time + delta

FIND-PLANS-IN-PG(plan-graph time)
   sort-establishers(plan-graph)
   goal-props = find-goal-props(plan-graph)
   WHEN check-goal-props(goal-props time) ;;all problem goals no mutex
      plan = FIND-PLAN-IN-PG1 (goal-props nil time)
      IF (plan-p plan) THEN return plan
      ELSE return nil

FIND-PLAN-IN-PG1 (goal-props actions time)
   IF (time = 0) or (goal-props = 0 ) THEN return make-plan(actions)
   ELSE
      IF memoized-failute-p(goal-props action time) THEN return nil
      ELSE
         result = ASSIGN-GOALS(goal-props actions nil nil)
         IF (plan-p result) THEN return result
         ELSE return memoize-as-failure(result time)

ASSIGN-GOALS (remaining-goal-props actions persistences conflict-set)
   IF remaining-goal-props
      THEN
         earlier = time - delta
         goal = first-goal-of(remaining-goal-props)
         rest = rest-goals-of(remaining-goal-props)
         conflict-set = conflict-set + goal
         IF can-persist(goal time) THEN persistences=goal+persistences
                                        RECUR(actions persistences rest conflict-set)
         ELSE
            IF establisher = find-new-establisher(goal time)
               THEN actions = establisher + actions
                    RECUR(actions persistences rest conflict-set)
               ELSE return conflict-set
         ;; Found a set of actions that work at this level,
         ;;so descend to the next level
         DESCEND(actions persistences earlier time)

RECUR(action persistences rest conflict-set)
   result = ASSIGN-GOALS(rest actions persistences conflict-set)
   IF (plan-p result) THEN return result
   ELSE IF member(goal result)
           THEN conflict-set = result+conflict-set
                return conflict-set
           ELSE return-from assign-goals result

DESCEND(actions persistences earlier time)
   preconditions = find-preconditios-of (actions)
   FIND-PLAN-IN-PG1((preconditions+persistences) actions earlier)
```

Figura 4.14: Algoritmo de planificación de TGP.

Capítulo 5

Lenguaje de representación del conocimiento de control

Uno de los objetivos de la tesis doctoral es estudiar la posibilidad de transferir conocimiento de control entre las diferentes técnicas de planificación y a los propios sistemas de aprendizaje. Para ello es necesario definir un lenguaje común que represente el conocimiento de control que pueda ser utilizado por cualquier técnica de planificación y ser extraído por los sistemas de aprendizaje. Debe ser suficientemente expresivo para poder cubrir todo el conocimiento y suficientemente flexible para poder añadir cualquier particularidad de un planificador concreto. En la tesis se ha partido del lenguaje utilizado en PRODIGY, haciendo las adaptaciones pertinentes para cada una de las técnicas de planificación utilizadas en esta tesis y comprobando experimentalmente en cada caso su validez. Para que el lenguaje sea válido se debe tener en cuenta lo siguiente: que sea declarativo, para poder generarlo automáticamente, y que sea posible interpretarlo (operacionalización); es decir, que pueda ser utilizado por un sistema de planificación para tomar las decisiones, lo que implica que el lenguaje debe poder acceder al meta-estado del planificador.

En la siguiente sección se describe el lenguaje de partida, el utilizado en el planificador PRODIGY, y en las secciones donde se describen los nuevos sistemas de aprendizaje desarrollados en la tesis, se especifican las variantes utilizadas. El lenguaje utilizado en PRODIGY ofrece la flexibilidad de ser ampliado fácilmente mediante la definición de nuevos meta-predicados. La última sección de este capítulo explica cómo realizarlo. Previamente, la sección 5.2 resume el proceso de equiparación y ejecución de reglas realizado por PRODIGY.

5.1. Descripción del lenguaje de PRODIGY

El modelo para representar el conocimiento de control se toma del lenguaje utilizado por PRODIGY que permite la definición de reglas de control para guiar el proceso de búsqueda del plan. Las reglas constan de tres partes: el nombre de la regla, la parte condicional o requisitos que se debe cumplir en el meta-estado de planificación para poder dispararse, y la parte del consecuente o decisión a tomar. La parte condicional se compone de meta-predicados (dado que algunos de sus argumentos representan predicados), que describen características del meta-estado actual de planificación. La parte del consecuente da lugar a tres tipos de reglas diferentes: de selección, de rechazo y de preferencia. Las de selección determinan que hay un subconjunto de alternativas que deben tenerse en cuenta descartando el resto de nodos seleccionados. Las reglas de rechazo le obligan a descartar un subconjunto de alternativas de entre las seleccionadas. Y las reglas de preferencia determinan un orden en el que estudiar las alternativas restantes. Las acciones, a su vez, pueden ser variadas y depender del planificador: elegir la meta sobre la que trabajar, elegir un operador para resolver una meta, elegir los *bindings*[1] apropiados para el operador seleccionado, decidir aplicar un operador cuyos *bindings* ya han sido instanciados o trabajar en una nueva meta. Debería haber una acción por cada punto de decisión del planificador. Por ejemplo, en planificación jerárquica, puede ser el método de expansión a utilizar en cada refinamiento; en planificación POP, la forma de resolver una amenaza; en planificación basada en grafos, las metas que en cada nivel pueden persistir. Además, se deben adaptar a la manera en que el planificador toma esas decisiones. Por ejemplo, PRODIGY para seleccionar un operador instanciado lo realiza en dos nodos: primero elige el operador y luego los valores de sus variables. Sin embargo, en planificación basada en grafos, la misma decisión se realiza en un sólo nodo; y en los planificadores POP se diferencia entre si el operador a utilizar ha sido previamente utilizado en el plan o no, dando lugar a dos tipos de decisión diferentes.

En la figura 5.1 se muestra el lenguaje utilizado en las reglas de PRODIGY en notación BNF y en la figura 5.2 un ejemplo de regla de control en el dominio Logistics. La parte de precondiciones normalmente consiste en una conjunción de meta-predicados, aunque también admite la disyunción. PRODIGY cuenta con un conjunto de meta-predicados básicos que recogen las principales características del proceso de planificación (para una completa descripción de todos consultar [Minton *et al.*, 1989]). Algunos de ellos utilizados en esta tesis son:

- True-in-state <literal>: comprueba si <literal> es cierto en el estado actual de planificación. En la regla del ejemplo, el meta-predicado `(true-in-state (at <tr1> <po1>))` comprueba si el objeto representado por la variable `<tr1>` se encuentra en el lugar indicado por la variable `<po1>`.

[1] Instanciaciones de operador (en términos lógicos, sustituciones de valores a variables).

regla-control = (**CONTROL-RULE** nombre-regla
(**if** precondiciones)
(**then** consecuente))

precondiciones = (**and** (meta-predicado)$^{+}$)

consecuente = seleccion|rechazo|preferencia|**SUB-GOAL**|**APPLY**

seleccion = **SELECT** tipo-candidato valor

rechazo = **REJECT** tipo-candidato valor

preferencia = **PREFER** tipo-candidato valor-preferido otra-preferencia

tipo-candidato= **NODE**|**GOAL**|**OPERATOR**|**BINDINGS**

Figura 5.1: Lenguaje reglas de control de PRODIGY en notación BNF.

```
(control-rule regla-logistics-aips98-l3-9-e2-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <po1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob3> <tr1>))
          (some-candidate-goals ((at <ob1> <po0>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob3> object)
          (type-of-object <ob1> object)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr1>) (<loc> . <po1>))))
```

Figura 5.2: Ejemplo regla de control en el lenguaje de PRODIGY.

- Current-goal <meta>: comprueba si <meta> es la meta actual que está resolviendo el planificador. En el ejemplo, se tiene `(current-goal (at <ob3> <po1>))` que comprueba que la meta sea que un objeto, representado por la variable `<ob3>`, esté en un lugar, representado por la variable `<po1>`.

- Some-candidate-goals <metas>: comprueba si alguno de los literales en la lista de <metas> es una meta pendiente. En el ejemplo, `(some-candidate-goals ((at <ob1> <po0>)))` indica que debe existir una meta pendiente de que un objeto esté en otro lugar. Este meta-predicado también comprueba que los objetos asignados a variables con distintos nombres sean diferentes.

- Current-operator <operador>: comprueba si <operador> es el nombre del operador actualmente seleccionado para resolver una meta. En el ejemplo, el operador seleccionado para resolver la meta es `unload-truck` porque se indica en el meta-predicado `(current-operator unload-truck)`.

- Type-of-object <var> `tipo`: comprueba si el tipo del objeto, según la jerarquía de tipos en la definición del dominio, de la variable <var> es `tipo`. En el ejemplo, para cada una de las variables que intervienen en la definición de las reglas se define uno de estos meta-predicados, como `(type-of-object <tr1> truck)` que indica que el objeto representado por la variable `<tr1>` sea un camión.

Los argumentos de los predicados pueden incluir variables y constantes. La regla del ejemplo es de selección de variables del operador `unload-truck`. La única variable libre de este operador, cuando se elige para conseguir la meta de que un objeto se encuentre en un lugar, es el camión desde el que descargar el objeto. Si el objeto se encuentra dentro de un camión, representado por el meta-predicado `(true-in-state (inside <ob3> <tr1>))`, la regla elige ese camión.

Las reglas definidas con la sintaxis descrita se utilizan en el proceso de búsqueda realizado por PRODIGY según el algoritmo representado en la figura 5.3. El algoritmo incluye cuatro puntos de decisión sobre los que puede haber retroceso. Cada una de estas decisiones puede ser condicionada por reglas de cierto tipo (`SUB-GOAL, APPLY, node, goal, operator, bindings`). Observando el algoritmo, los puntos de decisión se producen en el paso 3 del bucle `While`: el primero para elegir entre aplicar un operador o resolver otra meta del conjunto de metas pendientes y las reglas que pueden guiar esta decisión son las de `SUB-GOAL` y las de `APPLY`; el segundo, para elegir una de las metas pendientes y puede ser guiada por reglas de tipo `goal`; el tercero, para seleccionar el operador que resuelva la meta elegida en el paso anterior y puede ser guiada por reglas de tipo `operator`; y el último para seleccionar las variables libres del operador seleccionado que puede ser guiada por reglas de tipo `bindings`. Todavía existe un punto más de decisión para seleccionar el operador instanciado a aplicar, pero no

puede ser guiada por reglas. El algoritmo para seleccionar el conjunto de decisiones posibles en cada punto de decisión de acuerdo con las reglas se describe en la figura 5.4.

Cada meta-predicado está implementado como una función en LISP por lo que es factible adaptarlas a otros sistemas de planificación y definir nuevos meta-predicados. Antes de explicar cómo hacer estas definiciones, conviene entender el proceso de equiparación de reglas realizado por PRODIGY.

5.2. Proceso de equiparación de reglas

Para verificar si una regla se cumple, PRODIGY realiza un proceso de equiparación simple lineal, salvo en el caso de las reglas de instanciación de operador cuya equiparación se realiza mediante una red RETE [Forgy, 1982]. Se comprueba secuencialmente las formas de asignar valores de verdad a cada meta-predicado y a sus variables generándose una lista con las posibles asignaciones de las variables que contengan las reglas, a constantes del proceso actual de planificación.[2] Se toma el primer meta-predicado de la regla y se obtiene una lista con las asignaciones posibles de las variables a contantes. Esta lista se compone de un conjunto de sustituciones, donde cada sustitución es de la forma:

```
((var1 . valor1) (var2 . valor2) ...(vanN . valorN))
```

o en BNF:

```
sustituciones = (B*)
B=((var . valor)*)
```

Cada meta-predicado debe devolver uno de los siguientes valores: *true*, si es cierto, y no se añade información a la lista de sustituciones; *nil*, si no se cumple; o nuevas sustituciones que se juntan con las anteriores. Con cada una de las posibles asignaciones de la lista se comprueba si se cumple el segundo meta-predicado y se añaden o crean nuevas asignaciones a las ya existentes. Se sigue así con todas las condiciones. Para todas las sustituciones de valor de variables de la parte izquierda que la hacen cierta, σ_i, se tiene una instanciación de la regla de control. Ejecutar una regla de control significa que se tomará la decisión especificada en la parte derecha de la regla después de sustituir las variables en las mismas utilizando cada σ_i. Este proceso se podría hacer más eficientemente utilizando el algoritmo RETE en todos los tipos de reglas.

[2] También se denominan *bindings* y, aunque tengan la misma sintaxis que los *bindings* de los operadores, su significado es diferente.

PRODIGY4.0(S, G, D, C)

S: estado del problema
G: conjunto de metas pendientes
D: definición del dominio
C: conjunto de reglas (conocimiento de control)
P: plan. Inicialmente $\varnothing$
O: operador del dominio sin instanciar
B: sustitución (*bindings*) de las variables de un operador
O_B: operador O instanciado con las variables B
A: conjunto de operadores aplicables
G_0: precondiciones de todos los operadores en O

While $G \nsubseteq S$ y árbol de búsqueda no agotado

1. $G = G_0 - S$
2. $A \leftarrow \varnothing$
   Forall $O_B \in O \mid$ precondiciones $(O_B) \subseteq S$ do $A \leftarrow A \cup \{O_B\}$
3. If select-subgoal-or-apply(G, A, S, C)=*subgoal*
   Then
   (.1) $G \leftarrow$ select-goal(G, S, C)
   (.2) $O \leftarrow$ select-relevant-operator(G, D, S, C)
   (.3) $O \leftarrow$ select-bindings(O, S, C)
   (.4) $O \leftarrow O \cup O_B$
   Else
   (.1) $O_B \leftarrow$ select-applicable-operator(A, S, C)
   (.2) $S \leftarrow$ apply(O_B, S)
   (.3) $O \leftarrow O - O_B$
   (.4) $P \leftarrow$ enqueue-at-end(P, O_B)
4. Si hay una razón para suspender la búsqueda
   Entonces retroceder

$ST \leftarrow$ árbol de búsqueda

Devolver P y ST

Figura 5.3: Algoritmo de búsqueda en PRODIGY.

Comprobar-reglas(T, Op, C)

T: tipo de regla
Op: conjunto de opciones posibles para el punto de decisión
C: conjunto de reglas (conocimiento de control)

$0 = Op$; $0' = \varnothing$

1. $\forall$ regla de SELECT T que seleccione opciones O''
   $0' = (0' \cup 0'') \cap O$
   $0 = 0'$
2. $\forall$ regla de REJECT T que rechace opciones O''
   $0 = 0 - 0''$
3. reordenar O de acuerdo con las reglas de PREFER T

devolver 0

Figura 5.4: Algoritmo para seleccionar las opciones en cada punto de decisión de acuerdo a un conjunto de reglas en PRODIGY.

5.3. Nuevos meta-predicados

En el algoritmo de equiparación de reglas descrito antes, por cada precondición de las reglas se llama a una función de nombre igual que el meta-predicado sustituyendo las variables que intervienen en el argumento del meta-predicado por los valores indicados en cada sustitución actual. Como se ha dicho, la función puede devolver: otro conjunto de sustituciones que se juntan a los ya obtenidos previamente; *true*, en cuyo caso el algoritmo continúa equiparando el resto de meta-predicados de la regla sin agregar nuevas sustituciones; o *nil*, en cuyo caso la regla falla para esa sustitución y el proceso de equiparación para la sustitución correspondiente y la regla se detiene. Por tanto, para definir un nuevo meta-predicado basta con implementar una función que tome las entradas adecuadas y que devuelva la salida correspondiente. Si se necesita algún parámetro más de entrada habría que pasárselo en la llamada a la función, modificando la función pertinente del módulo de equiparación de reglas que invoca dichas funciones. Por ejemplo, para definir un meta-predicado que determine si una acción ya ha sido utilizada en el plan parcial de un planificador POP, suponiendo que `*CURRENTNODE*` es el nodo actual, se implementaría la función representada en la figura 5.5

Los argumentos de entrada son la acción que se quiere comprobar; es decir, el nombre de la variable utilizada en la definición de la regla. La función `get-actions-from-partial-plan` obtiene la lista de acciones del plan parcial construido hasta ese momento. La función `find-matching` encuen-

```
(DEFUN operator-in-plan (action binding)
  (declare (special *CURRENTNODE*))
  (let ((actions (get-actions-from-partial-plan *CURRENTNODE*)))

    (if action
       (let ((bins (find-matching action actions binding)))
         (if (and bins (listp bins))
                (verify-constraints action bins :str t) bins))
       (null actions))))
```

Figura 5.5: Ejemplo de función en LISP para definir un meta-predicado.

tra todos los valores que puede tomar la variable `action` recorriendo la lista `actions` y devuelve otra lista de *bindings*. Por tanto, la función `operator-in-plan` puede devolver los siguientes valores: el subconjunto de sustituciones de `bins` que cumplen las restricciones comprobadas en la función `verify-constraints`, *nil* si `find-matching` no encuentra asignaciones posibles o *true* si `action` es *nil* y no hay acciones en el plan parcial.

Parte II

Sistemas de aprendizaje

Uno de los objetivos de la tesis doctoral es desarrollar sistemas de aprendizaje automático aplicables a los planificadores base para extraer conocimiento de control que incrementen su eficiencia resolviendo nuevos problemas. Se han implementado dos sistemas de aprendizaje diferentes, uno aplicable a planificación mixta jerárquica, en particular HTN, y POP y otro a planificación basada en grafos de planes. La razón de estas elecciones es doble: porque estas técnicas de planificación se están utilizando ampliamente tanto en investigaciones como en aplicaciones y porque existen muy pocos sistemas de aprendizaje centrados en incrementar la eficiencia en resoluciones futuras para ellas.

En esta parte se explican los dos sistemas de aprendizaje desarrollados. Primero se describe HEBL diseñado para el planificador híbrido HTN-POP, HYBIS. La descripción detallada del planificador se expuso en la sección 4.4. Son varias las razones para usar este planificador:

- Permite una descripción modular de los métodos de expansión posibilitando el aprendizaje del mejor método en cada situación. La mayoría de los HTN son "ad-hoc" imposibilitando dicho aprendizaje.
- Utiliza otra técnica de planificación, POP, de la que también es posible obtener conocimiento de control. Desarrollando un único sistema de aprendizaje, se puede estudiar la influencia del aprendizaje sobre dos técnicas de planificación diferentes.
- Resuelve problemas de manufacturación del mundo real.

El segundo sistema de aprendizaje implementado es GEBL. Se ha utilizado el planificador TGP que es la versión temporal de GRAPHPLAN, por estar implementado en el lenguaje LISP y por incorporar una serie de mejoras respecto al algoritmo original de Graphplan. Sólo se aprende de la parte de GRAPHPLAN no temporal en esta tesis. El último capítulo pretende ser una propuesta de metodología de aprendizaje de control, utilizando EBL, aplicable a distintos paradigmas de planificación.

Capítulo 6

Sistema de aprendizaje HEBL

En este capítulo se presenta un sistema, HEBL , que aprende conocimiento de control para mejorar la eficiencia de un planificador especialmente diseñado para la resolución de problemas de manufacturación, (HYBIS [Castillo *et al.*, 2000]). HYBIS es un planificador jerárquico con una representación de sus operadores basada en autómatas finitos que es capaz de obtener secuencias de control en un proceso de manufacturación. Mezcla técnicas de planificación jerárquica (HTN) [Currie and Tate, 1991] y Planificación de Orden Parcial (POP) [Weld, 1994]. Los problemas en HYBIS se van descomponiendo jerárquicamente en problemas de mayor a menor nivel de abstracción. En cada nivel se genera un plan utilizando POCL (*Partial Order Causal Link*). Una vez encontrado el plan en un nivel, cada acción se transforma en un problema del nivel inferior según un método de expansión. El método de aprendizaje propuesto está basado en HAMLET siendo dos los aspectos fundamentales a destacar: el aprendizaje durante la generación de planes de orden parcial y el aprendizaje durante la descomposición jerárquica.

6.1. Descripción

En manufacturación un problema se describe por las transformaciones necesarias realizables sobre un elemento/s base para producir un producto elaborado. El dominio es el modelo de conocimiento del sistema de manufacturación. El modelo se puede representar como un conjunto de agentes, donde cada agente representa un dispositivo industrial, junto con las operaciones o acciones que es capaz de realizar; y un conjunto de axiomas que describen hechos que siempre se verifican. Cada agente es descrito jerárquicamente según las diferentes partes de las que esté compuesto, que a su vez pueden ser otros agentes. El comportamiento de cualquier agente de la jerarquía se describe mediante un autómata finito de forma que los estados del autómata se corresponden con un conjunto de diferentes estados en los que se pueda encontrar un agente y las transiciones del autómata se correspon-

den con un conjunto de acciones que puede ejecutar. En este contexto, un problema consiste en un estado inicial representado por un conjunto de literales que describen tanto el sistema de manufacturación como los productos base; y unas metas, que representan el conjunto de literales que describen las transformaciones necesarias para obtener el producto manufacturado. La solución a un problema es una secuencia ordenada de acciones de los agentes del dominio que consiguen la meta, empezando desde el estado inicial especificado.

Como ocurre en todos los planificadores independientes del dominio, HYBIS no encuentra el mejor plan rápidamente porque gasta mucho tiempo estudiando alternativas no válidas antes de llegar a la solución. Para evitar esto, se propone adquirir conocimiento automáticamente para, posteriormente, guiar el proceso de planificación. Este conocimiento se adquiere por la experiencia de resolver problemas anteriores al generar una explicación de los pasos seguidos por el propio proceso de planificación.

6.2. Proceso de aprendizaje

El planificador HYBIS es completo pero no siempre es eficiente incluso con las heurísticas independientes del dominio que utiliza. En algunos casos dichas heurísticas no guían al planificador por el mejor camino de búsqueda. Sin embargo, aprendiendo de la propia experiencia de planificación se pueden identificar los puntos en que las heurísticas fallan para extraer el conocimiento de control necesario que permitan podar las partes del árbol de búsqueda erróneas y aumentar así la eficiencia en resoluciones futuras de nuevos problemas.

Para adquirir conocimiento de control de este planificador HTN-POCL se sigue un algoritmo con tres pasos:

1. El planificador se ejecuta sobre un problema y se etiquetan todos los nodos del árbol de búsqueda de forma que se puedan identificar los puntos de decisión de éxito, los nodos que conducen a la solución.

2. En los puntos de decisión de éxito se generan reglas de control de forma que cuando se vuelva a ejecutar el planificador otra vez con las reglas, vaya directamente a la solución.

3. Las constantes y variables en las reglas de control se generalizan para que puedan ser usadas en otros problemas.

Como en HAMLET, el aprendizaje en HEBL comienza después de que el planificador haya encontrado una solución. Dado que HYBIS devuelve el árbol de búsqueda completo en todos los niveles de la jerarquía, HEBL genera reglas de

control para todos los niveles de abstracción. Aunque también se puede comenzar el proceso de aprendizaje justo antes de un refinamiento jerárquico, cuando ha encontrado un plan para un nivel determinado, en cuyo caso sólo generaría reglas para ese nivel y los anteriores. En esta tesis hemos seguido la primera opción.

6.2.1. Etiquetado del árbol

Cada nodo del árbol de búsqueda tiene un campo etiqueta que puede tomar los siguientes valores:

- *success*, el nodo pertenece al camino solución del problema.
- *failure*, pertenece a un camino de fallo.
- *abandoned*, el planificador comienza a expandir este nodo pero lo abandona debido a la heurística que le recomienda seguir por otro camino.
- *unknown*, el planificador no ha expandido el nodo.

Durante el proceso de planificación, cada vez que se genera un nodo se etiqueta como *unknown*. Si se detecta un fallo, el nodo afectado se etiqueta como *failure*. Cuando se llega a una solución se van etiquetando todos los antecesores del nodo como *success*. Cuando empieza el proceso de aprendizaje, lo primero que se hace es etiquetar el árbol mediante un algoritmo recursivo ascendente, empezando por los nodos más profundos en el árbol y subiendo hasta el nodo raíz:

- Si un nodo tiene al menos un sucesor que sea de éxito, se etiqueta como *success* (esto se hace justo al encontrar la solución).
- Si todos sus sucesores son de fallo él también se etiqueta como *failure*.
- Si está etiquetado como *unknown*, pero tiene al menos un sucesor, se cambia su etiqueta a *abandoned*.
- El resto se consideran *unknown*.

Una vez que se tiene el árbol etiquetado existen dos puntos de decisión; es decir, nodos que son candidatos para que se aprenda una regla de control:

- *Failure-Success*: son nodos que tienen, al menos, dos hijos: uno etiquetado con éxito y otro con fallo.
- *Abandoned-Success*: igual que antes, pero en vez de fallo se trata de un nodo abandonado.

En la figura 6.1 se muestra un ejemplo de árbol de búsqueda, señalando los puntos de decisión de donde se pueden aprender reglas. Los nodos de éxito están rodeados por un círculo y los nodos hojas de fallo tienen una X. Los nodos 23, 41, 42 y 43 se etiquetan como *abandoned* porque no son ni de éxito ni de fallo y tienen al menos un sucesor.

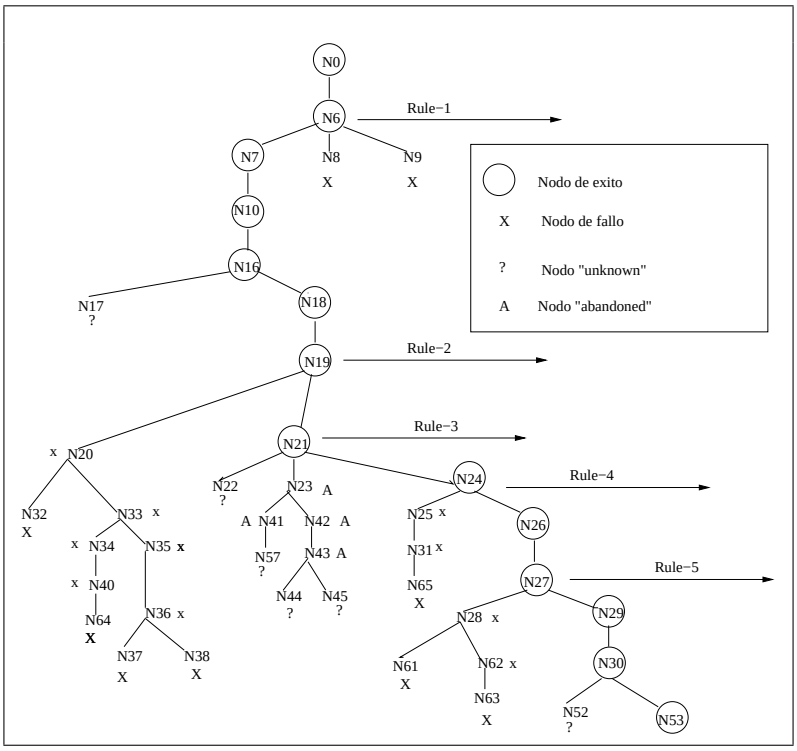

Figura 6.1: Ejemplo abstracto de árbol de búsqueda en HEBL .

Cuando encuentra uno de estos dos puntos de decisión, se genera una regla de control según se explica a continuación.

6.2.2. Generación de reglas de control

En los nodos de decisión donde se ha decidido aprender, se genera una regla de control para que el planificador pueda seleccionar la opción de éxito la próxima vez. De entre los tres tipos de reglas descritos en el capítulo 5: de selección, de rechazo y de preferencia esta tesis considera sólo las de selección.

En un planificador HTN-POCL hay dos tipos diferentes de nodos donde se pueden aprender reglas:

1. Puntos HTN: de qué manera se puede hacer un refinamiento, es decir, qué méto-

do de expansión de los posibles es mejor elegir; y

2. Puntos POCL

   (.1) Para elegir un operador que ya está en el plan o añadir uno nuevo del dominio, cuando se intenta hacer cierta una meta pendiente.

   (.2) Para decidir en ambos casos qué operador seleccionar.

   (.3) Para resolver una amenaza, si usar *promote* o *demote*.

Por tanto, se ha decidido aprender tres tipos de reglas:

- `SELECT operator-plan`: para resolver una meta reutilizar una acción instanciada incluida en el plan parcial.
- `SELECT operator-new`: para resolver una meta seleccionar una nueva acción instanciada del dominio.
- `SELECT expansion`: para refinar una acción seleccionar un método de expansión.

Las reglas constan de tres partes: el nombre de la regla, la parte condicional o requisitos que debe cumplir, y la parte del consecuente o decisión a tomar. La parte condicional se compone de una conjunción de los siguientes meta-predicados:

- `HTN-LEVEL`, su argumento se extrae del nivel en el que está el planificador jerárquico. No tiene sentido aplicar reglas aprendidas para un nivel en otro ya que tanto las acciones como los métodos de expansión dependen del nivel de abstracción del dispositivo al que se refieren.
- `CURRENT-GOAL`, su argumento se extrae de la meta que está tratando de alcanzar el planificador.
- `SOME-CANDIDATE-GOALS`, su argumento son las otras metas que deberán resolverse en ese nodo.
- `OPERATOR-NOT-IN-PLAN`, para comprobar que, en las reglas de tipo `Operator-new`, la acción que se pasa como argumento no esté ya incluida en el plan parcial; o el meta-predicado `OPERATOR-IN-PLAN`, para asegurarse que, en las reglas de tipo `Operator-plan`, realmente se encuentre la acción pasada como argumento. Estos meta-predicados también comprueban que los argumentos del operador verifican las restricciones del agente (esta información se encuentra en la descripción del dominio). Todos los operadores tienen por lo menos un argumento para representar el agente ejecutor de la acción y el resto de argumentos, si existen, deben estar incluidos en las restricciones de dicho agente.

- OPERATOR-TO-EXPAND, para identificar la acción que el planificador está tratando de refinar. Es sólo para las reglas de SELECT expansion. También comprueba que los argumentos del operador cumplan las restricciones del agente.
- COMPONENTS-OF para identificar los agentes del nivel inferior que componen un agente agregado.
- Por último hay un meta-predicado TRUE-IN-STATE por cada literal que es cierto en el estado inicial. Para hacer más eficientes las reglas de control y disminuir los meta-predicados TRUE-IN-STATE se hace una regresión de metas. Es decir, sólo se consideran los literales del estado inicial que sean requeridos, directa o indirectamente, por las precondiciones del operador que añade la regla.

```
(control-rule regla-1
 (IF (AND (HTN-LEVEL 1)
          (CURRENT-GOAL (CONTAINS <I3> <?SRC330>))
          (SOME-CANDIDATE-GOALS ((STATE <LINE-3PROD%%TRANS-3> OFF)
                                 (STATE <STILL%%STILL1-AGG> OFF)
                                 (STATE <STILL%%STILL1-AGG> READY)))
          (OPERATOR-NOT-IN-PLAN (FILTRATE
                                          (<FILTER-AGG%%FILTERAGG-AGG>
                                          <?RESULT>)))
          (TRUE-IN-STATE (STATE <LINE-3PROD%%TRANS-3> OFF))
          (TRUE-IN-STATE (STATE <FILTER-AGG%%FILTERAGG-AGG> OFF))))
 (THEN SELECT OPERATOR-NEW
             (FILTRATE (<FILTER-AGG%%FILTERAGG-AGG> <?RESULT>))))
```

Figura 6.2: Ejemplo regla de control en HEBL .

En la figura 6.2 se muestra un ejemplo de regla de control aprendida por HEBL . Se trata de una regla de selección de la acción filtrate, no incluida en el plan parcial, para conseguir la meta de que el producto representado por la variable <I3> esté contenido en el contenedor representado por la variable <?SRC330>. Debe existir como meta pendiente alguna de las metas representadas en el argumento del meta-predicado SOME-CANDIDATE-GOALS: que el estado de un dispositivo representado por la variable <LINE-3PROD%%TRANS3> sea OFF, o que el estado de un dispositivo representado por la variable <STILL%%STILL1-AGG> sea OFF o READY. En la sección 6.2.3 se explica la sintaxis de estas variables proveniente del proceso de generalización realizado por el algoritmo de aprendizaje. Cada constante del proceso de planificación se transforma en una variable (se incluye entre <>) con el mismo nombre y precedida por un prefijo denotando el tipo de dispositivo. Las variables del proceso de planificación (van precedidas por ?) se transforman directamente a variables del proceso de aprendizaje. Para obtener los literales del estado inicial que son relevantes para poder tomar esta decisión, bien porque son directamente las precondiciones de la acción, o bien porque sean

requeridas por las acciones necesarias para satisfacer las precondiciones de la acción, se realiza una regresión de metas. De esta forma se reduce a dos el número de meta-predicados `TRUE-IN-STATE` de la regla, en vez de los 63 literales que definen el estado inicial de este dominio. Esta regresión de metas se realiza a través del conjunto de enlaces causales de la siguiente manera: la acción (filtrate (filteragg-agg ?result)) aparece como acción consumidora en los siguientes enlaces causales, donde el formato es ((acción-productora) (literal) (acción-consumidora)):

```
((start) (state filteragg-agg off) (filtrate (filteragg-agg i3)))
((on (trans-1 i1 filteragg)) (open-flow i1 tank1 filteragg)
                                        (filtrate (filteragg-agg i3)))
((mix (tank1-agg i1)) (mix i1 tank1) (filtrate (filteragg-agg i3)))
```

De aquí se obtiene el primer literal (state filteragg-agg off) dado que la acción productora de este enlace causal es la acción start. Para el segundo enlace causal se hace la regresión de metas con la acción-productora, es decir (on (trans-1 i1 filteragg)) de la misma manera y se obtiene un nuevo literal (state trans-1 off) y un operador nuevo para hacer la regresión de metas (mix (tank1-agg i1)), pero que ya estaba incluido por ser la acción productora del tercer enlace causal. Procediendo de la misma manera con todos los enlaces causales y acciones productoras, al final se obtienen los literales:

```
(contains r1 s1)      (contains r2 s2)    (contains r3 s3) (state trans-3 off)
(state tank1-agg off) (state trans-1 off) (state filteragg-agg off)
```

Sólo se incluyen en la regla aquellos literales en los que al menos uno de sus argumentos aparece en el resto de meta-predicados. En este ejemplo, son los dos literales (state trans-3 off) y (state filteragg-agg off) que constituyen los dos meta-predicados `TRUE-IN-STATE` de la regla. Los argumentos `r1, s1, r2, s2, r3` y `tank1-agg` no aparecen en ningún otro meta-predicado de la regla y por eso se descartan el resto de literales.

Un método de expansión consiste en un conjunto de metas que deben ser adquiridas por los agentes del siguiente nivel de abstracción. En algunas ocasiones, estas metas representan simplemente estados que deben ser alcanzados por los agentes agregados del agente cuya acción se está tratando de refinar, en cuyo caso es preciso utilizar el meta-predicado `COMPONENTS-OF`. Esto nos lleva a tener dos tipos de reglas; unas cuando el método de expansión consiste en un conjunto de literales del tipo `(STATE <AGENT> state)` y otras para los demás casos.

Resumiendo, se definen cuatro tipos reglas de control diferentes, dos relacionadas con los puntos de decisión POP y otras dos con los HTN. Cada una sigue una plantilla específica:

1. Nodos (o decisiones) POP de selección de un nuevo operador del dominio para resolver una meta:

```
(control-rule name1
   (IF (AND (HTN-LEVEL (level))
            (CURRENT-GOAL goal)
            (SOME-CANDIDATE-GOALS (list-of-subgoals))
            (OPERATOR-NOT-IN-PLAN (operator arguments))
            (TRUE-IN-STATE literal1)
            ....
            (TRUE-IN-STATE literaln)))
   (THEN SELECT OPERATOR-NEW (operator arguments)))
```

La figura 6.3 muestra un ejemplo de una regla para seleccionar una nueva acción no incluida en el plan parcial, (ONLINE ADDITIVE-TRANSPORT1), cuando el planificador está en el nivel 1, trabajando en la meta, (CONTAINS FLOUR RTANK), teniendo pendiente por resolver al menos una de las metas que aparecen en los argumentos del meta-predicado SOME-CANDIDATE-GOALS, y siendo ciertos en el estado inicial los literales (CONTAINS FLOUR ADDITIVE1) y (STATE ADDITIVE-TRANSPORT1 OFF).

```
(control-rule rule-pop-1
 (IF (AND (HTN-LEVEL (1))
          (CURRENT-GOAL (CONTAINS FLOUR RTANK))
          (SOME-CANDIDATE-GOALS ((STATE ADDITIVE-UNIT OFF)
                                 (CONTAINS FERMENT REACTOR)
                                 (STATE  ADDITIVE-UNIT OPEN)
                                 (STATE  REACTOR READY)
                                 (BOTTLED MIXTURE)))
          (OPERATOR-NOT-IN-PLAN (ONLINE  ADDITIVE-TRANSPORT1))
          (TRUE-IN-STATE (CONTAINS FLOUR ADDITIVE1))
          (TRUE-IN-STATE (STATE ADDITIVE-TRANSPORT1 OFF))))
 (THEN SELECT OPERATOR-NEW
          (ONLINE ADDITIVE-TRANSPORT1)))
```

Figura 6.3: Ejemplo de regla de control en HEBL de selección de un operador nuevo del dominio.

2. Nodos POP de selección de un operador ya existente en el plan parcial para resolver una meta:

```
(control-rule name2
   (IF (AND (HTN-LEVEL (level))
            (CURRENT-GOAL goal)
            (SOME-CANDIDATE-GOALS (list-of-sugoals))
            (OPERATOR-IN-PLAN (operator arguments))
            (TRUE-IN-STATE literal1)
            ....
            (TRUE-IN-STATE literaln)))
   (THEN SELECT OPERATOR-PLAN (operator arguments)))
```

La figura 6.4 muestra un ejemplo de este tipo de regla que reutiliza la acción START, cuando está en el nivel 1, trabajando en la meta (STATE REACTOR READY),

teniendo pendiente de resolver alguna de las metas que aparecen en el meta-predicado SOME-CANDIDATE-GOALS, y siendo cierto en el estado inicial el literal (STATE REACTOR READY).

```
(control-rule rule-pop-2
 (IF (AND (HTN-LEVEL (1))
          (CURRENT-GOAL (STATE REACTOR READY))
          (SOME-CANDIDATE-GOALS ((ADDED MILK ENZYME)
                                 (CONTAINS MIXTURE  REACTOR)
                                 (BOTTLED MIXTURE)))
          (OPERATOR-NOT-IN-PLAN (START))
          (TRUE-IN-STATE (STATE REACTOR READY))))
 (THEN SELECT OPERATOR-PLAN (START)))
```

Figura 6.4: Ejemplo de regla de control en HEBL de selección de un operador ya presente en el plan parcial.

3. Nodos HTN para seleccionar un método de expansión compuesto por un conjunto de literales que representan el estado de agentes del nivel inferior:

```
(control-rule name3
   (IF (AND (HTN-LEVEL (level))
            (OPERATOR-TO-EXPAND (operator arguments))
            (TRUE-IN-STATE literal1)
            ....
            (TRUE-IN-STATE literaln)
            (COMPONENTS-OF (agent-agg (agent1 ... agentr)))))
   (THEN SELECT EXPANSION ((STATE agent1 ESTADO1)
                             ...
                           (STATE agentm ESTADOm))))
```

La figura 6.5 muestra un ejemplo de regla que utiliza el método de expansión ((STATE HEAT1 OFF) (STATE ST1 OFF) (STATE SV OFF)), cuando el planificador está haciendo el refinamiento jerárquico de la acción (OFF REACTOR ?RESULT171), los agentes que componen el agente REACTOR son (HEAT1 ST1 SV) y en el estado inicial son ciertos los literales que aparecen como argumentos de los meta-predicados TRUE-IN-STATE. Esta regla obligaría al planificador a refinar la acción (OFF REACTOR ?RESULT171) en ((STATE HEAT1 OFF) (STATE ST1 OFF) (STATE SV OFF)) del siguiente nivel de abstracción.

4. Nodos HTN para seleccionar un método de expansión diferente al anterior; es decir, que no esté constituido por literales que representan el estado de agentes del nivel inferior y por tanto, no es necesario identificar los agentes del nivel inferior que componen el agente agregado.

```
(control-rule rule-htn-1
 (IF (AND (HTN-LEVEL (2))
          (OPERATOR-TO-EXPAND (OFF REACTOR ?RESULT171))
          (TRUE-IN-STATE (STATE SV OFF))
          (TRUE-IN-STATE (STATE ST1 OFF))
          (TRUE-IN-STATE (STATE HEAT1 OFF))
          (COMPONENTS-OF (REACTOR
                          (HEAT1 ST1 SV)))))
 (THEN  SELECT EXPANSION
          ((STATE HEAT1 OFF) (STATE ST1 OFF) (STATE SV OFF))))
```

Figura 6.5: Ejemplo de regla de control en HEBL de selección de un método de expansión consistente en un conjunto de estados de los agentes agregados.

```
(control-rule name4
   (IF (AND (HTN-LEVEL (level))
            (OPERATOR-TO-EXPAND (operator arguments))
            (TRUE-IN-STATE literal1)
            ....
            (TRUE-IN-STATE literaln)))
   (THEN SELECT EXPANSION (expansion)))
```

La figura 6.6 muestra un ejemplo de control para seleccionar el método de expansión (CONTAINS FERMENT REACTOR), cuando el planificador está tratando de refinar la acción (REACT REACTOR FERMENT) y en el estado inicial es cierto el literal (STATE REACTOR READY).

```
(control-rule rule-htn-2
 (IF (AND (HTN-LEVEL (2))
          (OPERATOR-TO-EXPAND (REACT  REACTOR FERMENT))
          (TRUE-IN-STATE (STATE REACTOR READY))))
 (THEN SELECT  EXPANSION
          ((CONTAINS FERMENT REACTOR))))
```

Figura 6.6: Ejemplo de regla de control en HEBL de selección de un método de expansión genérico.

6.2.3. Generalización

Cuando se generan las reglas, los argumentos de los meta-predicados son o bien constantes, porque representan objetos particulares del dominio o bien variables de planificación que cambian en cada ejecución. Para evitar que las reglas de control dependan de los nombres particulares usados cuando se aprendió es preciso generalizarlas convirtiéndolas en variables dentro de las reglas, proceso denominado parametrización. Cada variable se puede emparejar sólo con objetos del mismo

tipo. Por eso cada variable se nombra con un prefijo compuesto por el tipo y los caracteres % %. Por ejemplo, <still % %still1-agg> representa una variable que es del tipo *still*. No todas las constantes se pueden parametrizar; en algunos casos no tiene sentido. Por ejemplo, si tenemos el literal (state trans-3 off), *trans-3* es un buen candidato para ser parametrizado, pero *off* no, ya que el significado de que el transporte está apagado se perdería. Por tanto, actualmente, no se generaliza el segundo argumento de los predicados *state*, porque representa el estado concreto en que se encuentra el agente. Este esquema de generalización es independiente del dominio aunque sea dependiente del planificador.[1]

La regla del ejemplo 1 de la figura 6.3 se generaliza como se muestra en la figura 6.7.

```
(control-rule rule-1
 (IF (AND (HTN-LEVEL (1))
          (CURRENT-GOAL (CONTAINS <FLOUR> <RTANK>))
          (SOME-CANDIDATE-GOALS ((STATE <NOZZLELINE-SIMPLE%%ADDITIVE-UNIT> OFF)
                                 (CONTAINS <FERMENT> <REACTOR-MIXTURE%%REACTOR>)
                                 (STATE  <NOZZLELINE-SIMPLE%%ADDITIVE-UNIT> OPEN)
                                 (STATE  <REACTOR-MIXTURE%%REACTOR> READY)
                                 (BOTTLED <MIXTURE>)))
          (OPERATOR-NOT-IN-PLAN (ONLINE  <TRANSPORTLINE%%ADDITIVE-TRANSPORT1>))
          (TRUE-IN-STATE (CONTAINS <FLOUR> <ADDITIVE1>))
          (TRUE-IN-STATE (STATE <TRANSPORTLINE%%ADDITIVE-TRANSPORT1> OFF))))
 (THEN SELECT OPERATOR-NEW
          (ONLINE <TRANSPORTLINE%%ADDITIVE-TRANSPORT1>)))
```

Figura 6.7: Generalización de una regla en HEBL .

6.3. Equiparación y ejecución de reglas

Cuando el proceso de aprendizaje termina, se crea un fichero con un conjunto de reglas de control que son usadas en la resolución de problemas futuros para podar el árbol de búsqueda. Como se vio en el apartado 4.4, cada vez que el planificador ha de resolver una tarea, obtiene una lista con las posibles formas de resolverla (los *choices* del algoritmo de planificación). Si utiliza reglas de control, genera otra lista paralela con el mismo formato pero a partir de las reglas de control cuyas precondiciones son ciertas en ese momento. La intersección de ambas listas pasa a ser la lista de *choices* y el algoritmo de planificación continúa. En caso de que dicha intersección sea vacía, se queda con los *choices* originales. Para verificar si una regla se cumple, se realiza un proceso de equiparación clásico explicado en la sección 5.2. La única diferencia es que ahora se comprueba meta-predicado a meta-predicado y se genera una lista con las posibles asignaciones de las variables

[1] Salvo en el caso de los literales que representan el estado de un dispositivo, como `(state trans-3 off)`, en que la generalización sí depende del dominio.

que contengan las reglas a constantes del proceso actual de planificación, siempre que sean del mismo tipo. En PRODIGY no es necesaria esta comprobación de tipos porque se utilizan los meta-predicados `type-of-object`. En los dominios de manufacturación tratados, los nombres de los operadores (las acciones de los agentes) se repiten en los distintos agentes, lo que hace que haya más de una acción cuyos argumentos satisfagan la lista de asignaciones posibles obtenida al comprobar la parte izquierda de la regla. Por tanto, por cada regla que se verifica puede haber más de una acción posible y hay que considerarlos todos.

```
(control-rule regla-29
 (IF (AND (HTN-LEVEL 3 )
          (CURRENT-GOAL (CONTAINS <R1> <TANK%%TANK1>))
          (SOME-CANDIDATE-GOALS  (CONTAINS <R2> <TANK%%TANK1>)
                                 (CONTAINS <R3> <TANK%%TANK1>)
                                 (CONTAINS <I1> <TANK%%TANK1>)))
          (OPERATOR-NOT-IN-PLAN (OPEN <VALVE-FWD%%V15> <?PROD> <?SRC> <?DST>))
          (TRUE-IN-STATE (STETE <VALVE-FWD%%V15> OFF))
          (TRUE-IN-STATE (CONTAINS <R1> <S1>))
          (TRUE-IN-STATE (CONTIANS <R3> <S3>))
          (TRUE-IN-STATE (CONTAINS <R2> <S2>)))
 (THEN SELECT  OPERATOR-NEW (OPEN <VALVE-FWD%%V15> <?PROD> <?SRC> <?DST>)))
```

Figura 6.8: Ejemplo de regla en HEBL para explicar proceso de equiparación de reglas.

Por ejemplo, si se tiene la regla representada en la figura 6.8, al comprobar la parte izquierda de la regla saldrán varios valores posibles que pueden tomar las variables <`valve-fwd%%v15`>, <`?prod`>, <`?src`> y <`?dst`>. Por ejemplo, <`valve-fwd%%v15`> podría ser cualquier válvula de tipo `valve-fwd` y en este dominio hay 12, cada una con su correspondiente acción `OPEN`, con lo cual podría haber hasta 12 operadores distintos a aplicar por esta regla (seguramente sean menos porque los otros meta-predicados habrán reducido el número).

6.4. Experimentos y resultados

La experimentación se ha dividido en dos fases. En la primera, simplemente se quiere comprobar la validez del método de aprendizaje propuesto: si las reglas de control aprendidas son correctas y consiguen un aumento de la eficacia del proceso de planificar al disminuir los recursos utilizados en cada ejecución. Si las precondiciones de la reglas son demasiado generales, se pueden disparar en un lugar incorrecto, haciendo que el planificador elija el operador erróneo. Al ser reglas de selección, el resto de alternativas se descartan, con lo cual se pierde la posibilidad de explorar el nodo con la acción adecuada y el sistema nunca llega a la solución. Dada las características de los dominios tratados, en que hay muchos agentes de un determinado tipo y todos con las mismas acciones, se hace especialmente relevante esta cuestión. Por ejemplo, en el dominio ITOPS hay 10 válvulas de tipo

valve-fwd, todas con dos acciones *OPEN* y *SHUT*. Semánticamente es muy distinto abrir o cerrar una válvula u otra pero sintácticamente es complicado distinguir estas situaciones. Por otro lado, si las precondiciones son demasiado específicas sólo se dispararían en un conjunto muy reducido de problemas y llevarían a un problema de utilidad.

En la segunda fase de experimentación se quiere comprobar que las reglas son capaces de generalizar y mejoran la eficiencia en la resolución de nuevos problemas en dominios similares a los que se realizó el aprendizaje. Es decir, plantas industriales con un número diferente de agentes del mismo tipo a los de la planta original. En HYBIS la definición de los dominios y problemas de planificación difiere del resto de planificadores porque cada planta industrial constituye un dominio pero representa un único problema a resolver. En planificación estándar se define un dominio, representando el conjunto de acciones que se pueden ejecutar, y un número de problemas indeterminado que para solucionarlos requieren utilizar las acciones del dominio.

En ambos casos los experimentos consisten en definir manualmente dominios y problemas que representen una determinada planta industrial. Dichos problemas se proporcionan al sistema para que los resuelva en un tiempo máximo fijado.

Las variables independientes de la experimentación son:

- dominio de planificación,
- problemas: variando la complejidad, medida por ejemplo, por el número de metas a satisfacer, número de agentes en el dominio . . . ,
- tiempo límite de ejecución para encontrar una solución.

y las variables dependientes son:

- tiempo de CPU que se tarda en encontrar la solución del problema,
- número de nodos expandidos en el árbol de búsqueda,
- porcentaje de tiempo de CPU ahorrado con las reglas de control.

El objetivo del sistema de aprendizaje es mejorar la eficiencia del planificador; es decir, disminuir los recursos consumidos (tiempo y memoria) utilizados al resolver los problemas. Para evaluar el sistema se escogen dos conjuntos diferentes de problemas, el de entrenamiento y el de validación. Se ejecuta el sistema de aprendizaje con los problemas de entrenamiento para obtener el conocimiento de control. Los problemas de validación se introducen en el planificador dos veces, primero sin utilizar ningún conocimiento y la segunda vez utilizando el conocimiento aprendido. Se computan las variables dependientes definidas antes y se comparan los resultados. Este mismo proceso se realiza con diferentes valores de las variables independientes anteriormente definidas.

6.4.1. Validez de las reglas

La manera más inmediata y simple de comprobar la validez del método de aprendizaje es utilizando el mismo conjunto, compuesto por un único problema, para entrenar y para validar.

Se han hecho varios experimentos utilizando diferentes dominios con características distintas. En la tabla 6.1 se muestran las características de los dominios y problemas utilizados, como son el número de agentes definidos (Agentes), el número de niveles de abstracción (Niveles), el número de operadores o acciones (Operadores) que hay entre todos los agentes, el número de literales del estado inicial (Inicial) y el número de metas que tiene el problema resuelto (Metas).

Dominio	AGENTES	NIVELES	OPERADORES	INICIAL	METAS
ITOPS03	42	3	92	63	1
BC-2	19	2	44	27	5
PAPILLA	26	2	61	46	3
MANIPULADOR	15	3	46	26	1
PLANTA-3	10	2	20	16	2

Tabla 6.1: Características de los dominios utilizados en los experimentos de HEBL .

En la tabla 6.2 se muestran los resultados al resolver un problema sin utilizar reglas y al resolver ese mismo problema usando las reglas aprendidas en una ejecución previa (del mismo problema). Se representa: el número de nodos generados por el proceso de planificación hasta encontrar la solución (N), los segundos tardados hasta llegar a la solución (T), el porcentaje de ahorro en nodos (Ahorro N) y en tiempo (Ahorro T) al utilizar las reglas y el número total de reglas utilizadas en todos los niveles de abstracción (Reglas).

	Sin reglas		**Con reglas**				
Dominio	N	T(S)	N	T(S)	AHORRO N %	AHORRO T %	REGLAS
ITOPS03	898	244	639	212	29	13	59
BC-2	637	60	319	34	50	43	36
PAPILLA	583	62	326	40	44	35	40
MANIPULADOR	235	21	185	15	21	29	33
PLANTA-3	198	9	125	4	37	55	14

Tabla 6.2: Resultados de HEBL para la validación de las reglas que genera.

En todos se observa que disminuye tanto el número de nodos generados como el tiempo que tarda en llegar a la solución, aunque las diferencias no sean muy grandes, debido a la naturaleza de los problemas. Tal como se muestra en la tabla 6.1 son problemas sencillos entre 1 y 5 metas.

6.4.2. Generalización de las reglas en nuevos problemas

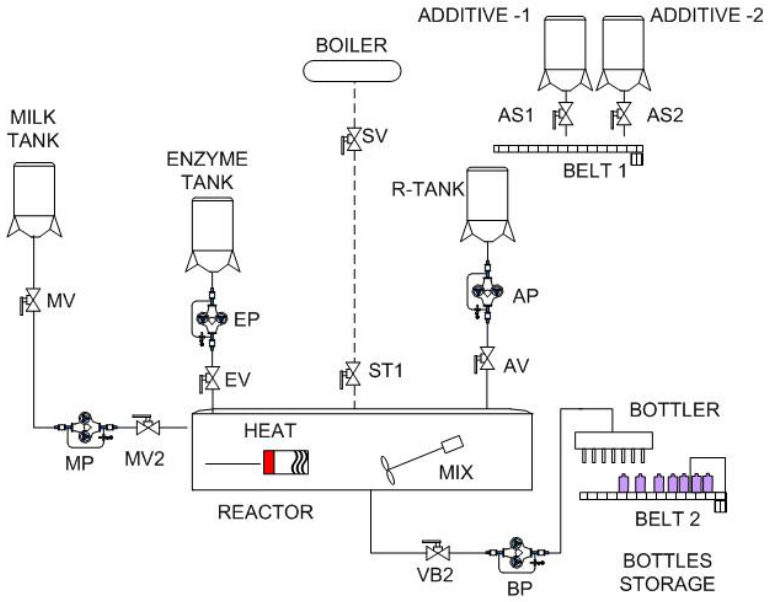

Figura 6.9: Sistema de manufacturación real, PAPILLA, que produce una mezcla elaborada a partir de leche, encimas y harina.

Para comprobar la capacidad de generalización del algoritmo hemos realizado experimentos en el domino Papilla de la figura 6.9 que representa un dominio de manufacturación real de la industria láctea.[2] Se compone de 26 agentes de distintos tipos interconectados entre sí, (válvulas, bombas, mezcladores, calentadores, cintas transportadoras y embotelladoras), 46 acciones diferentes y 2 niveles de abstracción. El objetivo de este dominio es añadir un ingrediente (harina), inicialmente en `ADDITIVE-1`, a la leche contenida en `MILK-TANK` y embotellar la mezcla. La leche y la encima se transforman en fermento en `REACTOR`. El resultado se mezcla con la harina y produce el producto final. El diseño de un programa de control para este sistema es difícil incluso para un ingeniero debido a la cantidad de agentes y al número posible de acciones que se pueden ejecutar.

Para definir los conjuntos de entrenamiento y validación definimos los siguientes dominios, a partir de éste, añadiendo nuevos agentes del mismo tipo. Tal como se dijo antes, a diferencia de planificación estándar, cada dominio en HYBIS sirve para resolver un único problema.

- PAPILLA_2B: se añade una segunda embotelladora y la primera se asume que no está operable (se quita del estado inicial una de las precondiciones que activan la embotelladora). El planificador no detecta su no funcionamiento

[2] La compañía es PULEVA y el proyecto fue el TIC2001-4936-E del Ministerio de Ciencia y Tecnología.

hasta muy tarde en el árbol de búsqueda necesitando más tiempo para encontrar la solución. Si las dos embotelladoras están activas el planificador encuentra la misma solución que el problema original y no se comprueba la capacidad de generalización del algoritmo.

- PAPILLA_3B: se añaden 2 embotelladoras más, estando activa sólo la tercera.
- PAPILLA_4B: se añaden 3 embotelladoras más, estando activa sólo la cuarta.
- PAPILLA_2B2R: se añaden una embotelladora y un reactor. El problema consiste en duplicar la producción de la mezcla final, una en cada reactor, y embotellar cada una en una embotelladora.

Como conjunto de entrenamiento se utilizan los problemas PAPILLA y PAPILLA_2B. De forma independiente se extrae el conocimiento de cada problema. En ambos casos se generaron 43 reglas de control. Del resto de problemas no es posible extraer conocimiento debido a problemas de memoria del planificador. El método de aprendizaje requiere almacenar todo el árbol de búsqueda para explorarlo, una vez se encuentra la solución, identificando los puntos de decisión. El planificador tiene una opción que permite la posibilidad de almacenarlo o no. En el resto de problemas, al ser más complejos y necesitar más nodos, no fue posible activar dicha opción. Además, suele ser más conveniente hacer aprendizaje sobre problemas pequeños porque se obtienen reglas más generales. Las reglas de control se generan obteniendo el meta-estado del episodio actual de planificación y haciendo regresión de metas para determinar los literales del estado inicial. Por tanto, cuanto menor sea el problema de aprendizaje, menor será el meta-estado, menos literales habrá en el estado inicial y las reglas serán más simples y generales.

La tabla 6.3 muestra los resultados con estos dos conjuntos de reglas: reglas_1 aprendidas del problema PAPILLA y reglas_2 aprendidas del problema PAPILLA_2B. Muestra el número de nodos (N) generados por el proceso de planificación, el tiempo en segundos (T) hasta que encuentra la solución y el porcentaje de nodos (AN) y tiempo (AT) ahorrado al utilizar las reglas.

	Sin reglas		**Con reglas_1**				**Con reglas_2**			
Dominio	N	T(s)	N	T(s)	AN	AT	N	T(s)	AN	AT
PAPILLA_2B	2506	630	1463	393	42 %	38 %	1539	429	39 %	32 %
PAPILLA_3B	4458	1350	2604	750	42 %	44 %	2753	862	38 %	36 %
PAPILLA_4B	6410	1952	3745	1082	42 %	45 %	3967	1090	38 %	44 %
PAPILLA_2B2R	4057	1094	808	770	80 %	30 %	842	911	79 %	17 %
PAPILLA_3B2R	6821	1455	1091	842	84 %	42 %	1125	966	83 %	34 %

Tabla 6.3: Resultados de HEBL con nuevos problemas.

Se observa que tanto el número de nodos como el tiempo disminuyen con las reglas de control. La media de tiempo ahorrado es del 36 % y en nodos del 57 %.

También se ha medido el tiempo total de equiparación para medir la utilidad de las reglas, tal como se muestra en la Tabla6.4. La columna TIEMPO representa el tiempo de equiparación de las reglas en segundos y la columna %TOTAL representa el porcentaje respecto al tiempo total de búsqueda. Los resultados muestran que no existe un problema grave de utilidad, empleando en media entre 8 y 10 % más de tiempo, y eso que se utiliza un esquema de equiparación de reglas lenta frente a otros algoritmos más eficientes como el RETE que disminuirían este tiempo.

	Con reglas_1		**Con reglas_2**	
Dominio	TIEMPO	%TOTAL	TIEMPO	%TOTAL
PAPILLA_2B	50	13 %	43	10 %
PAPILLA_3B	63	8 %	80	9 %
PAPILLA_4B	85	8 %	84	8 %
PAPILLA_2B2R	44	6 %	68	7 %
PAPILLA_3B2R	55	6 %	37	4 %

Tabla 6.4: Tiempo total de equiparación de las reglas en HEBL .

Los resultados utilizando el primer conjunto de reglas son mejores que los obtenidos utilizando el segundo dominio. Esto demuestra la conveniencia de entrenar con problemas simples para aplicar el conocimiento a problemas más complejos.

6.5. Conclusiones y trabajos futuros

En los planificadores HTN los planes se construyen en distintos niveles de jerarquía, empezando por el nivel más alto de abstracción y refinando poco a poco hasta niveles más específicos. El ingeniero puede definir métodos para describir cómo realizar el refinamiento de un nivel a otro. Por ejemplo, HYBIS es un planificador jerárquico que además tiene definido un método por omisión para pasar de un nivel a otro inferior. El refinamiento jerárquico implica resolver un nuevo problema de la misma forma en que lo hace un POP. Sin embargo, aunque el uso de la jerarquía simplifica la complejidad computacional, el proceso sigue siendo aún ineficiente. En un planificador híbrido como HYBIS se puede incrementar la eficiencia tanto en los puntos de decisión propios de los HTN como en los de POP.

En este trabajo se han discutido varios aspectos de aprendizaje automático aplicado a este tipo de planificadores y se han extendido algunas de las idea de aprendizaje automático para poder aplicarlas a un planificador híbrido HTN-POP. En particular, se han analizado los puntos de decisión POP en los que el planificador decide si aplicar un nuevo operador ya utilizado en el plan parcial o utiliza uno nuevo y en ambos casos de qué operador concreto se trata. Se han generado reglas de control que ayuden a tomar estas decisiones basándose en las decisiones tomadas por el planificador en resoluciones de problemas que llevaron a la solución. También se ha estudiado el refinamiento jerárquico. Cada tarea se puede descom-

poner en varias subtareas del nivel inferior utilizando un método predefinido. Cada posible descomposición representa una nueva rama en el árbol de búsqueda y se pueden aprender reglas de control para podar las ramas que no llevan a la solución. Se ha evaluado el sistema de aprendizaje propuesto generando todas las reglas de control anteriores y comprobado que incrementan la eficiencia del planificador, disminuyen el tiempo y número de nodos necesarios para encontrar la solución. También se ha verificado la conveniencia de adquirir el conocimiento de control de problemas simples para aplicarlo a problemas más complejos.

El concepto de dominio en HYBIS difiere del de otros planificadores. Para HYBIS un dominio es una planta industrial, diseñada específicamente para resolver un único problema; es decir, para generar un producto manufacturado concreto a partir de unas materias primas iniciales. Lo más usual es que se encuentre un único plan que realice esta transformación y el planificador no necesite volver a ejecutarse. Sin embargo, puede ocurrir que en una misma planta industrial se requiera hacer una pequeña modificación del diseño, con lo que se tendría que volver a encontrar el plan adecuado, con todas las dificultades que ello conlleva. Aunque la modificación sea mínima, el proceso de planificación tiene que empezar desde cero otra vez. Si se tienen reglas de control aprendidas en la primera ejecución, este segundo problema de planificación se resolverá con mayor eficiencia. En caso de tener un problema que el planificador no pudiese resolver se podrían definir relajaciones de dicho problema y entrenar el sistema con ellos. Las reglas de control obtenidas ayudarían a encontrar la solución al problema original.

En un futuro, se podría extender el esquema de aprendizaje para mejorar las reglas de control obtenidas especializándolas, generalizándolas o modificándolas por medio de métodos inductivos tal como se ha hecho con otros planificadores (HAMLET [Borrajo and Veloso, 1997] y EVOCK [Aler *et al.*, 2002]). También, HYBIS se ha extendido para generar planes condicionales que ofrece nuevas oportunidades de aprendizaje. Por último, realizar un análisis de la aportación de las reglas HTN y de las POP para estudiar la posibilidad de aplicarlo a planificadores similares que utilicen el mismo paradigma.

Capítulo 7

Sistema de aprendizaje GEBL

En este capítulo se presenta un sistema de aprendizaje de conocimiento de control aplicable a planificación basada en grafos de planes, desarrollado en la tesis doctoral. Se ha denominado GEBL, de Graphplan EBL. El planificador utilizado es TGP descrito en la sección 4.6. El método de aprendizaje propuesto está basado en HAMLET; en cuanto, al lenguaje de representación del conocimiento de control y la técnica de aprendizaje deductiva, EBL, utilizada para dar una explicación de las decisiones correctas tomadas durante el proceso de búsqueda del plan solución. También permite hacer un aprendizaje *eager* o *lazy* dependiendo de si aprende de todos los puntos de decisión posibles o sólo de aquéllos en los que la decisión correcta no fue la tomada inicialmente por el planificador. Se verá la conveniencia de utilizar uno u otro según el fin que se le vaya a dar al conocimiento de control generado.

7.1. Descripción

Los planificadores basados en grafos de planes, como se explica en la sección 4.5, alternan dos fases: la expansión del grafo del plan y la búsqueda hacia atrás del plan solución en el grafo generado. La primera fase se realiza en tiempo polinómico, generando todos los predicados y acciones instanciadas que conforman el grafo del plan. En la segunda fase es donde reside la complejidad computacional al tratarse de un algoritmo de búsqueda hacia atrás sin heurística.[1] Aunque el espacio de búsqueda se haya simplificado de lógica preposicional a lógica proposicional, sigue siendo un proceso costoso en el que es preciso retroceder en muchos puntos porque las decisiones tomadas llevan a puntos muertos. La utilización de los *memoizations* o *nogoods* del algoritmo incrementa la eficiencia de resolución pero no evita todos los retrocesos. Es en estos casos donde las técnicas de apren-

[1] TGP utiliza heurísticas débiles importadas de los CSP: ordenación de metas, de mayor a menor dificultad; y ordenación de operadores que alcanzan una meta, de menor a mayor coste.

dizaje automático tienen cabida para intentar mejorar la eficacia del planificador, podando las ramas erróneas del árbol de búsqueda que conducen a nodos de fallo.

El sistema GEBL extrae conocimiento de control, en forma de reglas, de un planificador basado en grafos de planes utilizando técnicas de aprendizaje deductivo, en particular EBL, durante la fase de búsqueda del plan solución en el grafo del plan. Analiza el árbol de búsqueda generado por el planificador al resolver los problemas de entrenamiento, estudiando las decisiones tomadas que llevaron a la solución y las que llevaron a nodos de fallo, para dar una explicación de las circunstancias que las propiciaron y generar las reglas de control.

El método es general e independiente del problema que se esté solucionando. Genera conocimiento de control aplicable a cualquier otro problema del mismo dominio. Existen varios trabajos que han incorporado alguna técnica de aprendizaje para aumentar la eficiencia de GRAPHPLAN, como se expone en la sección 4.5. Uno de los más notables es el de Zimmerman y Kambhampati
[Zimmerman and Kambhampati, 2005]. Sin embargo, todas las propuestas de estos trabajos se centran en aprender sólo para el problema actual de resolución sin tratar la generalización a problemas diferentes. Por tanto, el sistema aquí propuesto pretende complementar estas preguntas.

Antes de detallar el sistema de aprendizaje se explica el árbol de búsqueda que se puede derivar del algoritmo de planificación realizado por TGP.

7.2. Árbol de búsqueda en TGP

El algoritmo de extracción de la solución realizado por TGP es un algoritmo de búsqueda hacia atrás sobre el grafo del plan. La búsqueda comienza por las metas, últimos niveles del grafo, y va progresando hasta el estado inicial, nivel 0 del grafo. Cada estado o nodo del árbol puede describirse por tres elementos:

- El nivel del grafo del plan, i, o instante de tiempo donde se hace la búsqueda.[2]
- El conjunto de metas, G, al que hay que buscar una asignación válida de acciones que las consigan.
- El plan parcial o acciones, A, que hasta ese punto conforman la solución actualmente construida.

El primer nodo, $N(i, G, \varnothing)$, se genera con las metas, G, del problema, el nivel, i, donde empieza la fase de extracción de la solución y como plan parcial el

[2]Para realizar el tratamiento temporal, TGP transforma los niveles del grafo del plan para que representen instantes de tiempo. Nivel del grafo y tiempo son equivalentes en TGP.

conjunto vacío. El algoritmo de planificación en el nivel i busca una asignación válida de las metas $g_i \in G$: para cada meta g_i puede, o bien, buscar una acción instanciada que la tenga entre sus efectos, o bien, persistirla (en ambos casos, si no violan ninguna relación de mutex). En este último caso, se pospone la asignación de la acción a los siguientes niveles de búsqueda, más próximos al estado inicial. Es equivalente a escoger la acción `no-op` para dicha meta cuya única precondición (y efecto) es la propia meta. Cuando encuentra una asignación válida de las metas, se genera un nuevo nodo hijo en el nivel inferior, $N(i-1, G', A')$, donde G' son las precondiciones de las nuevas acciones elegidas para satisfacer las metas de G (incluida las acciones `no-op`, cuya única precondición es la propia meta) y A' son las nuevas acciones elegidas para satisfacer las metas (sin incluir las `no-op`). Pueden existir tantos nodos hijos como asignaciones válidas distintas haya de las metas G. El resto de nodos del árbol se generan de la misma forma durante la búsqueda de asignaciones válidas de las metas G' en los siguientes niveles del grafo del plan. Las acciones nuevas encontradas en cada nivel se añaden al principio del plan parcial, de forma que el conjunto A' de un nodo hijo es igual al conjunto A del nodo padre más las nuevas acciones encontradas por el algoritmo de planificación en ese nivel.

La figura 7.1 muestra parte de un posible árbol de búsqueda generado por TGP. Representa un episodio de búsqueda fallido empezando en el nivel 6 y los primeros cuatro nodos del segundo episodio de éxito, a partir del nivel 7 (los nodos de fallo se representan con cuadrados de líneas discontinuas y los de éxito con línea continua). El primer episodio corresponde a una búsqueda en la que las metas del problema, `G0`, aparecen por primera vez en el nivel 6, sin relaciones de mutex entre ellas. El primer nodo del árbol se construye con `G0`, nivel 6 y plan parcial (NIL). El segundo nodo, `(5,G1,A1)`, se origina cuando TGP busca una asignación válida de operadores que satisfagan todas las metas de `G0`. Es el conjunto `A1`, y `G1` son las precondiciones de las acciones de `A1` junto con las metas que persisten en el nivel 6. El segundo nodo, `(4,G2,A2)`, se genera de forma similar, al intentar satisfacer las metas `G1` en el nivel 5. El conjunto `A2` contiene las acciones `A1` más las nuevas acciones encontradas para satisfacer todas las metas de `G1`. `G2` son las precondiciones de las nuevas acciones junto con las metas que persisten de `G1`. El algoritmo continúa hasta llegar al nodo `G3A3`. En el nivel 2 del grafo no existe un conjunto de acciones capaz de satisfacer las metas de `G3` sin que sean mutex con las acciones de `A3`. Se retrocede hasta el nodo `G2A2` y se busca otra asignación posible de acciones, generándose el nodo `(3,G4,A4)` que también falla. Se retrocede al nodo `(4,G2,A2)` otra vez, pero al no haber más conjuntos de acciones posibles se retrocede al nodo `(5,G1,A1)` y se prueba con el nodo `(4,G5,A5)` que también falla. El proceso sigue hasta agotar todas las combinaciones y TGP incrementa el grafo del plan con un nuevo nivel de acciones y sus efectos. El segundo episodio de búsqueda comienza. El primer nodo se genera con las metas originales del problema, `G0`, y el plan vacío pero en el nivel 7. Muchos de los nodos generados en el primer episodio de búsqueda para un nivel i se repiten en episodios posteriores en

el nivel $(i+1)$ y, aunque fallaran en la primera búsqueda, en otros episodios pueden conducir hasta la solución al haber más acciones y proposiciones en el grafo que hacen desaparecer relaciones de mutex. El árbol de búsqueda seguiría creciendo de forma similar, a partir del nodo `(6,G12,A12)`, hasta llegar al nivel 0, en que el último nodo generado, su conjunto de acciones A, será la solución del problema (no está representado en la figura).

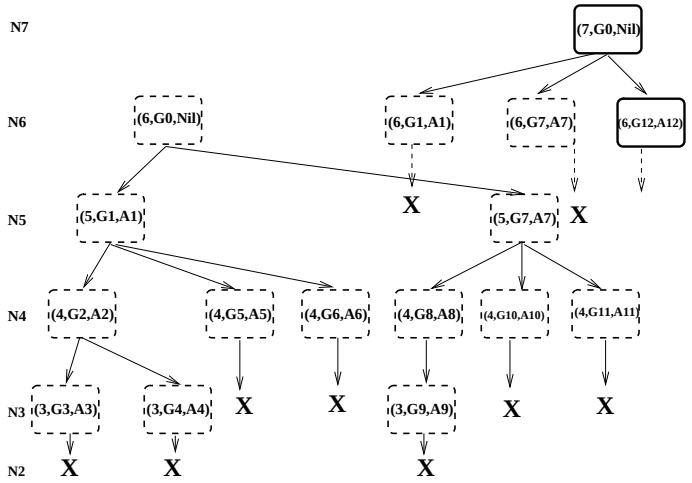

Figura 7.1: Ejemplo abstracto de parte del árbol de búsqueda en TGP.

Este árbol de búsqueda es una abstracción de la búsqueda realizada por TGP. El algoritmo original de TGP no lo construye y es el sistema de aprendizaje el que lo va generando y almacenando. Cada nodo del árbol representa un conjunto de metas a satisfacer, en un nivel del grafo del plan, y sus hijos representan las diferentes maneras de satisfacerlas. A cada nodo se le denominará punto de decisión (PD) y sobre ellos se basa el algoritmo de aprendizaje que se explica en la siguiente sección.

7.3. Proceso de aprendizaje

Analizando el algoritmo de búsqueda hacia atrás que realizan los planificadores basados en grafos de planes del plan solución, se identifican algunos puntos en los que el algoritmo es ineficaz y es preciso retroceder. Es en estos puntos donde las técnicas de aprendizaje automático pueden extraer el conocimiento de control preciso para guiar la búsqueda por las ramas de éxito. Se distinguen los siguientes puntos de aprendizaje:

- En qué nivel del grafo del plan empezar a hacer la búsqueda de la solución. El que estén todas las metas presentes en un nivel y no exista una relación de mutex activa entre ellas es una condición necesaria pero no suficiente para

encontrar un plan válido. Muchas veces el algoritmo comienza la búsqueda en un nivel erróneo, expandiendo el árbol de búsqueda inútilmente hasta comprobar que no hay solución y retrocediendo sucesivamente hasta que no hay más acciones posible con lo que hay que expandir el grafo del plan en un nivel más y comenzar un nuevo episodio de búsqueda.

- Para cada conjunto de metas que se está asignando operadores en un nivel, hay que decidir cuales pueden persistir.

- Por último, en las metas en que no es posible la persistencia, hay que elegir un operador de entre todos los posibles que la soportan.

Aprender el nivel del grafo en el que comenzar la búsqueda es equivalente a determinar, "a priori", la longitud del plan solución. Se considera un problema complejo en aprendizaje aplicado a planificación. Por eso, esta tesis se centra sólo en los otros dos puntos de decisión. El algoritmo de TGP en esos puntos, sigue dos heurísticas (débiles) que no siempre son las más adecuadas:

- Intentar la persistencia primero antes que añadir un nuevo operador.

- De entre los posibles operadores que soportan una meta se intentan antes los que están en un nivel inferior en el grafo del plan.

El método de aprendizaje implementado en GEBL sigue un algoritmo con tres pasos:

1. El planificador resuelve un problema de aprendizaje generándose una traza de todos los puntos de decisión probados durante el proceso de búsqueda del plan solución. Cada PD se etiqueta como de éxito o de fallo según llevaron a la solución o a un punto muerto.

2. En los puntos de decisión de éxito, se generan reglas de control que expliquen el estado de planificación que condujo a tales decisiones.

3. Las constantes en las reglas de control se generalizan para que puedan ser usadas en otros problemas. En GEBL la parametrización o generalización consiste, simplemente, en incluir entre <> las constantes de planificación.

El proceso de aprendizaje comienza después de que el planificador encuentra una solución para un problema. Si no encuentra solución, no se puede extraer ningún conocimiento de control.

7.3.1. Generación y etiquetado de los puntos de decisión

Cada punto de decisión del árbol de búsqueda tiene un campo etiqueta que puede tomar los siguientes valores:

- *success*, el punto pertenece al camino solución del problema. Las acciones representadas en dicho nodo son parte del plan solución.
- *failure*, pertenece a un camino de fallo. El fallo se origina cuando no se encuentra una asignación válida para todas las metas de dicho nodo en el nivel actual, o todos sus nodos hijos fallan.
- *memo-failure*, el planificador no ha expandido el punto de decisión porque ya está almacenado como *memoization* en un episodio de búsqueda previo.

Durante el proceso de planificación, cada vez que el planificador intenta asignar valores a un conjunto de metas en un nivel (instante de tiempo), se crea un nuevo punto de decisión con esas metas, el nivel de búsqueda, las acciones que hasta ese momento tenga el plan parcial y con el valor *memo-failure* de la etiqueta de aprendizaje. Si se encuentra una asignación correcta, dicho punto se etiqueta como *success* y se continúa la búsqueda con el siguiente nivel. Si se produce algún error, se etiqueta como *failure* y se retrocede al nivel anterior. Si ese conjunto de metas, en ese nivel, ya está almacenado como *memo* su etiqueta permanece con el valor *memo-failure* inicial.

Cuando el algoritmo de planificación termina satisfactoriamente, se encuentra una solución al problema, y se recorre la lista de los puntos de decisión anteriores buscando pares de nodos etiquetados como *success* con niveles consecutivos. Por cada pareja encontrada, se llama al módulo generador de reglas de control creándose una regla por cada meta del primer punto de decisión. Según el tipo de aprendizaje establecido, *eager* o *lazy* (es un parámetro de entrada del sistema de aprendizaje), dicha selección se hará de distinta manera. En aprendizaje *lazy* sólo se generan reglas de control si la primera decisión del planificador no es la que conduce a la solución mientras que el aprendizaje *eager* aprende de todas las decisiones. En ambos casos, se comienza seleccionando el primer PD etiquetado como *success*, P1:

1. Si el siguiente PD es de éxito, P2, y el aprendizaje *eager*, llama al módulo de generación de reglas de control con P1 y P2.
2. Si no, busca el siguiente PD de éxito, P3, y llama al módulo de generación de reglas con P1 y P3 (aprendizaje tanto *lazy* como *eager*). Si el aprendizaje es *lazy*, al módulo generador de las reglas, también se le pasa el punto de decisión P2 para identificar las decisiones tomadas por el planificador, como se explicará a continuación.

Posteriormente, hace P1=P2 en el primer caso y P1=P3 en el segundo, y se repite el proceso anterior hasta que no queden más PD en la lista.

Por ejemplo, supóngase que tenemos el problema del dominio Zenotravel representado en la figura 7.2. En el dominio Zenotravel existen ciudades, personas y aviones. Cada persona está inicialmente en una ciudad y la meta es desplazarla a otra distinta. Para ello se deben utilizar los aviones que pueden volar a dos velocidades diferentes consumiendo más o menos combustible, `fly` y `zoom`. Los aviones tienen un nivel de combustible inicial y tras cada vuelo se ve disminuido en uno (acción `fly`) o dos niveles (acción `zoom`). Existe una acción `refuel` para repostar los aviones (ver el apéndice A.2). En el ejemplo, existen 2 ciudades, 2 personas y un avión. Inicialmente las dos personas están en ciudades diferentes y la meta es intercambiarlas de ciudad; es decir, que `person0` que está en `city0` pase a `city1` y que `person1` que está en `city1` pase a `city0`. El avión está inicialmente en la misma ciudad que `person1` y tiene un nivel de combustible suficiente para hacer los dos viajes. La traza con los puntos de decisión y sus etiquetas generadas por TGP al resolver dicho problema se muestran en las figura 7.3. Para

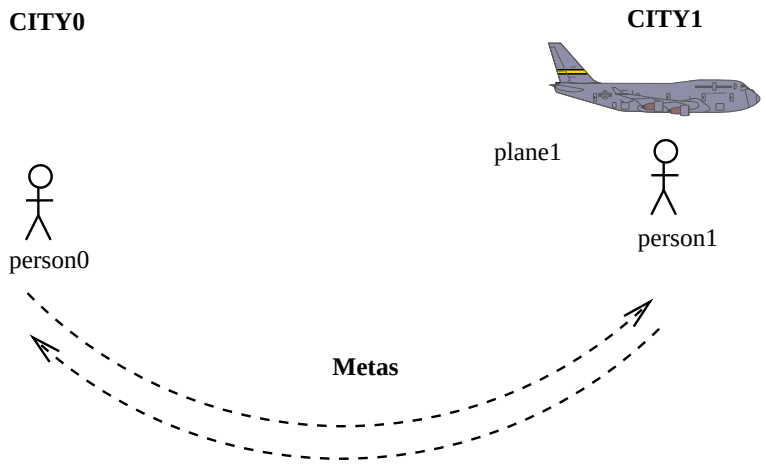

Figura 7.2: Ejemplo de problema en el domino Zenotravel en el que la meta es intercambiar dos personas de las ciudades en las que inicialmente se encuentran.

este problema, TGP expande el grafo del plan hasta el nivel (instante de tiempo) 5 en que aparecen, por primera vez, las dos metas del problema sin ser mutex entre sí. Entonces, empieza un episodio de búsqueda. Se genera el primer punto de decisión con las metas del problema `((at person1 city0) (at person0 city1))` y conjunto de acciones vacío (NIL). El algoritmo de búsqueda empieza en el nivel 5, encontrando la acción `(debark person0 plane1 city1)` para conseguir la meta `(at person0 city1)` y persiste la otra meta `(at person1 city0)`. Esto genera el nodo hijo representado en la línea *Time:(4)* que tiene la meta que ha persistido y las precondiciones del operador seleccionado (`debark`); es decir, `((at plane1 city1) (in person0 plane1))`. La acción `(debark`

```
Time:(5) (SUCCESS)
Goals=((at person1 city0) (at person0 city1))
Actions=NIL

Time:(4) (SUCCESS)
Goals=((at person1 city0) (at plane1 city1) (in person0 plane1))
Actions=(((debark person0 plane1 city1) (at person0 city1)))

Time:(3) (SUCCESS)
Goals=((at person1 city0) (in person0 plane1) (fuel-level plane1 fl1)
       (at plane1 city0))
Actions=(((fly plane1 city0 city1 fl1 fl0) (at plane1 city1))

         ((debark person0 plane1 city1) (at person0 city1)))

Time:(2) (SUCCESS)
Goals=((in person1 plane1) (fuel-level plane1 fl1) (at plane1 city0)
       (at person0 city0))
Actions=(((board person0 plane1 city0) (in person0 plane1))
         ((debark person1 plane1 city0) (at person1 city0))

         ((fly plane1 city0 city1 fl1 fl0) (at plane1 city1))
         ((debark person0 plane1 city1) (at person0 city1)))

Time:(1) (SUCCESS)
Goals=((in person1 plane1) (at person0 city0) (fuel-level plane1 fl2)
       (at plane1 city1))
Actions=(((fly plane1 city1 city0 fl2 fl1) (fuel-level plane1 fl1))

         ((board person0 plane1 city0) (in person0 plane1))
         ((debark person1 plane1 city0) (at person1 city0))
         ((fly plane1 city0 city1 fl1 fl0) (at plane1 city1))
         ((debark person0 plane1 city1) (at person0 city1)))

Time:(0) (SUCCESS)
Goals=((at person1 city1) (at person0 city0) (fuel-level plane1 fl3)
       (at plane1 city1))
Actions=(((board person1 plane1 city1) (in person1 plane1))

         ((fly plane1 city1 city0 fl2 fl1) (fuel-level plane1 fl1))
         ((board person0 plane1 city0) (in person0 plane1))
         ((debark person1 plane1 city0) (at person1 city0))
         ((fly plane1 city0 city1 fl1 fl0) (at plane1 city1))
         ((debark person0 plane1 city1) (at person0 city1)))
```

Figura 7.3: Ejemplo de árbol de búsqueda en el dominio Zenotravel.

`person0 plane1 city1)` se añade al principio de las acciones del nodo hijo. Comienza una nueva búsqueda en el nivel 4, donde se encuentra la acción `(fly plane1 city0 city1 fl1 fl0)` para conseguir la meta `(at plane1 city1)` y se persisten el resto de metas. Esto genera el nodo hijo de la línea *Time:(3)* con las precondiciones de la acción `fly` y las metas que persisten. La nueva acción se añade al principio del plan parcial (conjunto de acciones del nodo hijo). El algoritmo continúa hasta alcanzar el nivel 0 y las acciones del último punto de decisión representan la solución del problema. Todos los puntos de decisión del camino solución se etiquetan como *success*. En este ejemplo, no hay ningún nodo de fallo y por tanto, no hay retroceso. Si el aprendizaje fuese *lazy*, no se generaría ninguna regla de control. Pero, con aprendizaje *eager*, se generan varias reglas por cada par de puntos de decisión de éxito en niveles consecutivos; es decir, del punto de decisión de la línea *Time: (5)* y *Time: (4)*, de *Time: (4)* y *Time: (3)*, de *Time: (3)* y *Time: (2)*, *Time: (2)* y *Time: (1)* y por último, de *Time: (1)* y *Time: (0)*. Al final de la siguiente sección se muestra un ejemplo de aprendizaje *lazy*.

7.3.2. Generación de reglas

El módulo de generación de reglas recibe como entrada dos puntos de decisión de éxito con niveles consecutivos. Corresponde a un estado de planificación en que, dado un conjunto de metas, parte de ellas persisten (se pospone la asignación de operadores a niveles más próximos al estado inicial), y a otras, se le asignan operadores para satisfacerlas. Por cada meta implicada en el primer punto de decisión, se crea una regla de control para seleccionar la opción tomada por el planificador (reflejada en el segundo punto de decisión) que condujo a la solución; es decir, persistir la meta o asignarle un operador.

Por ejemplo, dados los dos primeros puntos de decisión de la figura 7.3, *Time:(5)* y *Time:(4)*, el conocimiento de control que se puede obtener es:

- Elegir el operador `(debark person0 plane1 city1)` para satisfacer la meta `(at person1 city0)`.
- Elegir la persistencia para la meta `(at person1 city0)`.

Para que las reglas sean generales y puedan utilizarse en otros problemas, es necesario dar una explicación del estado de planificación durante la decisión. El conocimiento de control se representa como reglas de control IF-THEN. Las precondiciones representan dicho estado de planificación, mediante meta-predicados y la parte del consecuente representa la acción realizada por el planificador. Dicho estado se obtiene del primer punto de decisión. El segundo punto de decisión se necesita para obtener las acciones realizadas.

GEBL aprende dos tipos de reglas para guiar las búsquedas futuras en TGP:

- `SELECT operator`: selecciona un operador instanciado que resuelva una meta.
- `SELECT operator persits`: selecciona la persistencia para una meta.

La parte condicional de las reglas se compone de una conjunción de los siguientes meta-predicados que determinan el estado de planificación:

- `CURRENT-GOAL`, su argumento es la meta que el planificador está tratando de conseguir.
- `SOME-CANDIDATE-GOALS`, su argumento es el resto de metas en el punto de decisión. Para hacer más eficientes las reglas sólo se consideran aquellas metas que tengan alguna relación de mutex (no activa) con `CURRENT-GOAL`.
- `ALL-SOLVED-ACTIONS`: su argumento es el plan parcial actual. Se consideran sólo las que tengan alguna relación de mutex (no activa) con `CURRENT-GOAL`.

- Por último, hay un meta-predicado TRUE-IN-STATE por cada literal que es cierto en el estado actual de planificación. Para hacer más eficientes las reglas de control y disminuir los meta-predicados TRUE-IN-STATE se hace una regresión de metas. Esto es, sólo se consideran los literales del estado que sean requeridos, directa o indirectamente, por las precondiciones del operador que añade la regla. En la sección 7.3.2 se explica en detalle cómo realizar la regresión de metas en planificadores basados en grafos.

La figura 7.4 muestra la regla de SELECT OPERATOR generada del primer punto de decisión del ejemplo anterior. Esta regla selecciona el operador debark para desplazar una persona de una ciudad a otra. Selecciona el avión más conveniente; es decir, el que está en la misma ciudad que la persona y con suficiente combustible. El argumento del meta-predicado ALL-SOLVED-ACTIONS es NIL porque se trata de la primera decisión y el plan parcial está vacío. El argumento de SOME-CANDIDATE-GOALS es NIL, porque la otra meta del punto de decisión, no es mutex (no activo) con (at <person0><city1>).

```
(control-rule regla-ZENO-TRAVEL-ZENO1-e1
 (IF (AND (CURRENT-GOAL (AT <PERSON0> <CITY1>))
          (TRUE-IN-STATE (AT <PERSON0> <CITY0>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY1>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL3>))
          (TRUE-IN-STATE (AIRCRAFT <PLANE1>))
          (TRUE-IN-STATE (CITY <CITY0>))
          (TRUE-IN-STATE (CITY <CITY1>))
          (TRUE-IN-STATE (FLEVEL <FL1>))
          (TRUE-IN-STATE (FLEVEL <FL2>))
          (TRUE-IN-STATE (FLEVEL <FL3>))
          (TRUE-IN-STATE (NEXT <FL2> <FL3>))
          (TRUE-IN-STATE (NEXT <FL1> <FL2>))
          (TRUE-IN-STATE (PERSON <PERSON0>))
          (SOME-CANDIDATE-GOALS NIL) (ALL-SOLVED-ACTIONS NIL)))
 (THEN SELECT OPERATORS (DEBARK <PERSON0> <PLANE1> <CITY1>))
```

Figura 7.4: Ejemplo de regla de SELECT OPERATOR en el dominio Zenotravel.

La regla de persistir generada en ese mismo punto de decisión se muestra en la figura 7.5. Lógicamente, las precondiciones son las mismas que la regla anterior.

Cuando el aprendizaje es *lazy*, no se invoca al módulo generador de reglas si entre los dos puntos de decisión de éxito no hay, al menos, uno intermedio etiquetado como *failure*. Si los nodos intermedios están etiquetados como *memo-failure* (son *memoization* del algoritmo de planificación) se considera que es equivalente a la opción por omisión ya que no se llegan a explorar esos nodos. Con los dos puntos de decisión de éxito, se genera una regla de control por cada una de las metas del primer punto que seleccione la acción elegida por el planificador (esta información se puede obtener comparando los dos puntos de decisión de éxito). Sin embargo, alguna de estas acciones seleccionadas para resolver una de las metas del punto

```
(control-rule regla-ZENO-TRAVEL-ZENO1-p1
 (IF (AND (CURRENT-GOAL (AT <PERSON1> <CITY0>))
          (TRUE-IN-STATE (AT <PERSON1> <CITY1>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY1>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL3>))
          (TRUE-IN-STATE-STATIC (AIRCRAFT <PLANE1>))
          (TRUE-IN-STATE-STATIC (CITY <CITY0>))
          (TRUE-IN-STATE-STATIC (CITY <CITY1>))
          (TRUE-IN-STATE-STATIC (FLEVEL <FL1>))
          (TRUE-IN-STATE-STATIC (FLEVEL <FL2>))
          (TRUE-IN-STATE-STATIC (FLEVEL <FL3>))
          (TRUE-IN-STATE-STATIC (NEXT <FL2> <FL3>))
          (TRUE-IN-STATE-STATIC (NEXT <FL1> <FL2>))
          (TRUE-IN-STATE-STATIC (PERSON <PERSON0>))
          (TRUE-IN-STATE-STATIC (PERSON <PERSON1>))
          (SOME-CANDIDATE-GOALS NIL) (ALL-SOLVED-ACTIONS NIL)))
 (THEN SELECT OPERATORS PERSIST)
```

Figura 7.5: Ejemplo de regla de persistir en el dominio Zenotravel.

de decisión podría ser la opción por omisión del planificador. Aunque el siguiente punto de decisión posterior al de éxito sea de fallo, no implica que todas las acciones elegidas por el planificador para satisfacer todas las metas en ese punto sean incorrectas, basta con que una no lo sea para que todo el nodo se considere de fallo. Por tanto, para hacer un aprendizaje `lazy` estricto y no generar reglas de control de estas selecciones, al módulo generador de reglas es preciso pasarle un parámetro más: el punto de decisión de fallo inmediatamente posterior al primer punto de éxito del que se van a aprender las reglas para obtener las acciones seleccionadas inicialmente (opción por omisión) por TGP.

Por ejemplo, la figura 7.6 muestra parte del árbol de búsqueda generado para un problema del dominio Driverlog. En este dominio existen paquetes que deben ser transportados a diferentes sitios en camiones. Los camiones han de ser conducidos por conductores que pueden estar en diferentes sitios y deben andar hasta llegar a ellos. Los caminos por los que pueden andar los conductores son diferentes de las carreteras por las que pueden ir los camiones (para una descripción más detallada ver el apéndice A.3). El nodo *Time: (5)* es de éxito, y el siguiente nodo expandido en el nivel 4 es de fallo. A continuación, se expanden varios nodos donde la búsqueda falla o que ya están memorizados como de fallo (en la figura sólo se representan con el nivel y la etiqueta porque no son relevantes para la explicación), hasta que llega al último nodo de éxito, *Time: (4)*, representado en la figura. El módulo generador de reglas recibe los dos nodos de éxito y el nodo de fallo. Con aprendizaje *eager* se generarían tres reglas de seleccionar operador, una por cada nueva meta conseguida; es decir, `((load-truck package1 truck2 s2) (in package1 truck2))`, `((walk driver2 p2-1 s1) (at driver2 s1))` y `((unload-truck package2 truck2 s2) (at package2 s2))`. Sin embargo, con aprendizaje *lazy*, la acción `((load-truck package1 truck2 s2) (in package1 truck2))` es la decisión por omisión de TGP, tal como se ve en el no-

do *Time:(4) (FAILURE)*, y no se genera una regla para dicha meta.

```
Time:(5) (SUCCESS)
Goals=((AT TRUCK2 S2) (AT PACKAGE2 S2) (AT TRUCK1 S2) (AT DRIVER2 S1)
       (IN PACKAGE1 TRUCK2) (IN PACKAGE3 TRUCK2)
       (DRIVING DRIVER1 TRUCK2))
Actions=(
(DRIVE-TRUCK TRUCK2 S2 S0 DRIVER1) (AT TRUCK2 S0))
(UNLOAD-TRUCK PACKAGE1 TRUCK2 S0) (AT PACKAGE1 S0))
(UNLOAD-TRUCK PACKAGE3 TRUCK2 S0) (AT PACKAGE3 S0))
(DISEMBARK-TRUCK DRIVER1 TRUCK2 S0) (AT DRIVER1 S0))
(WALK DRIVER1 S0 P0-1) (AT DRIVER1 P0-1))
(WALK DRIVER1 P0-1 S1) (AT DRIVER1 S1)))

Time:(4) (FAILURE)
Goals=((IN PACKAGE2 TRUCK1) (AT PACKAGE1 S2) (DRIVING DRIVER1 TRUCK2)
       (IN PACKAGE3 TRUCK2) (AT DRIVER2 S1) (AT TRUCK1 S2)
       (AT TRUCK2 S2))
Actions=(
(LOAD-TRUCK PACKAGE1 TRUCK2 S2) (IN PACKAGE1 TRUCK2))
(UNLOAD-TRUCK PACKAGE2 TRUCK1 S2) (AT PACKAGE2 S2))

(DRIVE-TRUCK TRUCK2 S2 S0 DRIVER1) (AT TRUCK2 S0))
(UNLOAD-TRUCK PACKAGE1 TRUCK2 S0) (AT PACKAGE1 S0))
(UNLOAD-TRUCK PACKAGE3 TRUCK2 S0) (AT PACKAGE3 S0))
(DISEMBARK-TRUCK DRIVER1 TRUCK2 S0) (AT DRIVER1 S0))
(WALK DRIVER1 S0 P0-1) (AT DRIVER1 P0-1))
(WALK DRIVER1 P0-1 S1) (AT DRIVER1 S1)))

Time:(4) (FAILURE) Time:(4) (FAILURE) Time:(4) (FAILURE) Time:(3) (FAILURE)
Time:(2) (FAILURE) Time:(2) (FAILURE) Time:(3) (FAILURE) Time:(2) (MEMO-FAILURE)
Time:(2) (FAILURE) Time:(2) (MEMO-FAILURE)  Time:(3) (FAILURE) Time:(2) (FAILURE)
Time:(2) (MEMO-FAILURE)  Time:(4) (FAILURE) Time:(3) (MEMO-FAILURE)  Time:(3) (FAILURE)
Time:(2) (MEMO-FAILURE)  Time:(2) (FAILURE) Time:(2) (MEMO-FAILURE)  Time:(3) (MEMO-FAILURE)

Time:(4) (SUCCESS)
Goals=((IN PACKAGE2 TRUCK2) (AT DRIVER2 P2-1) (AT PACKAGE1 S2)
       (DRIVING DRIVER1 TRUCK2) (IN PACKAGE3 TRUCK2) (AT TRUCK1 S2)
       (AT TRUCK2 S2))
Actions=(
(LOAD-TRUCK PACKAGE1 TRUCK2 S2) (IN PACKAGE1 TRUCK2))
(WALK DRIVER2 P2-1 S1) (AT DRIVER2 S1))
(UNLOAD-TRUCK PACKAGE2 TRUCK2 S2) (AT PACKAGE2 S2))

(DRIVE-TRUCK TRUCK2 S2 S0 DRIVER1) (AT TRUCK2 S0))
(UNLOAD-TRUCK PACKAGE1 TRUCK2 S0) (AT PACKAGE1 S0))
(UNLOAD-TRUCK PACKAGE3 TRUCK2 S0) (AT PACKAGE3 S0))
(DISEMBARK-TRUCK DRIVER1 TRUCK2 S0) (AT DRIVER1 S0))
(WALK DRIVER1 S0 P0-1) (AT DRIVER1 P0-1))
(WALK DRIVER1 P0-1 S1) (AT DRIVER1 S1)))
```

Figura 7.6: Ejemplo de puntos de decisión para aprendizaje *lazy* en el dominio Driverlog.

Regresión de metas

La regresión de metas consiste en obtener los literales del estado actual de planificación que sean requeridos por un operador, bien porque son directamente las precondiciones del operador, o bien porque son requeridos por las acciones necesarias para satisfacer las precondiciones del operador. El algoritmo para obtener la regresión de metas R, de una acción A_i, considerando el estado S, y la solución del problema P se muestra en la figura 7.7.

Además de hacer la regresión de metas sobre la acción seleccionada por la regla, se hace sobre las metas del meta-predicado `SOME-CANDIDATE-GOALS` para obtener también los literales del estado inicial requeridos por el argumento de este meta-predicado. La regresión de metas de una meta, es la regresión de metas

REGRESIÓN-METAS (A_i:acción, S:estado inicial, P:plan)

Q = precondiciones (A_i)
$R = \varnothing$
while $Q \neq \varnothing$
    q = primer elemento de Q
    if $q \in S$ then $R = R \cup q$
    else
        A_q = acción que consiguió q según P
        Q_r = precondiciones (A_q)
        $Q = Q \cup Q_r$
    quitar q de Q
devolver R

Figura 7.7: Algoritmo para obtener la regresión de metas de una acción.

de la acción que la establece según el plan solución.

El mayor problema para realizar la regresión de metas en planificación basada en grafos, es que el estado permanece invariante durante toda la búsqueda. El estado inicial se añade al grafo del plan en el nivel 0. Posteriormente el grafo del plan se amplia, pero el estado correspondiente, en el momento de seleccionar las acciones por el planificador, no está claramente definido; el grafo del plan no es suficiente para determinarlo. Esta información es fundamental para que las reglas de control generadas disciernan entre unos estados y otros. La opción adoptada en este trabajo es hacer la regresión siempre al estado inicial del problema y añadir el meta-predicado `ALL-SOLVED-ACTIONS` para recoger la información de cómo puede haber cambiado dicho estado. Esto es tan sólo una aproximación. Tiene la ventaja de que a la hora de hacer la equiparación de las reglas, para comprobar si se pueden disparar (proceso bastante costoso), basta con realizarlo una vez, como se explicará en la sección 7.4. Otra alternativa estudiada fue tratar de buscar dicho estado en el grafo del plan, pero los resultados fueron peores. Para ello, cuando se generan las reglas de control a partir de un punto de decisión de un nivel, i, se considera que el estado de planificación está compuesto por los literales l_i del grafo del plan cuya etiqueta sea menor o igual que i y además no estén negados (no exista en ese nivel o inferior el literal (not l_i)). Esto, además de hacer el proceso de equiparación muy costoso, sigue sin discernir correctamente los estados ya que el nivel del grafo dónde aparece por primera vez un literal tampoco es muy determinante en la búsqueda. Generalmente los literales se añaden al grafo del plan en niveles relativamente bajos comparados con los niveles en los que se realizan las búsquedas de las acciones que los satisfacen.

7.4. Equiparación y ejecución de reglas

Cuando el proceso de aprendizaje termina, se crea un fichero con un conjunto de reglas de control que son usadas en la resolución de problemas futuros para podar el árbol de búsqueda. Para verificar si una regla se cumple, se realiza un proceso de equiparación simple lineal explicado en la sección 5.2.

El coste computacional de este proceso se incrementa con el número de variables de las reglas, llegando incluso a hacerlo inviable (problema de la utilidad). De ahí, la conveniencia de reducir, tanto los meta-predicados de una regla (regresión de metas), como sus argumentos (sólo se consideran `SOME-CANDIDATE-GOALS` y `ALL-SOLVED-ACTIONS` que tienen alguna relación de mutex con la meta implicada en la regla). En planificación basada en grafos, el problema de la utilidad es especialmente crítico debido a la cantidad de información recogida en cada punto de decisión. Por otra parte, el algoritmo de Graphplan es muy rápido, siendo necesario que las reglas de control eviten explorar muchos nodos de fallo para que el tiempo de ahorro de exploración compense al de equiparación.

La equiparación de las reglas durante la resolución de un problema en TGP se hace en dos fases:

- Al principio de la resolución de cada problema, se equiparan los meta-predicados `TRUE-IN-STATE` una única vez, ya que, como se explicó antes, sólo dependen del estado inicial del problema.
- El resto de meta-predicados se comprueban antes de iniciar una asignación a un conjunto de metas en un nivel, partiendo de los pares variable-valor encontrados en el paso anterior.

El algoritmo de TGP intenta como primera opción persistir una meta. Si la persistencia no es posible, intenta el primer operador que tenga asignado para satisfacerla. Cada meta tiene un campo con la lista de operadores instanciados que pueden conseguirla. Dichos operadores están ordenados según el nivel del grafo del plan en que se introdujeron. En caso de que una regla de selección de operador se dispare, se reordena esta lista poniendo como primer operador el indicado por la regla. En realidad, las reglas no son de selección de operador sino de preferencia, ya que el resto de operadores no se quitan de la lista. No se ha descubierto la manera de implementar reglas de selección estrictas en TGP porque, en un mismo problema de planificación, una misma meta puede ser satisfecha por más de un operador en distintas partes del plan; es decir, el algoritmo de TGP puede realizar diferentes procesos de búsqueda, en diferentes niveles, para encontrar operadores que satisfagan la misma meta y en cada búsqueda el operador encontrado ser diferente. Sin embargo, dada la simplificación en la representación del grafo del plan realizada por TGP, las metas se almacenan una sola vez en el grafo, habiendo una única lista de operadores instanciados que pueden conseguirlas que se utiliza en

todas las búsquedas en que dicha meta deba ser alcanzada. En el consecuente de las reglas se ha mantenido la denominación SELECT en vez de PREFER porque, desde el punto de vista de adquisición del conocimiento, las reglas aprendidas son de selección, lo que ocurre es que, tal como está implementado el algoritmo de TGP, no es posible adaptarlas como tales en el proceso de equiparación y ejecución de reglas. Sería necesario un cambio en el propio algoritmo de TGP que sobrepasa los objetivos de esta tesis.

Las reglas de selección de operadores cambian el comportamiento por omisión de TGP para que no se intente primero la persistencia, a no ser que, en un mismo punto de decisión, también se dispare una regla de persistencia. En este caso, se hace la ordenación de los operadores, pero se prueba primero la persistencia.

7.5. Experimentos y resultados

Los experimentos descritos en este apartado tienen por objetivo demostrar la validez del método de aprendizaje propuesto, para incrementar la eficiencia del propio planificador TGP en la resolución de problemas. Se extrae el conocimiento de control de un conjunto de problemas de entrenamiento y se aplica en la resolución de otros problemas diferentes, del mismo dominio, esperando que haya una disminución de los recursos consumidos por el ordenador en su resolución (tiempo y memoria) respecto a la no utilización de conocimiento de control. Se fija un tiempo máximo de resolución. El aprendizaje es *lazy* para corregir sólo las decisiones en que las heurísticas del planificador fallan y minimizar el problema de utilidad típico de EBL. En el capítulo siguiente se verá cómo se puede utilizar el conocimiento de control extraído de TGP en otra técnica de planificación. En este caso el tipo de aprendizaje que conviene hacer es *eager*.

Las variables independientes de la experimentación son:

- dominio de planificación,
- problemas: variando la complejidad, medida, por ejemplo, por el número de metas a satisfacer,
- métrica de calidad a utilizar: longitud del plan paralelo y número de operadores en la solución,

y las variables dependientes son:

- número y porcentaje de problemas de un mismo dominio resueltos en el tiempo límite de ejecución establecido,
- tiempo de CPU que se tarde en encontrar la solución del problema,

- calidad de la solución encontrada, según la métrica de calidad establecida,
- puntos de decisión: número de nodos del árbol de búsqueda. Representa el número total de veces que el planificador intenta buscar una asignación válida a un conjunto de metas.

La experimentación se ha hecho utilizando los dominios de las competiciones internacionales de planificación. TGP resuelve problemas PDDL con los requisitos *strips* y *equality*. Los dominios de las competiciones definidos con estos requisitos son las versiones STRIPS de:

- IPC02: Driverlog, Zenotravel, Satellite, Depots y Rover.
- IPC00: Miconic, Logistics, Blocks y FreeCell.

Los primeros experimentos se realizaron con estos dominios y utilizando los problemas definidos para las competiciones. TGP intentó resolver todos los problemas propuestos en las competiciones. En los dominios Depots, Rover y FreeCell no consiguió solucionar ninguno de los problemas por lo que se desecharon para el aprendizaje. En el dominio Blocks sólo resolvió un problema, pero no aprendió ninguna regla de él, por lo que también se descartó. En los otros dominios se utilizaron los problemas más sencillos (los A primeros) para hacer el aprendizaje. La tabla 7.1 resume los resultados obtenidos. Muestra el número de problemas (columnas Problemas) de aprendizaje (A) y de validación (V), los problemas resueltos por TGP sin conocimiento de control (TGP) y utilizando las reglas (GEBL) y el número de reglas generadas (Reglas). Se muestra tanto el número de problemas resueltos (Res) como el porcentaje (% Res) de resueltos respecto al total de problemas probados. El tiempo de ejecución fue 60s.

DOMINIO	Problemas		TGP		GEBL		
	A	V	Res	% Res	Res	% Res	Reglas
Driverlog	3	20	9	45 %	9	45 %	8
Zenotravel	3	20	8	40 %	8	40 %	1
Logistics	2	30	4	13 %	0	0 %	6
Satellite	3	20	4	20 %	3	15 %	6
Miconic	10	150	33	22 %	31	21 %	8

Tabla 7.1: Resumen resultados de GEBL en TGP con los problemas de las IPCs. Número de problemas resueltos.

Estos resultados reflejan que el método de aprendizaje no hace que se resuelvan más problemas. Las tablas detalladas con los valores de las variables dependientes mencionadas antes están en el apéndice C.1. La tabla 7.2 muestra un resumen de dichos resultados. Muestra la suma total de las variables medidas de todos los problemas resueltos por TGP tanto con reglas de control como sin ellas: el tiempo de

resolución en segundos (TE), el tiempo de equiparación de reglas (TM), los puntos de decisión (PD), la calidad de las soluciones medidas como la longitud del plan paralelo (Q) y el número de reglas aprendidas (R). En todos los casos el tiempo de CPU necesario para resolver los problemas es peor con reglas de control que sin ellas. La calidad de las soluciones es la misma y los puntos de decisión, en el dominio Driverlog, Zenotravel y Miconic es prácticamente igual y en el Satellite, con las reglas, se incrementan en un 13 %. Esto demuestra que apenas hay diferencia entre usar las reglas de control y no usarlas. En el dominio Logistics, se comprobó experimentalmente que la disminución en el número de problemas resueltos al utilizar las reglas es debido a la expiración del tiempo de ejecución (60s) mas que a que las reglas se disparen en sitios incorrectos.

DOMINIO	TGP			GEBL				
	TE(s)	PD	Q	TE(s)	TM(s)	PD	Q	R
Driverlog	51.50	682321	167	50.30	0.12	682321	167	8
Zenotravel	15.58	193608	43	16.64	0.95	193605	43	1
Satellite	1.35	59729	26	4.40	3.08	69072	26	6
Miconic	107.62	4817076	322	167.56	70.73	4817190	322	8

Tabla 7.2: Resumen resultados de GEBL en TGP con los problemas de las IPCs. Tiempo, calidad y puntos de decisión acumulados de los problemas resueltos por ambos sistemas.

Se hicieron otros experimentos generando más problemas aleatorios, tanto de entrenamiento como de validación. Se incluyó un dominio más (Robocare, ver apéndice A.7) no utilizado en las competiciones porque es un dominio especialmente idóneo para generar planes paralelos. Los resultados detallados están en el apéndice C.1 y en las tablas 7.4 y 7.3 se muestra el resumen. La primera tabla refleja el número de problemas resueltos por cada sistema, junto con los problemas utilizados para validar y entrenar, y la segunada tabla muestra el acumulado de los valores de las variables independientes medido para los problemas que resuelven ambos sistemas. Los resultados vuelven a confirmar que la utilización de las reglas de control casi no varía el comportamiento de TGP, simplemente se observa un incremento en el tiempo de resolución debido al proceso de equiparación de reglas. En los tres primeros dominios, tanto el número total de puntos de decisión como la calidad total tienen el mismo valor en ambos casos, sólo en el último dominio se aprecia una reducción del 85 % en el número de puntos de decisión, pero la calidad de las soluciones es igual y el tiempo total en encontrar las soluciones se incrementa en un 92 % con las reglas. Este aumento es debido al tiempo de equiparación de reglas. En el dominio Logistics este tiempo es especialmente alto, incluso haciendo que expire el tiempo total antes de encontrar las soluciones; por eso se resuelven menos problemas con reglas que sin ellas. El tiempo total de ejecución fue de 60s.

DOMINIO	Problemas		TGP		GEBL		
	A	V	Res	% Res	Res	% Res	Reglas
Driverlog	23	100	100	100 %	100	100 %	8
Zenotravel	200	100	21	21 %	21	21 %	1
Logistics	200	100	93	93 %	4	4 %	6
Robocare	27	336	68	20 %	68	20 %	12

Tabla 7.3: Resumen resultados de GEBL en TGP con los problemas aleatorios. Número de problemas resueltos.

DOMINIO	TGP			GEBL				
	TE(s)	PD	Q	TE(s)	TM(s)	PD	Q	R
Driverlog	41.79	224672	769	42.24	0.07	224672	769	8
Zenotravel	0.30	195	43	0.20	0.00	195	43	1
Logistics	3.35	22487	34	150.14	146.79	22485	34	6
Robocare	138.20	3745275	693	1771.76	1682.15	558265	693	12

Tabla 7.4: Resumen resultados de GEBL en TGP con los problemas aleatorios. Tiempo, calidad y puntos de decisión acumulados de los problemas resueltos por ambos sistemas.

7.6. Conclusiones y trabajos futuros

Se ha implementado un sistema de aprendizaje basado en EBL para extraer conocimiento de control de un planificador basado en grafos de planes, con el objetivo de mejorar la eficiencia de resolución del planificador de nuevos problemas. Sin embargo, los experimentos realizados demuestran que el sistema de aprendizaje no mejora dicha eficiencia. El tiempo necesario para resolver los problemas utilizando el conocimiento de control aprendido es mayor que sin utilizarlo y el número de nodos expandidos durante la búsqueda sólo disminuye ligeramente en algunos de los problemas, sin que, aparentemente, siga alguna pauta preestablecida. Esto hace que aunque se consiguiera mejorar el tiempo de equiparación de reglas, no se conseguiría un incremento significativo de la eficiencia del planificador. Únicamente en uno de los dominios probados, el Robocare en el que el paralelismo de las acciones es alto, se ha observado una reducción considerable de la memoria utilizando las reglas, pero la calidad de las soluciones permanece igual y el tiempo de ejecución aumenta considerablemente. La razón de este comportamiento puede ser que en este dominio no hay conflictos entre metas; es decir, ningún efecto de una acción borra una precondición de otra, y probablemente también el alto grado de paralelismo del dominio.

A continuación se enumeran los problemas detectados que hacen que el sistema de aprendizaje no mejore la eficiencia del planificador:

- Estado de planificación. En planificación basada en grafos no hay estados;

se tiene el estado inicial de cada problema y el grafo del plan. Estos no son suficientes para discernir los estados de planificación intermedios durante las decisiones parciales realizadas por el planificador, necesarios para generar las explicaciones durante el aprendizaje EBL.

- Problema de utilidad. Los puntos de decisión o nodos del árbol de búsqueda son demasiado grandes: contienen todas las metas que se intentan satisfacer en un nivel, más el plan parcial. Esto hace que las reglas generadas tengan muchas variables, con el consiguiente incremento del proceso de equiparación de reglas. Por otra parte, el algoritmo de Graphplan es especialmente rápido, por lo que necesita que las reglas eviten explorar muchos nodos de fallo para que el tiempo de ahorro de exploración compense al de equiparación.

- Tipo de conocimiento aprendido. Tal como está implementado el algoritmo de búsqueda, cada meta tiene un campo donde se representan los posibles operadores que la pueden establecer y las metas se almacenan sólo en el grafo del plan una única vez (etiquetándolas con el nivel en que inicialmente se introducen en el grafo). Una misma meta se puede satisfacer varias veces en la resolución de un mismo problema; es decir, que para encontrar la solución puede ser necesario satisfacer la misma meta en distintas partes del plan utilizando diferentes operadores instanciados en cada caso. Pero la información para satisfacerla en todos los casos es única: la del campo de la meta donde se guardan los posibles operadores que la pueden establecer. Si una regla de selección de un operador se dispara para alcanzar una meta en un punto dado, el resto de operadores considerados para resolver esa meta se debería quitar de este campo, pero si se quitan, en el siguiente punto del algoritmo de planificación en que la misma meta tiene que ser satisfecha otra vez no los puede utilizar. Esto hace que las reglas sólo pueden ser de preferencia y no de selección; es decir, el operador seleccionado por la regla se puede colocar al principio de los operadores que alcanzan la meta, pero el resto de operadores no se pueden quitar. Aunque en el consecuente de las reglas generadas por GEBL se ha mantenido el nombre SELECT, el módulo de equiparación las considera de preferencia en vez de selección. No se utiliza el nombre PREFER porque desde el punto de vista de adquisición del conocimiento sería deseable que fueran reglas de selección.

Se puede concluir que la forma como se ha aplicado EBL al planificador TGP en esta tesis no incrementa la eficiencia del planificador. Las reglas aprendidas diferencian correctamente los estados de planificación, no se disparan en sitios incorrectos, pero no generalizan suficientemente para incrementar el rendimiento de TGP; son reglas demasiado específicas. Mejores resultados han dado las investigaciones que aceleran episodios consecutivos de búsqueda en un mismo problema evitando incurrir en errores anteriores, más que intentar generalizar las decisiones correctas a otros problemas. Sin embargo, se puede reutilizar el conocimiento

aprendido para aplicarlo a otro tipo de planificador que no presente los problemas anteriores. Por ejemplo, a planificación bidireccional. En la siguiente parte de la tesis se explica las adaptaciones necesarias para que GEBL obtenga conocimiento de control válido para un planificador bidireccional.

Como trabajos futuros se podría aplicar EBL para generar reglas de control que aprendan *memos* genéricos que se puedan aplicar a diferentes problemas. También, se puede aprender a ordenar las metas antes de cada una de las búsquedas realizadas por TGP en los distintos niveles. Kambhampati, en uno de sus trabajos, las ordenó de mayor a menor nivel en el grafo del plan, pero en TGP se defiende que es preferible el orden inverso. Por tanto, es posible que el orden más adecuado dependa de cada problema en particular.

Capítulo 8

Metodología para aplicar EBL a sistemas de planificación

En los capítulos anteriores se han descrito dos sistemas de aprendizaje desarrollados para extraer conocimiento de control de diferentes paradigmas de planificación: planificación híbrida HTN-POP y planificación basada en grafos de planes. Ambos sistemas están inspirados en otro sistema de aprendizaje aplicado a planificación bidireccional. El método de aprendizaje utilizado en todos es EBL. En este capítulo se pretende generalizar los tres sistemas para definir una metodología de diseño y desarrollo de sistemas de aprendizaje de conocimiento de control basada en la explicación que permita sistematizar los métodos y técnicas necesarias para aprender conocimiento de control de diferentes técnicas de planificación.

En la siguiente sección se describe la metodología. En la sección 8.2 se enumeran los requisitos del paradigma de planificación necesarios para aplicar la metodología. Por último, se analiza el conocimiento de control que se puede extraer con su utilización.

8.1. Guía para aplicar EBL a sistemas de planificación

Los pasos requeridos para aplicar EBL a sistemas de planificación, cuando se explican las decisiones de éxito del planificador son:

- Identificar el proceso de búsqueda del algoritmo de planificación, identificando los nodos del árbol de búsqueda. Esto puede no ser inmediato como ocurre en GEBL por la diferencia entre el algoritmo de búsqueda y el de planificación. Es decir, planificar (a “grosso modo”) consiste en encontrar operadores instanciados del dominio de planificación que satisfagan unas metas y para ello es preciso realizar un proceso de búsqueda en el espacio de pro-

blemas. Cada paradigma de planificación considera un espacio de problemas diferentes y aplica un determinado algoritmo de búsqueda. Los operadores de búsqueda no tienen porqué coincidir con los operadores de planificación. Por ejemplo, en TGP un estado de búsqueda (nodo del árbol) viene determinado por 3 factores: un conjunto de metas, el plan parcial y el nivel en el grafo del plan, porque TGP busca asignaciones de operadores instanciados del dominio de planificación que satisfagan un conjunto de metas sin violar ninguna relación de mutex, para lo cual necesita conocer el plan parcial y el nivel del grafo. Los operadores de búsqueda en TGP transforman un estado del espacio de estados de planificación en un estado del espacio de problemas de TGP compuesto también por un conjunto de metas, un plan parcial y un nivel, tal como se explicó en la sección 7.2. Sin embargo, los operadores de planificación en TGP son asignar una acción del dominio para alcanzar una meta, añadir o borrar relaciones de mutex entre nodos, almacenar un conjunto de metas como *memo* .... Por tanto, en TGP no hay coincidencia entre los operadores de búsqueda y los de planificación.

Pero, en HYBIS cada nodo del espacio de búsqueda representa una tarea pendiente de planificación que pueden ser metas pendientes, amenazas, interferencias, inconsistencias en el orden o cómo realizar un refinamiento jerárquico; y los nodos hijos representan las diferentes formas en que dicha tarea se puede resolver. En este caso, los operadores de búsqueda coinciden con los operadores de planificación. Cada nodo lleva asociado toda la información que el algoritmo necesita para resolver la tarea.

En PRODIGY hay tres tipos diferentes de nodos de búsqueda: pueden representar una meta, un operador o una sustitución de variables y los operadores de búsqueda de cada tipo son diferentes: en los nodos de tipo meta se busca un operador que la satisfaga generando un nodo hijo de tipo operador por cada posible operador del dominio que tenga dicha meta en la lista de efectos añadidos; en los nodos de tipo operador se busca la asignación de sus variables generando nodos hijos de tipo sustitución y en los nodos de tipo sustitución PRODIGY decide entre aplicar el operador instanciado o explorar otro nodo de tipo meta.

- Identificación de los posibles puntos de aprendizaje, estudiando los puntos de retroceso del algoritmo de búsqueda. Una vez identificado el árbol de búsqueda resulta inmediato encontrar los nodos en que el algoritmo de búsqueda tiene que retroceder porque la solución no se encuentra en la primera opción explorada. Generalmente todos los nodos son posibles puntos de retroceso, pero en cada algoritmo puede haber diferentes opciones que se puedan aprender de forma independiente y estudiar la conveniencia, en cada caso, de aprender esas decisiones o no. Por ejemplo, en HEBL de los diferentes tipos de nodos que hay sólo se aprende de los de tipo meta pendiente y refinamiento jerárquico, mientras que en HAMLET se aprende de todos. En

GEBL sólo hay un tipo de nodo del que aprender.

- Identificar, en los puntos anteriores, el tipo de acción realizada por el planificador para definir los diferentes tipos de reglas de control que se pueden generar. Por ejemplo, en GEBL hay un sólo tipo de punto de decisión, pero se pueden aprender dos acciones diferentes: el operador del dominio de planificación que consiga una de las metas o persistir dicha meta. En HEBL en los nodos de tipo meta se aprende el operador del dominio que la satisfaga diferenciando entre si dicho operador ya ha sido utilizado en el plan parcial o no; y en los nodos de tipo refinamiento jerárquico se aprende el método de expansión a utilizar. En HAMLET de cada punto de decisión se aprende una acción diferente: una meta de entre un conjunto de metas para empezar a trabajar sobre ella, un operador del dominio para satisfacer una meta, un conjunto de sustituciones que instancien las variables libres del operador elegido o la opción de aplicar un operador instanciado o empezar a resolver otra meta.

- Determinar si el algoritmo de planificación permite distinguir los estados en cada decisión realizada identificando si el propio algoritmo proporciona algún mecanismo para tal diferenciación o se pueden hacer aproximaciones. Estos estados son lo utilizados por EBL para explicar las circunstancias en que se produce cada decisión durante la resolución de un problema para generar la parte de precondiciones (meta-predicados) de las reglas que permitan identificar en resoluciones futuras situaciones similares de forma unívoca. En el caso de PRODIGY el propio algoritmo de planificación actualiza estos estados cuando aplica un operador instanciado. Sin embargo, en TGP el algoritmo de planificación sólo necesita explorar el grafo del plan que permanece invariante durante un episodio de búsqueda; es decir, no hay un estado diferente en cada decisión tomada por el algoritmo de búsqueda por lo que es necesario hacer aproximaciones que permitan discernir los estados en otros procesos de búsqueda. En el caso de HYBIS el algoritmo POP no modifica el estado pero se puede obtener a través del conjunto de enlaces causales, tal como se explicó en la sección 6.2.2.

- Preparar los problemas de entrenamiento de los que aprender que el planificador ejecutará de uno en uno. El tamaño y complejidad de los problemas de entrenamiento es un factor importante en EBL. Conviene que sean sencillos para que el planificador pueda resolverlos, requisito imprescindible para iniciar el proceso de aprendizaje, y también para que las reglas aprendidas sean lo más general posibles, sin demasiadas precondiciones que incrementan el proceso de equiparación de reglas produciendo el problema de la utilidad. Cuanto más complejos son los problemas de aprendizaje más específicas son las reglas aprendidas; es decir, se disparan en menos situaciones, pero discriminan mejor unos casos de otros, por lo que hay que llegar a un equilibrio entre generalización y especialización no siempre alcanzado con técnicas

exclusivas de aprendizaje deductivo.

- Guardar el árbol de búsqueda realizado por el planificador hasta encontrar la solución de cada problema. Si el planificador no lo almacena, generar la traza con la información necesaria. Esto puede consumir mucha memoria y tiempo del ordenador por lo que conviene que sea opcional su generación y almacenamiento. Así, durante el aprendizaje, que normalmente se realiza con problemas pequeños, se activa y en el resto de casos se desactiva. En HYBIS es un parámetro de ejecución del planificador; en GEBL se ha introducido en el algoritmo de aprendizaje: cuando se aprende de un problema se genera y se guarda el árbol de búsqueda, en el resto de resoluciones no; y PRODIGY siempre guarda el árbol de búsqueda en todas las resoluciones de problemas que realiza ya que este planificador se implementó para estudiar la influencia de las técnicas de aprendizaje automático en planificación.

- Etiquetar los nodos de búsqueda para la identificación de los nodos de éxito y de fallo. Cada vez que se genera un nuevo nodo durante el proceso de planificación se le asigna una etiqueta. Si el nodo falla se cambia su etiqueta a fallo. Cuando el algoritmo de planificación encuentra una solución a todos los nodos del árbol de éxito, desde abajo hacia arriba, se les cambia la etiqueta a éxito. De forma similar los nodos tales que todos sus sucesores son nodos de fallo, se etiquetan como de fallo. Además de nodos de éxito y fallo puede existir otro tipo de nodos que también se quieran identificar. Por ejemplo, en HEBL hay ramas del árbol de búsqueda incompletas porque la heurística utilizada desaconseja seguir explorándolas antes de llegar a fallar, y se denominan nodos abandonados. El algoritmo de planificación de TGP almacena *memos* cuando en un nivel de búsqueda no es posible encontrar una asignación válida a todas las metas. Esos nodos GEBL los etiqueta como *memo-failure* ya que, aunque sean de fallo, el propio algoritmo de planificación lo detecta y no los explora. En PRODIGY hay nodos que no explora, que se denominan desconocidos o, incluso, que no ha llegado a generar, que se denominan no-intentados.

- Cada punto de decisión de éxito puede generar una o varias reglas de control cuyo consecuente sean la decisión tomada por el planificador. Se puede optar por un aprendizaje *eager* o *lazy* dependiendo de si se aprende de todos los puntos de decisión o sólo de aquéllos que no constituyeron la primera decisión del planificador. Hay que implementar una función que tome como entrada la lista de todos los nodos etiquetados y vaya devolviendo, consecutivamente, el nodo o nodos de los que aprender.

- Utilizar los meta-predicados básicos usados por el lenguaje común para explicar el estado del planificador en cada decisión, identificando, en el algoritmo de planificación, la manera de obtenerlos. En la en la sección 5.3 se describió a alto nivel cómo implementar dichas funciones. La mayoría de

los algoritmos de planificación definen funciones para obtener los valores necesarios para la definición de los meta-predicados más comunes, como la meta actual que está tratando de resolver, la lista de metas pendientes .... En caso de no existir habría que implementarlos. Para reducir el número y tamaño de los meta-predicados que preguntan por el estado de planificación se realiza una regresión de metas de forma que se incluyan sólo los literales que afectan directamente a la decisión considerada. La forma de hacer esta regresión depende del paradigma de planificación. Por ejemplo, en HEBL se utiliza el conjunto de enlaces causales (ver sección 6.2.2) y en GEBL se contemplan diferentes alternativas para hacerlo (ver sección 7.3.2).

- Estudiar otros aspectos del estado de planificación analizado por el planificador que puedan servir para definir nuevos meta-predicados específicos del paradigma de planificación utilizado. Por ejemplo, el nivel HTN en HEBL o el plan parcial en GEBL.

- Implementar o reutilizar un mecanismo de equiparación y ejecución de las reglas que se dispare en cada punto de decisión según se explicó en la sección 5.2.

- Realizar de forma opcional algún estudio de utilidad de las reglas generadas para restringir la cantidad de reglas de cada dominio y evitar los problemas típicos de utilidad de los sistemas de aprendizaje basados en la explicación. Por ejemplo, el realizado en HAMLET siguiendo la ecuación de Minton [Minton, 1988] en que la utilidad de una regla r se calcula como:

$$u(r) = s(r) \times p(r) - m(r)$$

donde $u(r)$ es una estimación de la utilidad de r, $s(r)$ es una estimación del tiempo de búsqueda que la regla ahorra cuando se usa, $p(r)$ es una estimación de la probabilidad de que la regla se dispare, y $m(r)$ es una estimación del coste de equiparación de la regla. Cada una de estas funciones HAMLET las estima de la siguiente forma:

  - $s(r)$: es el número de nodos que hay por debajo del nodo donde se aprende la regla, multiplicado por el tiempo que PRODIGY tarda en expandir un nodo. Se puede hacer una mejor estimación utilizando un segundo conjunto de entrenamiento, como hizo Minton, y calcular la media del número de nodos usados cada vez por la regla.
  - $p(r)$: es el número de veces en que r se dispara en los problemas de entrenamiento dividido entre el número de veces en que PRODIGY intenta usar la regla. Igual que antes, se puede realizar una mejor estimación utilizando un segundo conjunto de entrenamiento.

- $m(r)$: es el tiempo total consumido intentando equipar la regla durante toda la resolución del problema dividido entre el número de veces en que se intenta disparar.

8.2. Aplicabilidad de la metodología

En principio, cualquier técnica de planificación en que se puedan realizar los pasos anteriores podría ser susceptible de ser mejorada mediante conocimiento de control extraído con la metodología. Sin embargo, existen algunas características de los planificadores que pueden determinar el grado de idoneidad del método de aprendizaje:

- Deficiencias del algoritmo de planificación. Verificar si las mayores deficiencias del algoritmo de planificación pueden ser solventadas o atenuadas con el aprendizaje. Por ejemplo, la mayor deficiencia detectada en el algoritmo de GRAPHPLAN se deriva de las simetrías de los dominios que imposibilitan la expansión del grafo del plan y GEBL no soluciona este problema porque sólo se puede aplicar en la segunda fase del algoritmo, en la extracción de la solución, una vez expandido el grafo del plan hasta un nivel en que aparecen todas las metas del problema sin ser mutex. Sin embargo, tanto en PRODIGY como en HYBIS las mayores deficiencias de los algoritmos de planificación vienen de la exploración de los nodos de búsqueda en que sus heurísticas se equivocan generando muchas ramas erróneas que llevan a puntos muertos que consumen todos los recursos del ordenador antes de llegar a la solución. El método de aprendizaje ayuda a realizar esta búsqueda de forma más eficiente podando las ramas que no conducen a la solución.

- Estados de planificación. Para poder dar las explicaciones de las decisiones correctas del planificador es necesario definir los meta-estados cuando se toman. Por tanto, es necesario que el planificador diferencie estos meta-estados, tal como se explicó en la sección anterior. Si no los diferencia, hay que hacer aproximaciones que siempre introducen un factor de inexactitud, como ocurría en GEBL.

- Tamaño de los puntos de decisión. Si los nodos del algoritmo de búsqueda contienen demasiada información, como en TGP, las reglas generadas contendrán muchas variables generando un problema de utilidad. El tiempo de verificación de las reglas supera al tiempo que ahorran. El problema se agrava si el algoritmo de planificación es especialmente eficiente, en cuyo caso las reglas tendrán que evitar explorar muchos nodos para compensar el tiempo de equiparación. El algoritmo de búsqueda realizado por GRAPHPLAN es muy rápido porque hace la búsqueda en el grafo del plan donde todos

los literales están instanciados y no hay variables; es decir, se transforma un problema de lógica de predicados a lógica proposicional.

Una vez comprobado que la técnica de planificación reúne las características necesarias para aplicar el método de aprendizaje, e implementado el sistema de aprendizaje correspondiente, siguiendo los pasos descritos en la sección anterior; la forma de actuación para obtener el conocimiento de control que incremente la eficiencia del planificador en resoluciones futuras es la siguiente:

1. Seleccionar un conjunto de problemas de entrenamiento de un dominio dado según los criterios apuntados en la sección anterior.
2. Aplicar el sistema de aprendizaje con los problemas anteriores y generar el conjunto de reglas. Normalmente es preferible utilizar aprendizaje *lazy* para que las reglas sólo corrijan las decisiones erróneas del planificador.
3. Realizar algún estudio sobre la utilidad de las reglas obtenidas para restringir su número o tamaño a las realmente útiles según el criterio que se establezca.
4. Utilizar las reglas anteriores para guiar las búsquedas futuras del planificador al resolver nuevos problemas más complejos.

La forma de evaluación de la metodología es simplemente: ejecutar el planificador para que resuelva un conjunto de problemas de validación, diferentes a los de entrenamiento, utilizando el conocimiento de control aprendido por el sistema de aprendizaje y sin utilizarlo. Se compara el tiempo y memoria que el planificador necesita para resolver cada problema en los dos casos. También se puede medir la calidad de las soluciones generadas, según la métrica de calidad que se establezca (por ejemplo, número de operadores en el plan). Aunque el objetivo del sistemas de aprendizaje no sea mejorar dicha calidad, conviene estudiar cómo se ve afectada con el aprendizaje.

8.3. Conocimiento de control aprendido

Analizando las reglas de control generadas por los tres sistemas de aprendizaje tratados en esta tesis se observa que existe un conocimiento compartido por todos los sistemas y otro específico para cada uno. Por ejemplo, HEBL aprende dos tipos de reglas relacionadas con la selección del operador a utilizar para conseguir una meta dada, `select operator-new` y `select operator-plan`. GEBL también aprende dos tipos de reglas para seleccionar el operador: `select operator` y `select operator persist`. Por último, HAMLET aprende otros dos tipos de reglas relacionadas con el operador a seleccionar: las de `select operators` y las de `select bindings`. En la tabla 8.1 se resume el tipo de

reglas aprendidas por cada uno de los tres sistemas de aprendizaje. Si el sistema de aprendizaje genera reglas del tipo indicado en la columna Tipo, la casilla correspondiente indica el nombre utilizado en el consecuente de las reglas creadas, y si la casilla está vacía es que el sistema de aprendizaje no contempla la generación de ese tipo de reglas. En las figuras 8.1 a la 8.7 se muestra un ejemplo de cada una de estas reglas.

Tipo	HAMLET	HEBL	GEBL
Selección	SELECT	SELECT	
Preferencia			SELECT
Rechazo			
Operador	operators bindings	operator-new operator-plan	operator operator persist
Metas	goals		
Refinamiento jerárquico		expansion	

Tabla 8.1: Resumen del tipo de reglas aprendidas por los sistemas de aprendizaje HAMLET, HEBL y GEBL.

En todas las reglas se observa que hay meta-predicados comunes, como `current-goal`, `some-candidate-goals` y `true-in-state`, y otros específicos del paradigma de planificación, como `htn-level` y `operator-in-plan` en HEBL , o el meta-predicado `all-solved-actions` en GEBL. En la tabla 8.2 se muestra los meta-predicados utilizados por cada sistema de aprendizaje.

Meta-predicado	HAMLET	HEBL	GEBL
current-goal	X	X	X
some-candidate-goals	X	X	X
true-in-state	X	X	X
current-operator	X		
type-of-object	X		
htn-level		X	
operator-not-in-plan		X	
operator-to-expand		X	
components-of		X	
all-solved-actions			X

Tabla 8.2: Meta-predicados utilizados en las reglas aprendidas por los sistemas de aprendizaje HAMLET, HEBL y GEBL.

El tratamiento del tipo de las variables difiere en los tres sistemas: en HEBL se indica en el nombre de las variables utilizadas en las reglas, en GEBL se tratan como `true-in-state` y en HAMLET como meta-predicados `type-of-object`.

También se observan diferencias en las reglas, derivadas del algoritmo de planificación. Por ejemplo, HYBIS cuando tiene que seleccionar un operador para conseguir una meta diferencia entre si el operador ya ha sido utilizado en el plan parcial o debe ser uno nuevo todavía no aplicado; por eso se generan dos tipos de reglas diferentes. En PRODIGY, la selección del operador instanciado se hace en dos puntos de decisión, primero se selecciona el operador y luego su instanciación. Todas estas diferencias deben ser reflejadas en el conocimiento de control generado.

Por último, en cada paradigma de planificación existe un conocimiento específico que puede ser aprendido con la metodología. Por ejemplo, en HTN se puede aprender el método de expansión; en POP, cómo resolver las amenazas; en planificación basada en grafos de planes, las metas que persisten en cada nivel. La parte de las precondiciones de estas reglas es similar a las otras pero el consecuente es diferente.

```
(control-rule INDUCED-SELECT-FLY-L188-03-DECOX-157992
  (if (and (current-goal (at <aircraft-157992-3> <city-157992-4>))
           (true-in-state (at <aircraft-157992-3> <city-157992-1>))
           (true-in-state (fuel-level <aircraft-157992-3>
                          <flevel-157992-2>))
           (some-candidate-goals nil)
           (type-of-object <flevel-157992-2> flevel)
           (type-of-object <city-157992-1> city)
           (type-of-object <city-157992-4> city)
           (type-of-object <aircraft-157992-3> aircraft)))
(then select operators fly))
```

Figura 8.1: Ejemplo de regla de control en HAMLET de selección de un operador.

```
(control-rule INDUCED-SELECT-BIND-BOARD-L138-02-DECOX-140063
  (if (and (current-operator board)
           (current-goal (in <person-140063-2> <aircraft-140063-3>))
           (true-in-state (at <person-140063-2> <city-140063-4>))
           (true-in-state (fuel-level <aircraft-140063-3>
                          <flevel-140063-1>))
           (some-candidate-goals nil)
           (type-of-object <flevel-140063-1> flevel)
           (type-of-object <person-140063-2> person)
           (type-of-object <aircraft-140063-3> aircraft)
           (type-of-object <city-140063-4> city)))
  (then select bindings
        ((<p> . <person-140063-2>) (<a> . <aircraft-140063-3>)
         (<c> . <city-140063-4>))))
```

Figura 8.2: Ejemplo de regla de control en HAMLET de selección de bindings.

```
(control-rule regla-10
 (if (and (htn-level (1))
          (current-goal (contains <flour> <rtank>))
          (some-candidate-goals (
                              (state <nozzleline-simple%%additive-unit> off)
                              (contains <fermento> <reactor-papilla%%reactor>)
                              (state <nozzleline-simple%%additive-unit> open)
                              (state <reactor-papilla%%reactor> ready)
                              (bottled <papilla>)))
          (operator-not-in-plan (online <transportline%%additive-transport1>))
          (true-in-state (contains <flourl> <additive1>))
          (true-in-state (state <transportline%%additive-transport1> off))))
 (then select operator-new (online <transportline%%additive-transport1>)))
```

Figura 8.3: Ejemplo de regla de control en HEBL de selección de un operador nuevo del dominio.

```
(control-rule regla-1
 (if (and (htn-level (2))
          (current-goal (flow <flour> <additive1> <rtank>))
          (some-candidate-goals nil)
          (operator-in-plan (on <belt%%belt1>))
          (true-in-state (contains <flour> <additive1>))
          (true-in-state (state <belt%%belt1> off))))
 (then select operator-plan (on <belt%%belt1>)))
```

Figura 8.4: Ejemplo de regla de control en HEBL de selección de un operador ya presente en el plan parcial.

```
(control-rule regla-24
 (if (and (htn-level (2))
          (operator-to-expand (off <line-3prod%%trans-3> <r4> <r5> <i3> <?dst>))
          (true-in-state (state <valve-agg%%line4> off))
          (true-in-state (state <valve-agg%%line5> off))
          (true-in-state (state <valve-agg%%line9> off))
          (components-of (<line-3prod%%trans-3> (<valve-agg%%line4>
                           <valve-agg%%line5> <valve-agg%%line9>)))))
 (then select expansion
  ((state <valve-agg%%line4> off)
   (state <valve-agg%%line5> off)
   (state <valve-agg%%line9> off))))
```

Figura 8.5: Ejemplo de regla de control en HEBL de selección de un método de expansión.

```
(control-rule regla-ZENO-TRAVEL-ZENO1-e1
 (if (and (current-goal (at <person0> <city1>))
          (true-in-state (at <person0> <city0>))
          (true-in-state (at <plane1> <city1>))
          (true-in-state (fuel-level <plane1> <fl3>))
          (true-in-state (aircraft <plane1>))
          (true-in-state (city <city0>))
          (true-in-state (city <city1>))
          (true-in-state (flevel <fl1>))
          (true-in-state (flevel <fl2>))
          (true-in-state (flevel <fl3>))
          (true-in-state (next <fl2> <fl3>))
          (true-in-state (next <fl1> <fl2>))
          (true-in-state (person <person0>))
          (some-candidate-goals nil)
          (all-solved-actions nil)))
 (then select operators (debark <person0> <plane1> <city1>))
```

Figura 8.6: Ejemplo de regla de control en GEBL de selección de un operador.

```
(control-rule regla-ZENO-TRAVEL-L004-01-p1
 (if (and (current-goal (in <person1> <plane10>))
          (true-in-state (at <person1> <city0>))
          (true-in-state (at <plane1> <city0>))
          (true-in-state-static (aircraft <plane1>))
          (true-in-state-static (city <city0>))
          (true-in-state-static (city <city1>))
          (true-in-state-static (person <person1>))
          (some-candidate-goals nil)
          (all-solved-actions ((debark <person1> <plane1> <city1>)))))
 (then select operators persist))
```

Figura 8.7: Ejemplo de regla de control en GEBL de persistir

Parte III

Transferencia de conocimiento de control

Capítulo 9

Transferencia entre sistemas de planificación

El objetivo de esta parte de la tesis es estudiar la viabilidad de transferir conocimiento de control entre distintas técnicas de planificación: cómo extraer conocimiento de control (o heurísticas) de un planificador para posteriormente aplicarlo a otro diferente, con el fin de mejorar la eficiencia en la resolución de problemas del segundo planificador o incrementar la calidad de los planes que encuentra. Como se ha dicho, no existe un planificador óptimo que destaque sobre todos los demás para todos los dominios y problemas; cada técnica tiene sus ventajas e inconvenientes. Por otra parte, hay planificadores que, aunque su algoritmo de planificación no sea tan eficiente como el de otros, presentan otras ventajas que, para algunos casos, puede resultar aconsejable su utilización. Por ejemplo, PRODIGY permite definir y manejar diferentes métricas de calidad, razonando sobre múltiples criterios; flexibilidad para definir fácilmente diferentes comportamientos (a través de las funciones que se pueden disparar cada vez que un nodo se expande, llamadas *handlers*); tiene capacidad para representar y razonar con variables numéricas en un mismo predicado (no posibles en PDDL); permite definir restricciones de los valores que pueden tener las variables de los operadores llamando a funciones LISP definidas por el usuario; permite incorporar conocimiento de control independiente que guíe la búsqueda y explorar el árbol de búsqueda generado en la resolución de los problemas para analizar cada una de las decisiones tomadas por el planificador. Además, PRODIGY se ha integrado con un *scheduler* permitiendo el razonamiento temporal y con recursos (sistema IPSS [Rodríguez-Moreno *et al.*, 2004]). Sin embargo, hay dominios en los que la eficiencia de PRODIGY es inferior a la de otros planificadores, como pueden ser los basados en grafos de planes.

El objetivo a corto plazo (y en el que se centra esta tesis), es transferir conocimiento de control de un planificador basado en grafo de planes al planificador bidireccional PRODIGY. A largo plazo, sería capturar heurísticas de cualquier técnica

de planificación para aplicarlas a cualquier otra técnica. Es decir, analizar las decisiones comunes realizadas por los distintos paradigmas de planificación y estudiar la posibilidad de representarlas y compartirlas entre ellos. Por ejemplo, muchos planificadores eligen un operador o una instanciación, de entre los definidos en el dominio, para alcanzar una determinada meta; o, eligen el orden en que resolver las metas.

Para poder transferir conocimiento de control de un planificador P1 a otro P2 se deben verificar las siguientes condiciones:

1. que existan decisiones comunes o similares entre ambos planificadores. Es decir, que alguna de las decisiones de P1 sea equivalente a decisiones en P2 directamente o tras alguna transformación que pueda realizarse de forma automática.

2. que exista un sistema de aprendizaje para extraer el conocimiento de control de P1,

3. que P2 disponga de un módulo capaz de interpretar y utilizar conocimiento de control (equiparación y ejecución de las reglas),

4. que exista un traductor sintáctico/semántico para que el lenguaje del conocimiento aprendido en P1 sea entendido por P2.

Para hacer transferencia de conocimiento de control del planificador basado en grafos, TGP, al planificador bidireccional, PRODIGY, se cumplen casi todos los requisitos anteriores:

1. Los dos planificadores tiene puntos de decisión comunes: qué operador elegir para satisfacer una meta y qué meta seleccionar para resolver primero.

2. Se tiene el sistema de aprendizaje GEBL que extrae conocimiento de control de TGP, que genera reglas de control de selección de un operador para resolver una meta.

3. PRODIGY permite incorporar heurísticas externas en forma de reglas de control.

Y falta por desarrollar lo siguiente:

- Ampliar GEBL para que aprenda reglas de selección de metas.

- Hacer un traductor del conocimiento generado por GEBL a la sintaxis de PRODIGY y a los puntos de decisión equivalentes.

En este capítulo se describe cómo se ha transferido conocimiento de control de TGP a PRODIGY. Primero, se comparan los algoritmos de búsqueda realizados por ambos planificadores para estudiar los puntos de decisión comunes. Luego, se describen las ampliaciones realizadas a GEBL para que genere reglas de selección de metas. Luego, se describe el traductor de las reglas generadas por GEBL. Después, se presentan los experimentos y resultados realizados. Por último, se hace un análisis del conocimiento de control transferido y se exponen las conclusiones obtenidas.

9.1. GRAPHPLAN **versus** PRODIGY

PRODIGY está basado en planificación bidireccional que combina búsqueda hacia atrás de las metas en el espacio de estados con simulaciones de ejecuciones de planes (ver sección 4.1) y TGP es un planificador temporal basado en grafos de planes que hace búsqueda hacia atrás sobre el grafo del plan (ver sección 4.6). Esto implica que existen las siguientes diferencias en los algoritmos de búsqueda de la solución de ambos planificadores:

- Árbol de búsqueda:
  - Los nodos del árbol de búsqueda en PRODIGY, cuando hace encadenamiento hacia atrás, pueden representar una meta, un operador o una asignación de valor a variables de cada operador.
  - Un nodo de búsqueda en TGP representa un conjunto de metas a las que se les busca una asignación posible de operadores en un nivel del grafo dado, junto con el plan parcial construido hasta ese instante.
- Puntos de decisión (posibles puntos de retroceso en la búsqueda):
  - Los puntos de decisión en PRODIGY pueden ser de cuatro tipos: selección de una meta sobre el conjunto de metas pendientes, selección de un operador para satisfacer la meta seleccionada, selección de los valores de las variables del operador elegido y decidir si aplicar un operador instanciado o resolver otra meta.
  - Los puntos de decisión en TGP son: dado un conjunto de metas en un nivel del grafo, determinar cuáles de ellas pueden persistir (se asignarán los operadores que las consigan en los siguientes niveles) y a cuáles se les pueden asignar un operador que las consigan en el nivel actual. Cada posible asignación de operadores genera un nuevo punto de decisión formado por las metas que persisten más las precondiciones de los operadores utilizados para conseguir las otras metas.

- Proceso de búsqueda:
  - PRODIGY combina búsqueda hacia atrás con simulaciones de ejecuciones de planes. Los operadores instanciados que antes se aplican durante la búsqueda en PRODIGY, constituyen las primeras acciones del plan solución.
  - TGP realiza un búsqueda hacia atrás, empezando en el último nivel del grafo y encontrando primero las últimas acciones que constituirán el plan solución.
- Estados:
  - PRODIGY modifica el estado dinámicamente durante el proceso de búsqueda según va aplicando operadores instanciados para satisfacer las metas.
  - TGP no modifica el estado durante la búsqueda; toda la información del estado está en el primer nivel del grafo del plan y permanece invariable durante un episodio de búsqueda.
- Lista de metas pendientes:
  - PRODIGY no añade precondiciones de operadores a la lista de metas pendientes que sean ciertas en el estado actual. Se ordenan según su definición en el fichero del problema.
  - TGP añade todas las precondiciones de los operadores alcanzados .[1]
- Soluciones de los problemas:
  - PRODIGY encuentra planes válidos de orden total.
  - TGP encuentra planes paralelos óptimos.

Con el fin de entender los dos algoritmos de planificación, se utiliza el ejemplo del dominio Zenotravel descrito en el capítulo anterior y representado en la figura 7.2. La traza con los puntos de decisión generados por TGP al resolver dicho problema se mostraba en la figura 7.3. La figura 9.1 representa el árbol de búsqueda generado por PRODIGY resolviendo el mismo problema. El algoritmo empieza seleccionando una meta sobre la que trabajar. Por omisión, PRODIGY la selecciona según la definición del fichero del problema eligiendo trabajar en

[1] En el artículo donde se presenta TGP, se afirma que importan la heurística (débil) de los CSP de ordenar las metas en forma decreciente, de mayor a menor dificultad. La dificultad de alcanzar una meta se mide según la distancia al estado inicial en el grafo del plan: cuanto más cerca esté más fácil será satisfacerla. Sin embargo, en el código tienen un comentario diciendo que funciona mejor ordenándolas al revés, de menor a mayor nivel del grafo, debido, probablemente, a que primero se prueba la persistencia. El código original, finalmente, no ordena las metas de ninguna forma. Para la transferencia de conocimiento convendría que se ordenaran en orden creciente, pero se ha utilizado el algoritmo original de TGP.

ellas según el orden en el que aparecen definidas. En este caso elige la meta `(at person0 city1)` que es la primera definida en el problema. Luego, selecciona el operador instanciado `(debark person0 plan1 city1)` (nodos 6 y 7) para alcanzarla. Las precondiciones del operador se añaden a la lista de metas pendientes, `(in person0 plane1)`, pero la precondición `(at plane1 city1)` no se añade porque es cierta en el estado actual de planificación.[2] El siguiente punto de decisión elige la meta `(in person0 plane1)` (nodo 8) para satisfacerla, seleccionando el operador instanciado `(board person0 plan1 city0)` (nodos 9 y 10). Sus precondiciones generan un nuevo nodo de tipo meta, `(at plane1 city0)` (nodo 12) y elige un operador instanciado para satisfacerla,`(fly plane1 city1 city0 fl3 fl2)` (nodos 14 y 15). En este punto, todas las precondiciones del operador instanciado son ciertas en el estado actual y PRODIGY puede aplicar el operador (búsqueda hacia adelante y opción por omisión) o continuar resolviendo otra meta pendiente. Cuando un operador se aplica, pasa a ser la siguiente acción del plan solución y el estado actual de planificación se cambia de acuerdo a los efectos del operador. Por tanto, PRODIGY aplica primero el operador `FLY` (nodo 16) y después el operador `BOARD` (nodo 17), generando las primeras acciones de la solución, `(fly plane1 city1 city0 fl3 fl2)` y `(board person0 plane1 city0)` (en la traza, cuando PRODIGY aplica un operador lo representa en mayúsculas). El algoritmo continúa alternando búsqueda hacia atrás con búsqueda hacia adelante (simulaciones de ejecución de planes) hasta que resuelve el problema.

Comparando los dos algoritmos de búsqueda se observan los puntos de decisión comunes de ambos planificadores:

- Decidir la meta a resolver primero. En cada punto de decisión de TGP, se seleccionan unas metas para resolver en ese instante y otras persisten (se resuelven en los siguientes niveles). PRODIGY, en los nodos de tipo meta, elige una meta para trabajar en ella.
- Elegir un operador instanciado para satisfacer una meta determinada.

Por ejemplo, en el primer punto de decisión de la figura 7.3, TGP persiste la meta `(at person1 city0)`, que es satisfecha en el nivel 2 (más próximo al estado inicial). PRODIGY elige la meta `(at person0 city1)` en primer lugar (nodo 5 de la figura 9.1) porque es la primera meta definida en el fichero del problema. Esto hace que PRODIGY seleccione, como primera acción del plan, volar el avión (inicialmente en *city1*) para recoger a *person0*. El plan generado por TGP es mejor porque aprovecha el vuelo del avión para traer a *person1*. Se puede generar una regla de control para que PRODIGY seleccione primero la meta que TGP persiste,

[2] Esto origina que la búsqueda de PRODIGY sea incompleta en los casos en que alguna precondición sea cierta en el estado actual pero posteriormente sea negada por la aplicación de algún operador [Fink and Blythe, 2005].

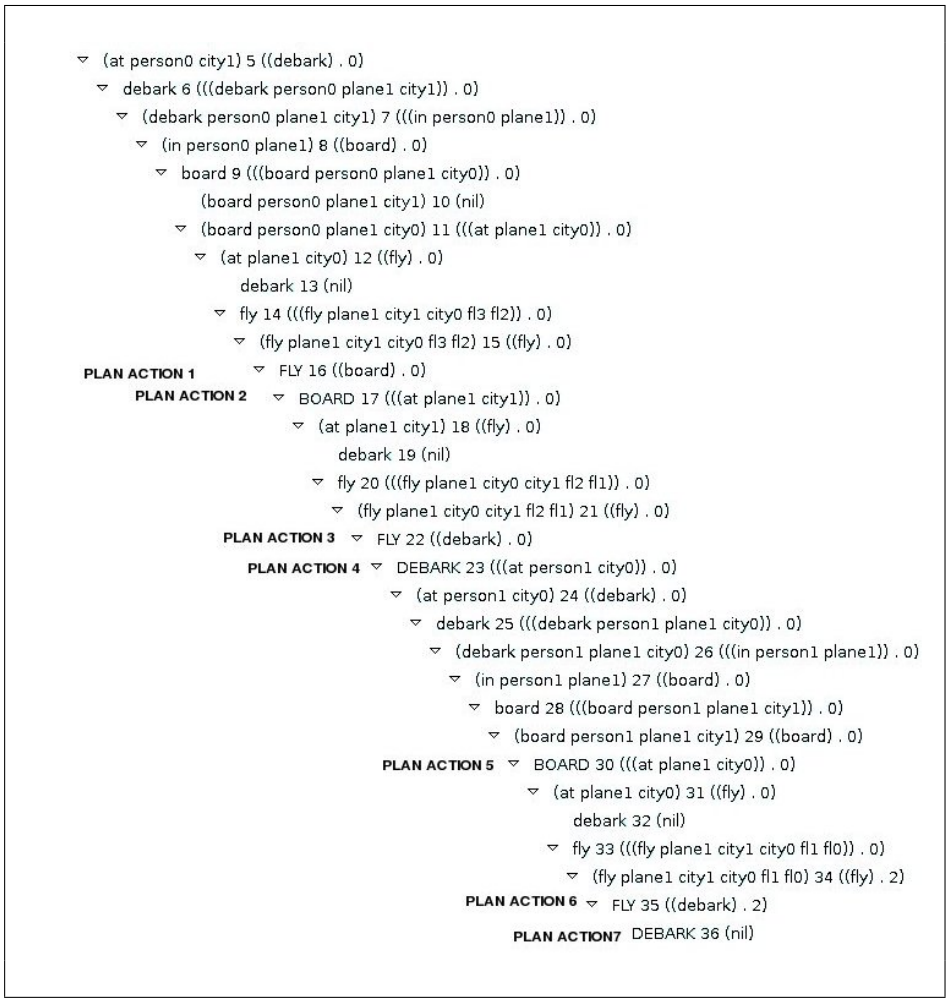

Figura 9.1: Ejemplo de árbol de búsqueda en PRODIGY.

cambiando el comportamiento por omisión de PRODIGY, para que empiece resolviendo la segunda meta y evitar así un vuelo innecesario del avión . Por otro lado, todos los puntos de decisión donde TGP selecciona un operador instanciado para satisfacer una meta tienen dos puntos de decisión equivalentes en PRODIGY: uno para seleccionar el operador y otro para seleccionar los valores de sus variables (*bindings*), excepto con la meta (`fuel-level plane1 fl1`) que no se introduce en la lista de metas pendientes de PRODIGY por ser cierta en el estado.

Por tanto, hay tres tipos de reglas que se pueden aprender en TGP para guiar la búsqueda de PRODIGY: reglas de `SELECT goals`, de `SELECT operators` y de `SELECT bindings`. En la siguiente sección, se explica cómo adaptar GEBL para aprender este conocimiento.

9.2. GEBL para transferir conocimiento

El proceso de aprendizaje seguido por GEBL es el descrito en la sección 7.3 pero con algunas modificaciones. Existen parámetros de ejecución del sistema que determinan su comportamiento según la aplicabilidad del conocimiento de control generado: para transferir a PRODIGY o para utilizarlo en TGP. Dichos parámetros son:

- Tipo de aprendizaje *lazy* o *eager*: para la transferencia conviene que sea *eager*, aprender de todos los puntos de decisión, ya que el comportamiento de ambos planificadores es diferente y la selección por omisión de uno no tiene por qué ser la misma que la del otro planificador. En aprendizaje *eager* se aprende tanto de las decisiones correctas por omisión del planificador como de las decisiones cuya opción por omisión no lleva a la solución. Por el contrario, en aprendizaje *lazy* sólo se aprende si la decisión por omisión del planificador no es la correcta.

- Procedimiento para hacer la regresión de metas. Como se explicó en el capítulo anterior, una de las dificultades mayores en GEBL es la falta de estados de planificación definidos durante las decisiones tomadas por el planificador en la búsqueda. En el caso de la transferencia, el problema se agrava porque dichos estados se tienen que verificar en PRODIGY cuando las reglas se ejecuten. No existe una equivalencia exacta entre los estados cuando TGP genera las reglas y cuando PRODIGY debería ejecutarlas; es preciso hacer algunas aproximaciones. Se ha optado por considerar dos aproximaciones diferentes que se explican en detalle en la siguiente sección.

- Tipo de reglas a aprender: para la transferencia hay que indicar al sistema que aprenda reglas de selección de metas y que no aprenda las reglas de elegir el operador `no-op` para conseguir una meta (reglas de persistencia). Las reglas de selección de operador se aprenden en ambos casos.

- Si el conocimiento generado va a ser transferido o no.

9.2.1. Regresión de metas y estados de planificación

El algoritmo de regresión de metas realizado por GEBL se explicaba en la sección 7.3.2. Necesita conocer el estado actual S. Al no haber estados definidos en planificación basada en grafos, se suponía que era siempre el mismo durante todo el proceso de búsqueda, el estado inicial del problema a resolver. Por tanto, éste constituye el primer modo para realizar la regresión de metas en GEBL.

En el caso de hacer la transferencia, el estado S debe representar el estado equivalente en PRODIGY. Tras el estudio detallado de los algoritmos de búsqueda realizados por ambos planificadores, no se observa que haya una equivalencia total de estados de planificación. TGP empieza la búsqueda para satisfacer las metas en el último nivel del grafo del plan (encadenamiento hacia atrás), y los primeros operadores que encuentra constituyen las últimas acciones del plan. Cuando persiste una meta está posponiendo la asignación del operador que la satisface a niveles más próximos al estado inicial y dicha acción aparecerá antes en la solución. Sin embargo, PRODIGY combina la búsqueda hacia atrás con la búsqueda hacia delante y, por omisión, intenta primero la búsqueda hacia adelante (aplicar los operadores instanciados). Los primeros operadores instanciados de la búsqueda constituyen las primeras acciones del plan solución. Parece adecuado hacer la siguiente suposición: el instante en que PRODIGY tomará la decisión de qué operador aplicar para resolver una meta de un problema resuelto también por TGP, las metas que TGP persiste, en ese punto de decisión, PRODIGY ya las habrá satisfecho, ya que se habrán aplicado los operadores pertinentes que las satisfacen y el estado de planificación estará modificado de forma acorde. Se ha observado experimentalmente que esta aproximación no siempre es correcta; parece más adecuada cuantas menos metas haya en el punto de decisión de TGP.

Por tanto, el segundo modo para hacer la regresión de metas en GEBL es haciendo esta aproximación. Para calcular dicho estado, se busca en la solución del problema (el proceso de aprendizaje empieza cuando el planificador encuentra un plan) el primer plan parcial en que todas las metas que persisten en el punto de decisión se satisfacen. Entonces, se progresa el estado inicial según los efectos de las acciones de dicho plan parcial. Es decir, se empieza con el estado inicial del problema y la primera acción del plan parcial. Se cambia el estado inicial según los efectos de dicha acción, borrando y añadiendo los literales de su lista de borrados y añadidos en la definición del operador en el dominio. Se aplica el mismo proceso con el estado resultante y la segunda acción del plan parcial. Se repite recursivamente hasta que no quedan acciones en el plan parcial.

Esta aproximación del estado equivalente de planificación en PRODIGY afecta también al argumento del meta-predicado `SOME-CANDIDATE-GOALS`. Si suponemos que las metas que persisten en el punto de decisión de TGP ya se habrán

resuelto en PRODIGY, las únicas metas pendientes serán a las que TGP asigna un operador en paralelo en ese nodo.

9.2.2. Generación de reglas

Para la transferencia, GEBL genera dos tipos de reglas:

1. Reglas de `SELECT goals`. En aquellos puntos de decisión en que TGP persiste una única meta, se genera una regla de `SELECT goals` con dicha meta para PRODIGY. Cuando se persisten más metas, en ese punto de decisión, no se puede asegurar cúal es la primera que se debe resolver y hacer una regla para que seleccione todas las metas que persiste es menos eficiente que seleccionar una única meta.

2. Reglas de `SELECT operators`. En un punto de decisión de TGP se crea una regla de éstas por cada una de las metas a las que se les asigna un operador en ese nivel del grafo.

De los dos primeros puntos de decisión del ejemplo de la figura 7.3, se generan dos reglas: una para seleccionar la meta `(at person1 city0)` y otra para seleccionar el operador `(debark person0 plane1 city1)` para conseguir la meta `(at person0 city1)`.

Las reglas de `SELECT goals` tienen los siguientes meta-predicados:

- `(TARGET-GOAL g1)`: `g1` es la meta que TGP persiste. La misma que selecciona la parte derecha de la regla.

- `(SOME-CANDIDATE-GOALS goals)`: resto de metas en el punto de decisión; serán sólo metas a las que TGP asigna un operador en ese nivel. Sólo se generan este tipo de reglas si TGP persiste una única meta en el punto de decisión. En el caso de persistir más de una meta, no se puede asegurar la primera meta que debe seleccionar PRODIGY y se ha optado por no generar ninguna regla.

- Un `TRUE-IN-STATE` por cada literal que sea cierto en el estado. Para calcularlo se hace la regresión de metas de `g1` y las metas de `goals`, suponiendo que el estado de planificación es el estado inicial del problema. La regla es para seleccionar una meta pendiente sobre la que trabajar (la meta que persiste); por tanto, no se puede suponer que PRODIGY ya la ha resuelto.

La figura 9.2, muestra la regla de `SELECT goals` del ejemplo anterior mencionado. Elige entre dos metas de desplazar personas de una ciudad a otra (los argumentos de los meta-predicados `TARGET-GOAL` y `SOME-CANDIDATE-GOALS`).

Una de las personas (variable <person1>) se encuentra en la misma ciudad donde hay un avión con suficiente nivel de combustible para volar. La regla selecciona la meta para desplazar dicha persona primero.

```
(control-rule regla-ZENO-TRAVEL-PZENO-s1
 (IF (AND (TARGET-GOAL (AT <PERSON1> <CITY0>))
          (TRUE-IN-STATE (AT <PERSON1> <CITY1>))
          (TRUE-IN-STATE (AT <PERSON0> <CITY0>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY1>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL2>))
          (TRUE-IN-STATE (AIRCRAFT <PLANE1>))
          (TRUE-IN-STATE (CITY <CITY0>))
          (TRUE-IN-STATE (CITY <CITY1>))
          (TRUE-IN-STATE (FLEVEL <FL1>))
          (TRUE-IN-STATE (FLEVEL <FL2>))
          (TRUE-IN-STATE (NEXT <FL1> <FL2>))
          (TRUE-IN-STATE (PERSON <PERSON0>))
          (TRUE-IN-STATE (PERSON <PERSON1>))
          (SOME-CANDIDATE-GOALS ((AT <PERSON0> <CITY1>)))))
 (THEN SELECT GOALS (AT <PERSON1> <CITY0>))
```

Figura 9.2: Ejemplo de regla de SELECT goals en el dominio Zenotravel aprendida por GEBL para transferir a PRODIGY.

Las reglas de SELECT operators son iguales a las descritas en la sección 7.3 salvo los meta-predicados TRUE-IN-STATE y los argumentos de los meta-predicados SOME-CANDIDATE-GOALS. La regresión de metas se calcula según el parámetro de GEBL establecido; por omisión, para la transferencia, es el segundo modo (suponiendo que las metas que persisten en el punto de decisión se habrán satisfecho). El argumento del meta-predicado SOME-CANDIDATE-GOALS es el resto de metas en el punto de decisión a las que TGP asigna un operador en ese nivel. No se incluyen aquéllas que TGP decidió persistir por la aproximación del estado equivalente en PRODIGY explicada anteriormente.

La figura 9.3 muestra la regla de SELECT operators generada en el ejemplo anterior, cuando se configuran los parámetros de GEBL para transferir conocimiento de control. Esta regla selecciona el operador debark para desplazar una persona de una ciudad a otra. PRODIGY intentaría (por omisión) desembarcar a la persona de cualquier avión definido en el problema, pero la regla escoge el avión mas conveniente: el que está en la misma ciudad que la persona y con suficiente nivel de combustible.

```
(control-rule rule-ZENO-TRAVEL-ZENO1-e1
 (IF (AND (CURRENT-GOAL (AT <PERSON0> <CITY1>))
          (TRUE-IN-STATE (AT <PERSON0> <CITY0>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY0>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL1>))
          (TRUE-IN-STATE (AIRCRAFT <PLANE1>))
          (TRUE-IN-STATE (CITY <CITY0>))
          (TRUE-IN-STATE (CITY <CITY1>))
          (TRUE-IN-STATE (FLEVEL <FL0>))
          (TRUE-IN-STATE (FLEVEL <FL1>))
          (TRUE-IN-STATE (NEXT <FL0> <FL1>))
          (TRUE-IN-STATE (PERSON <PERSON0>))
          (ALL-SOLVED-ACTIONS NIL)))
 (THEN SELECT OPERATORS (DEBARK <PERSON0> <PLANE1> <CITY1>))
```

Figura 9.3: Ejemplo de regla de `SELECT operators` en el dominio Zenotravel aprendida por GEBL para transferir a PRODIGY.

9.3. Traductor de reglas

Las reglas generadas anteriormente no se pueden usar directamente en PRODIGY. Es necesario construir un traductor para convertirlas a la sintaxis de PRODIGY. El traductor realiza las siguientes transformaciones:

- Divide las reglas de `SELECT operators` en dos: una para seleccionar el operador y otra para seleccionar los valores de sus variables (`bindings`).
- Traduce cada uno de los meta-predicados `TRUE-IN-STATE` referentes a tipos de variable, en un meta-predicado `TYPE-OF-OBJECT`. TGP no considera los tipos de las variables explícitamente; los representa como literales del estado inicial. Sin embargo, PRODIGY utiliza el tipo de cada variable en la definición del dominio igual que en *typed PDDL*.
- Quita los meta-predicados `ALL-SOLVED-ACTIONS` de las precondiciones de las reglas.
- Quita aquellas reglas en que el argumento de algún `TRUE-IN-STATE` sea igual al argumento de `CURRENT-GOAL`, `TARGET-GOAL` o de alguna de las metas pendientes de `SOME-CANDIDATE-GOALS`. Dada las aproximaciones seguidas para calcular el estado podría darse esta situación que imposibilitaría la ejecución de dicha regla. PRODIGY nunca puede tener como meta pendiente una meta que sea cierta en el estado actual.

Las reglas anteriores traducidas a la sintaxis de PRODIGY se muestran en la figura 9.4 y la figura 9.5.

```
(CONTROL-RULE REGLA-ZENO-TRAVEL-ZENO1-S1
 (IF (AND (TARGET-GOAL (AT <PERSON1> <CITY0>))
          (TRUE-IN-STATE (AT <PERSON1> <CITY1>))
          (TRUE-IN-STATE (AT <PERSON0> <CITY0>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY1>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL3>))
          (TRUE-IN-STATE (NEXT <FL2> <FL3>))
          (TRUE-IN-STATE (NEXT <FL1> <FL2>))
          (SOME-CANDIDATE-GOALS ((AT <PERSON0> <CITY1>)))
          (TYPE-OF-OBJECT <PERSON1> PERSON)
          (TYPE-OF-OBJECT <PERSON0> PERSON)
          (TYPE-OF-OBJECT <FL3> FLEVEL)
          (TYPE-OF-OBJECT <FL2> FLEVEL)
          (TYPE-OF-OBJECT <FL1> FLEVEL)
          (TYPE-OF-OBJECT <CITY1> CITY)
          (TYPE-OF-OBJECT <CITY0> CITY)
          (TYPE-OF-OBJECT <PLANE1> AIRCRAFT)))
 (THEN SELECT GOALS (AT <PERSON1> <CITY0>)))
```

Figura 9.4: Ejemplo de regla traducida a partir de la regla de `SELECT goals` de la figura 9.2.

9.4. Experimentos y resultados

La experimentación se ha hecho utilizando los dominios de las competiciones internacionales de planificación. Los primeros experimentos tienen como objetivo determinar qué dominios son los más apropiados para comprobar la utilidad de la transferencia de conocimiento. Posteriormente se han realizado pruebas más completas sobre dichos dominios.

Se quiere demostrar la posibilidad de transferir conocimiento de control del planificador TGP a PRODIGY. Para ello, se ejecuta GEBL con los parámetros adecuados para la transferencia (modo de aprendizaje *eager*, generación de reglas de selección de metas, el procedimiento seguido para hacer la regresión de metas y traducción de reglas), sobre un conjunto de entrenamiento de cada dominio. GEBL genera un conjunto de reglas que se traducen a la sintaxis de PRODIGY según se explicó en el apartado anterior. Dichas reglas se suministran a PRODIGY para que intente resolver los problemas de validación establecidos, con un tiempo máximo de resolución. Las variables independientes de la experimentación son:

- dominio de planificación,
- tipo de problemas: variando la complejidad, medida, por ejemplo, por el número de metas a satisfacer,
- métrica de calidad: número de operadores en el plan,

```
(CONTROL-RULE REGLA-ZENO-TRAVEL-ZENO1-E1
 (IF (AND (CURRENT-GOAL (AT <PERSON0> <CITY1>))
          (TRUE-IN-STATE (AT <PERSON0> <CITY0>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY0>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL1>))
          (TRUE-IN-STATE (NEXT <FL0> <FL1>))
          (TYPE-OF-OBJECT <PERSON0> PERSON)
          (TYPE-OF-OBJECT <FL1> FLEVEL)
          (TYPE-OF-OBJECT <FL0> FLEVEL)
          (TYPE-OF-OBJECT <CITY1> CITY)
          (TYPE-OF-OBJECT <CITY0> CITY)
          (TYPE-OF-OBJECT <PLANE1> AIRCRAFT)))
 (THEN SELECT OPERATORS DEBARK))

(CONTROL-RULE REGLA-ZENO-TRAVEL-ZENO1-E1-binds
 (IF (AND (CURRENT-OPERATOR DEBARK)
          (CURRENT-GOAL (AT <PERSON0> <CITY1>))
          (TRUE-IN-STATE (AT <PERSON0> <CITY0>))
          (TRUE-IN-STATE (AT <PLANE1> <CITY0>))
          (TRUE-IN-STATE (FUEL-LEVEL <PLANE1> <FL1>))
          (TRUE-IN-STATE (NEXT <FL0> <FL1>))
          (TYPE-OF-OBJECT <PERSON0> PERSON)
          (TYPE-OF-OBJECT <FL1> FLEVEL)
          (TYPE-OF-OBJECT <FL0> FLEVEL)
          (TYPE-OF-OBJECT <CITY1> CITY)
          (TYPE-OF-OBJECT <CITY0> CITY)
          (TYPE-OF-OBJECT <PLANE1> AIRCRAFT)))
 (THEN SELECT BINDINGS
  ((<P> . <PERSON0>) (<A> . <PLANE1>) (<C> . <CITY1>))))
```

Figura 9.5: Ejemplo de reglas traducidas a partir de la regla de `SELECT operators` de la figura 9.3.

- obtención o no de múltiples soluciones en el tiempo límite de ejecución establecido,

y las variables dependientes son:

- número y porcentaje de problemas de un mismo dominio resueltos en el tiempo límite de ejecución,

- tiempo de CPU que se tarda en encontrar la solución del problema,

- calidad de la solución encontrada, según la métrica de calidad establecida.

9.4.1. Selección de dominios

Los primeros experimentos se realizaron con los dominios y problemas de las competiciones utilizados en la sección 7.5. Es decir, los problemas de validación fueron los propuestos para las competiciones, utilizando los más sencillos para entrenar (en las tablas detalladas aparecen en negrita) y dejando un tiempo máximo de ejecución de 30s. Los resultados detallados están en el apéndice C.2 y la tabla 9.1 resume los resultados obtenidos. La tabla muestra el porcentaje de problemas resueltos por TGP, y por PRODIGY, sin conocimiento de control, y utilizando las reglas aprendidas por GEBL, junto con el número de reglas aprendidas. Estos resultados concuerdan con la conclusión deducida del estudio del estado del arte en planificación, de que no existe un planificador óptimo que destaque sobre todos los demás para todos los dominios y problemas. Los planificadores basados en grafos de planes son los que hoy en día, junto con los heurísticos, tienen una mayor aceptación entre la comunidad de planificación por su eficiencia. Sin embargo, en dos de los dominios estudiados, Satellite y Blocks, PRODIGY es más eficiente. En los dominios Depots, Rover y FreeCell, TGP no consiguió solucionar ninguno de los problemas de las competiciones y se descartaron para el aprendizaje. En el dominio Miconic, las reglas de GEBL mejoran considerablemente el rendimiento de PRODIGY (en un 89 %). En los dominios Satellite y Blocks, al utilizar el conocimiento aprendido por GEBL, PRODIGY resuelve los mismos problemas. Estudiando la calidad de las soluciones encontradas y el tiempo que tarda en encontrarlas, se deduce que la transferencia no mejora ninguno de estos dos valores, tal como se muestra en las tablas 9.2 y 9.3. Estas tablas representan el tiempo de CPU, en segundos, que tarda PRODIGY en encontrar la solución (T) y la calidad de los planes obtenidos (Q), medida como el número de operadores del plan. Los problemas en negrita son los utilizados para el aprendizaje. El tiempo de ejecución fue de 30s.

DOMINIO	TGP	PRODIGY	PRODIGY-GEBL	
				Reglas
Driverlog	45 %	0 %	5 %	46
Zenotravel	40 %	10 %	10 %	14
Logistics	13 %	0 %	0 %	100
Miconic	22 %	10 %	99 %	13
Satellite	20 %	100 %	100 %	26
Blocks	3 %	26 %	26 %	4

Tabla 9.1: Resumen resultados de transferencia de conocimiento en problemas de las IPCs. Porcentaje de problemas resueltos por cada sistema.

Por tanto, los experimentos más detallados se hicieron con los dominios Driverlog, Zenotravel, Logistics y Miconic porque la transferencia mejoró el rendimiento de PRODIGY, o al menos no lo empeoró. Los otros dos se descartaron porque originalmente PRODIGY se comporta mejor que TGP y la transferencia no consi-

guió mejorar ningún aspecto.

	PRODIGY		PRODIGY-GEBL	
Problema	Q	T	Q	T
STRIPS-SAT-X-1-p1	9	0.01	9	0.04
STRIPS-SAT-X-1-p2	17	0.03	17	0.06
STRIPS-SAT-X-1-p3	14	0.02	16	0.06
STRIPS-SAT-X-1-p4	23	0.04	23	0.11
STRIPS-SAT-X-1-p5	20	0.04	20	0.13
STRIPS-SAT-X-1-p6	20	0.04	20	0.10
STRIPS-SAT-X-1-p7	30	0.09	30	0.19
STRIPS-SAT-X-1-p8	26	0.06	26	0.21
STRIPS-SAT-X-1-p9	36	0.11	38	0.38
STRIPS-SAT-X-1-p10	36	0.12	36	0.31
STRIPS-SAT-X-1-p11	35	0.10	34	0.35
STRIPS-SAT-X-1-p12	55	0.23	55	0.57
STRIPS-SAT-X-1-p13	74	0.39	76	1.19
STRIPS-SAT-X-1-p14	76	0.36	76	0.85
STRIPS-SAT-X-1-p15	67	0.34	67	1.15
STRIPS-SAT-X-1-p16	60	0.32	66	1.44
STRIPS-SAT-X-1-p17	55	0.32	55	1.70
STRIPS-SAT-X-1-p18	61	0.25	61	0.46
STRIPS-SAT-X-1-p19	93	0.62	90	1.46
STRIPS-SAT-X-1-p20	203	2.36	203	2.64
TOTAL	1010	5.85	1016	13.40

Tabla 9.2: Resultados de transferencia de conocimiento en problemas de la IPC02, en el dominio Satellite.

	PRODIGY		PRODIGY-GEBL	
Problema	Q	T	Q	T
BLOCKS-4-0	10	0.01	10	0.01
BLOCKS-8-1	78	0.25	78	0.45
BLOCKS-6-2	38	0.09	38	0.09
BLOCKS-6-0	40	19.08	40	18.27
BLOCKS-4-2	14	2.46	14	2.23
BLOCKS-4-1	16	0.02	16	0.02
BLOCKS-12-1	372	6.81	372	7.36
BLOCKS-10-2	180	1.80	180	1.64
TOTAL	748	30.52	748	30.07

Tabla 9.3: Resultados transferencia de conocimiento en problemas de la IPC02, en el dominio Blocks

9.4.2. Experimentos detallados

El objetivo de estos experimentos es demostrar, primero, si la transferencia de conocimiento de control de TGP a PRODIGY es posible y útil; es decir, si puede mejorar la tarea de planificación de PRODIGY y, segundo, si puede mejorar a un sistema de aprendizaje diseñado para extraer conocimiento en ese planificador (HAMLET). Por último, se comparan los resultados con los producidos por TGP para verificar experimentalmente si PRODIGY, con las heurísticas aprendidas de TGP,

puede superarlo en alguno de los dominios estudiados.

Para hacer los experimentos, se entrenaron de forma separada HAMLET y GEBL con un conjunto de problemas de entrenamiento generados aleatoriamente y se probó sobre otro conjunto de problemas de validación, de dificultad creciente, generados aleatoriamente. Los problemas de entrenamiento deben ser pequeños para que los planificadores puedan resolverlos e iniciar los procesos de aprendizaje. Además, para la generalización de las reglas suele ser conveniente entrenar con problemas sencillos, sin demasiadas metas ni literales en el estado inicial tal como se apuntó en la sección 8.1. Las reglas generadas por GEBL se tradujeron a la sintaxis de PRODIGY y se *refinaron* con el módulo de refinamiento de HAMLET que sintácticamente borra aquellas reglas que sean más específicas que una regla más general (estén subsumidas por otra regla).

El número de problemas de entrenamiento y la complejidad de los de validación varió según la dificultad del dominio para PRODIGY. GEBL necesita menos problemas de entrenamiento que HAMLET porque realiza un aprendizaje *eager*, se aprende de todos los puntos de decisión, y no tiene un módulo inductivo para generalizar y refinar las reglas, con lo que se originan los problemas típicos de EBL (problema de la utilidad y generación de conocimiento demasiado específico). Estos problemas se pueden paliar utilizando pocos problemas de entrenamiento que no sean muy complejos. Para comparar los dos sistemas de aprendizaje se deberían utilizar los mismos problemas de entrenamiento, pero si se entrena HAMLET con los mismos problemas que GEBL se estaría limitando toda su potencia inductiva. Por eso, se ha optado por entrenar GEBL con subconjuntos de los problemas de entrenamiento utilizados en HAMLET. Cuando se han utilizado los mismos los resultados han sido mucho peores, por el problema de la utilidad. Se generan problemas aleatorios de entrenamiento para cada dominio que se estimen suficientes para el aprendizaje de HAMLET y se toma un subconjunto de ellos para entrenar GEBL de la siguiente manera:

- Zenotravel: se utilizaron 200 problemas de entrenamiento de 1 y 2 metas para entrenar HAMLET. GEBL se entrenó con los que resolvió TGP, 110 en total. Para la validación, se utilizaron 100 problemas entre 2 y 13 metas.
- Miconic: se utilizaron los problemas de la competición, 10 para entrenar HAMLET de 1 y 2 metas, 3 para entrenar GEBL de 1 meta y 140 para la validación entre 3 y 30 metas.
- Driverlog: se usaron 150 problemas de entrenamiento entre 2 y 4 metas en HAMLET y los de 2 metas para GEBL, 23 en total. Para la validación, se usaron 100 problemas entre 2 y 6 metas.
- Logistics: se utilizaron 400 problemas entre 1 y 3 metas en HAMLET y los primeros 200 problemas de 2 metas para GEBL. Para la validación, se usaron 100 problemas entre 1 y 5 metas.

Los problemas de entrenamiento constituyen un factor importante en los sistema de aprendizaje en planificación. Conviene realizar varios procesos de entrenamiento con diferentes conjuntos. En estos experimentos, se probaron con diferentes combinaciones de problemas, tanto por el tamaño del conjunto como por la dificultad de los problemas, y los mejores resultados se obtuvieron con los conjuntos explicados antes. El caso más extremo fue el del dominio Miconic en el que se entrenó con los mismos problemas de entrenamiento que HAMLET y las reglas resultantes hicieron que PRODIGY resolviera sólo el 25 % de los problemas de validación, frente al 99 % que consiguió con el conjunto de entrenamiento más sencillo.

Los resultados detallados obtenidos están en el apéndice C.2 y en la tabla 9.4 se presenta el resumen. Muestra el porcentaje de problemas resueltos (Res) de validación y las reglas obtenidas (Reglas) para cada uno de los sistemas. La última columna indica el número total de reglas generadas por GEBL antes de *refinarlas* por HAMLET. El tiempo de ejecución fue de 30s para todos los dominios menos el Miconic que fueron 60s. El tiempo máximo de resolución depende del tamaño de los problemas. Se ha verificado experimentalmente que aumentar dicho tiempo no mejora los resultados; si no se encuentra una solución en el tiempo establecido, la probabilidad de llegar a una solución en un tiempo varios órdenes de magnitud mayor es muy pequeña.

Dominio	TGP	PRODIGY	HAMLET		PRODIGY-GEBL		
	Res	Res	Res	Reglas	Res	Reglas	Todas
Zenotravel	1 %	37 %	40 %	5	98 %	14	459
Logistics	93 %	12 %	25 %	16	57 %	406	2347
Driverlog	100 %	26 %	4 %	9	77 %	71	325
Miconic	10 %	4 %	100 %	5	99 %	13	27

Tabla 9.4: Resultados del porcentaje de problemas aleatorios resueltos utilizando reglas aprendidas por GEBL, comparándolas con PRODIGY, HAMLET y TGP.

Los resultados demuestran que las reglas aprendidas por GEBL incrementan notablemente el porcentaje de problemas resueltos, respecto a PRODIGY solo y respecto a HAMLET, en todos los dominios menos en el Miconic en el que las reglas de HAMLET son ligeramente mejores. En dos de los dominios Miconic y Zenotravel, PRODIGY-GEBL superó al propio planificador TGP. En el dominio Zenotravel, TGP sólo resuelve un problema de validación porque en el resto de problemas los aviones no tienen suficiente combustible y es necesario aplicar el operador `refuel`. En estos casos, se ha observado que TGP no termina de generar el grafo del plan.

En el dominio Zenotravel, se generaron progresivamente problemas mas difíciles de validación, con el generador aleatorio de la competición, hasta conseguir un conjunto de 100 problemas en los que había 5 aviones, 10 ciudades y un número de personas variable entre 60 y 100. Generando problemas con un número de metas

entre 50 y 102. Dado que PRODIGY no considera metas pendientes que sean ciertas en el estado actual, se hizo un filtro para quitar aquellas metas de los problemas del conjunto que estuvieran en el estado inicial.

Utilizando las 14 reglas aprendidas con GEBL, PRODIGY resolvió el 64 % de ellos, dejando como límite de tiempo de ejecución 120s. La figura 9.6 muestra el tiempo de CPU, en segundos, y la calidad de los planes, en número de operadores en el plan. Por cada problema resuelto, hay dos puntos en la gráfica: uno representando el tiempo y otro la calidad. En el extremo, llega a resolver problemas de 102 metas en 53.16s y 576 operadores en la solución. Esto supone un enorme incremento, no sólo respecto a PRODIGY y HAMLET (únicamente resolvían problemas de 6 metas con 12 operadores en la solución), sino también respecto a TGP que no resolvió ninguno de estos problemas.

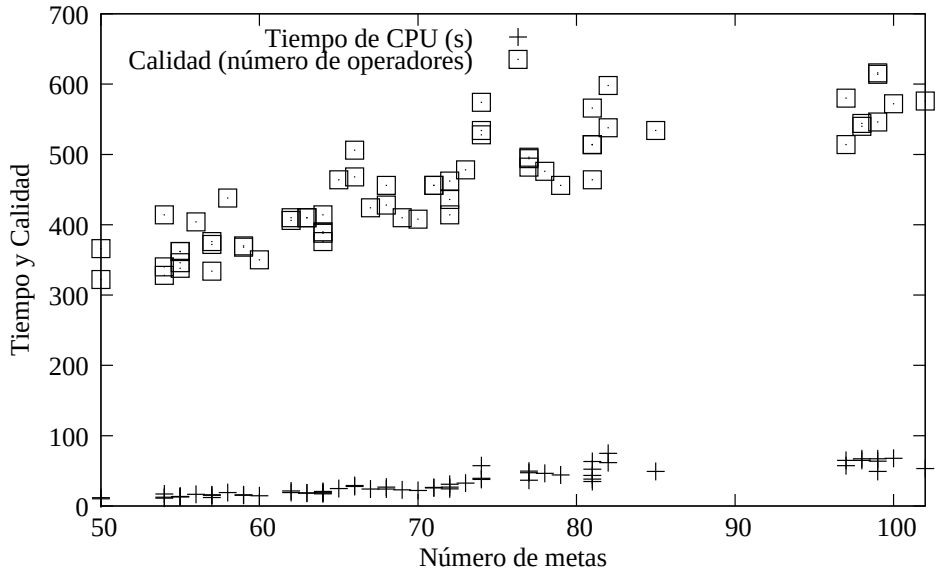

Figura 9.6: Resultados del cálculo del tiempo y la calidad en 100 problemas aleatorios entre 50 y 102 metas en el dominio Zenotravel resueltos por PRODIGY-GEBL.

Para comparar mejor el comportamiento de PRODIGY-GEBL respecto a TGP se utilizaron los 20 problemas de la competición. Se ejecutó PRODIGY con las 14 reglas aprendidas por GEBL durante 60s, obligando a PRODIGY a que buscara múltiples soluciones y la primera solución. Se dejó a TGP ese mismo tiempo para que resolviera los mismos problemas. La tabla 9.5 muestra el tiempo en segundos y la calidad de los planes, en número de operadores en el plan de cada sistema. La primera columna muestra los resultados de GEBL encontrando múltiples soluciones (GEBL-MS), la segunda columna cuando encuentra la primera solución (GEBL-PS) y la tercera columna muestra los resultados de TGP. Los resultados muestran que, respecto al tiempo (primera solución) y el número de problemas resueltos, GEBL es mejor que TGP. Sin embargo, las soluciones encontradas por TGP son de

mejor calidad. En los 8 problemas resueltos por todos, la calidad de los planes encontrados por GEBL es, en total, un 46 % peor que la calidad total de los planes encontrados por TGP. Al utilizar múltiples soluciones, GEBL sólo consigue mejorar la calidad en menos de un 3 %, que está muy desproporcionado con respecto al incremento del tiempo, más de 6 minutos y medio frente a los 0.58s. La razón de este comportamiento se explica en la siguiente sección al analizar las reglas aprendidas por GEBL.

En el dominio Miconic se hizo el mismo experimento, observándose un comportamiento similar, tal como muestra la tabla 9.6. GEBL resuelve un 89 % más de problemas que TGP pero la calidad de los problemas resueltos por ambos sistemas es peor: un 27 % en total encontrando la primera solución y un 19 % con múltiples soluciones.

	PRODIGY				TGP	
	GEBL-MS		GEBL-PS			
	Q	T	Q	T	Q	T
ZTRAVEL-1-2-P1	2	0.02	2	0.01	1	0.02
ZTRAVEL-1-3-P2	8	8.48	8	0.03	6	0.16
ZTRAVEL-2-4-P3	8	60.00	8	0.05	9	0.30
ZTRAVEL-2-5-P4	14	60.00	14	0.04	8	0.32
ZTRAVEL-2-4-P5	25	69.58	25	0.10	11	1.22
ZTRAVEL-2-5-P6	30	72.47	34	0.12	12	1.05
ZTRAVEL-3-6-P8	27	60.00	27	0.09	12	5.75
ZTRAVEL-3-7-P9	32	60.00	32	0.14	23	6.76
TOTAL	146	390.55	150	0.58	82	15.58
ZTRAVEL-2-6-P7	24	67.67	24	0.10	-	-
ZTRAVEL-5-25-P20	238	60.00	238	4.54	-	-
ZTRAVEL-5-25-P19	234	60.00	234	3.95	-	-
ZTRAVEL-5-20-P18	137	60.00	137	1.74	-	-
ZTRAVEL-5-20-P17	172	60.01	172	2.05	-	-
ZTRAVEL-5-15-P16	95	60.00	95	0.89	-	-
ZTRAVEL-5-15-P15	86	60.00	86	0.70	-	-
ZTRAVEL-5-10-P14	88	60.00	88	0.68	-	-
ZTRAVEL-3-10-P13	64	60.02	64	0.34	-	-
ZTRAVEL-3-8-P12	51	60.00	51	0.25	-	-
ZTRAVEL-3-7-P11	34	60.00	34	0.15	-	-
ZTRAVEL-3-8-P10	64	60.00	64	0.31	-	-

Tabla 9.5: Problemas de la competición en el dominio Zenotravel resueltos por PRODIGY-GEBL, con múltiples soluciones, y por TGP.

	GEBL-MS		GEBL-PS		TGP	
	Q	T	Q	T	Q	T
MIXED-F14-P7-R3	33	60.00	34	0.08	45.18	022
MIXED-F14-P7-R2	30	60.00	32	0.08	28.94	022
MIXED-F14-P7-R0	33	60.02	35	0.19	55.49	023
MIXED-F12-P6-R4	27	60.96	31	0.06	16.61	021
MIXED-F12-P6-R3	27	60.00	29	0.06	9.04	020
MIXED-F12-P6-R2	25	60.00	28	0.05	21.43	020
MIXED-F12-P6-R1	28	60.00	30	0.06	7.33	019
MIXED-F12-P6-R0	27	60.03	29	0.06	9.33	019
MIXED-F10-P5-R4	23	61.37	25	0.05	3.05	018
MIXED-F10-P5-R3	23	60.21	24	0.15	2.49	017
MIXED-F10-P5-R2	20	60.00	23	0.05	0.44	015
MIXED-F10-P5-R1	20	60.00	22	0.04	4.09	017
MIXED-F10-P5-R0	22	61.41	24	0.04	2.38	017
MIXED-F8-P4-R4	16	60.00	18	0.03	0.62	015
MIXED-F8-P4-R3	17	60.00	18	0.03	0.56	015
MIXED-F8-P4-R2	16	60.00	17	0.03	0.55	015
MIXED-F8-P4-R1	16	60.00	18	0.03	0.21	013
MIXED-F8-P4-R0	18	60.00	20	0.03	0.26	014
MIXED-F6-P3-R4	10	60.00	11	0.02	0.04	010
MIXED-F6-P3-R3	11	60.00	14	0.03	0.05	010
MIXED-F6-P3-R2	10	60.00	12	0.02	0.05	010
MIXED-F6-P3-R1	12	60.04	16	0.02	0.04	011
MIXED-F6-P3-R0	12	60.00	14	0.02	0.03	010
MIXED-F4-P2-R4	7	0.89	8	0.02	0.01	007
MIXED-F4-P2-R3	7	2.57	9	0.02	0.02	007
MIXED-F4-P2-R2	7	0.38	7	0.01	0.01	007
MIXED-F4-P2-R1	7	0.38	8	0.01	0.01	007
MIXED-F4-P2-R0	7	2.85	10	0.01	0.01	007
MIXED-F2-P1-R4	4	0.01	4	0.01	0.00	004
MIXED-F2-P1-R3	4	0.01	4	0.01	0.01	004
MIXED-F2-P1-R2	4	0.01	4	0.01	0.00	004
MIXED-F2-P1-R1	3	0.01	3	0.02	0.00	003
MIXED-F2-P1-R0	4	0.01	4	0.00	0.01	004
TOTAL	530	1391.16	585	1.35	208.29	427

Tabla 9.6: Problemas de la competición en el dominio Miconic resueltos por GEBL, con múltiples soluciones, y por TGP.

9.5. Análisis del conocimiento generado por GEBL

9.5.1. Dominio Zenotravel

Todas las reglas aprendidas por GEBL se listan en el apéndice B.1. GEBL genera una o dos reglas para seleccionar los *bindings* más apropiados para cada uno de los operadores del dominio. Para el operador `debark` genera dos reglas: una cuando la persona a desembarcar se encuentra en una ciudad donde hay un avión, y otra cuando la persona está dentro de un avión. En ambos casos, las reglas seleccionan el avión más apropiado, única variable libre en dicho operador: el avión que esté en la misma ciudad que la persona a desplazar, con combustible suficiente; y el avión en el que está la persona, respectivamente. Para el operador `board` se generan dos reglas prácticamente iguales, sólo difieren en los argumentos del meta-predicado `SOME-CANDIDATE-GOALS`. En estas reglas, la selección de la única variable libre, el avión, también es la más adecuada. Lo que no es correcto son las metas pendientes. En las dos considera que debe haber otra meta pendiente de alcanzar un nivel de combustible para el avión y en la segunda regla que, además, haya una segunda persona dentro del mismo avión. No son necesarias ninguna de estas condiciones. Las reglas para seleccionar los *bindings* de los operadores `fly` y `refuel` seleccionan las variables más adecuados en cada caso. También se genera una regla para seleccionar cada uno de los operadores del dominio, paralela a las reglas de *bindings* anteriores. La más interesante es la que se muestra en la figura 9.7 que indica que hay que utilizar el operador `refuel` para alcanzar la meta `(fuel-level <plane1><fl1>)` cuando el avión tiene un nivel de combustible <`fl0`>. La mayor dificultad de este dominio es aplicar el operador `refuel` correctamente. GEBL es capaz de aprender este conocimiento. El problema que tiene es que, tal como lo aprende, se aplica en más ocasiones de las estrictamente necesarias, dado que la única precondición que tiene es que haya un nivel posterior de combustible. Esto es lo que está originando la peor calidad de los planes generados por PRODIGY cuando utiliza las reglas de GEBL. Esta regla semánticamente es correcta porque selecciona el operador `refuel` cuando el avión tiene el nivel mas bajo de combustible (`TRUE-IN-STATE (fuel-level` <`plane1`> <`fl0`>)), pero, sintácticamente, no hay ningún meta-predicado que indique que <`fl0`> sea el nivel inferior.

Un problema parecido presenta la última regla aprendida de selección de meta: hay tres metas pendientes, dos para desplazar a personas diferentes a la misma ciudad y una tercera para desplazar un avión a esa misma ciudad, cuando tiene un nivel de combustible <fl0>. Las personas y el avión se encuentran originalmente en la misma ciudad. La regla selecciona la meta de desplazar al avión primero, pero en ningún momento se especifica que <fl0> sea el nivel más bajo de combustible. En ese caso, sí podría ser mejor empezar por satisfacer la meta de desplazar el avión para que haga el `refuel` y pueda desplazar a las personas. Por último, la otra regla que aprende de selección de metas es la representada en la figura 9.4 de

la que ya se comentó su utilidad (seleccionaba la meta de desplazar a una persona que estuviera en la misma ciudad en que hubiera un avión).

```
(control-rule regla-zeno-travel-l200-03-e2
  (if (and (current-goal (fuel-level <plane1> <fl1>))
           (true-in-state (next <fl0> <fl1>))
           (true-in-state (fuel-level <plane1> <fl0>))
           (true-in-state (at <plane1> <city1>))
           (type-of-object <plane1> aircraft)
           (type-of-object <city1> city)
           (type-of-object <fl0> flevel)
           (type-of-object <fl1> flevel)))
  (then select operators refuel))
```

Figura 9.7: Regla para seleccionar el operador `refuel` aprendida en GEBL en el dominio Zenotravel.

9.5.2. Dominio Driverlog

En el dominio Driverlog las reglas generadas por GEBL, tras el refinamiento, son:

- 13 reglas de selección de *bindings* del operador `unload-truck`
- 3 reglas de selección de *bindings* del operador `board-truck`
- 3 reglas de selección de *bindings* del operador `disembark-truck`
- 5 reglas de selección de *bindings* del operador `drive-truck`
- 4 reglas de selección de *bindings* del operador `walk`
- 1 regla de selección de *bindings* del operador `load-truck`
- 29 reglas para seleccionar operador: las mismas de cada operador que las de *bindings* correspondientes
- 13 reglas de selección de metas: 2 para seleccionar metas de tipo `(at <truck><s>)`, 2 de tipo `(in <package><truck>)` y 9 de tipo `(at <package><s>)`.

Al hacer búsqueda hacia atrás, el conocimiento más importante en este dominio es aplicar el operador `unload-truck` cuando la meta es llevar un paquete a un sitio. Las 13 reglas aprendidas se listan en el apéndice B.2. En ellas se observa que GEBL aprende las variables correctas para diferentes situaciones. Suponiendo que la meta actual sea llevar un paquete a $s0$, la primera regla describe un estado en el

que el paquete está dentro de un camión que se encuentra en $s0$. La segunda regla, el paquete está en otro sitio donde hay un camión conducido por un conductor y hay un `link` entre ambos sitios. La tercera, es igual a la anterior, pero el paquete está dentro del camión. En la cuarta, el camión no tiene conductor y el conductor se encuentra en el mismo sitio que el camión. En las siguientes reglas el conductor se encuentra en un sitio diferente al del camión existiendo caminos entre ambos. En la quinta, el paquete y el conductor están en un mismo sitio y el camión en $s0$. En la sexta, el conductor está en $s0$ y el camión y el paquete en el mismo sitio. En la séptima, el conductor y el camión están en el mismo sitio, el paquete en un tercer sitio y hay *links* entre los dos sitios. Esta regla añade la precondición innecesaria de que el mismo conductor está en $s0$. La octava es igual que la anterior, pero el conductor y el camión están en el mismo sitio. En la novena, el camión y el paquete están juntos y el conductor está en un tercer sitio. Necesita andar por dos caminos hasta llegar al camión. Las siguientes reglas son parecidas a alguna de las anteriores, pero añadiendo metas pendientes. Las reglas cubren casi todos los casos referentes a la localización inicial del camión, el conductor y el paquete interviniendo hasta tres sitios diferentes, pero algunas de estas reglas se podrían generalizar o especializar a través de algún mecanismo de inducción creando un conjunto más reducido de reglas, como hace HAMLET.

Las reglas de selección de metas se muestran en el apéndice B.2. Las cuatro primeras seleccionan entre dos metas de llevar paquetes a distintos sitios que se encuentran en otro sitio. Eligen empezar por el paquete en el que hay un camino entre el origen y el destino. Difieren en la situación original del conductor, el camión y los paquetes. Si hay un camión en la misma ciudad que uno de los paquetes a desplazar seleccionan empezar por esa meta. Las dos siguientes reglas eligen empezar por metas para desplazar paquetes antes que conductores que siempre es una buena estrategia en este dominio. En la regla 12, hay un paquete, un camión y un conductor en el mismo sitio y dos metas una de llevar el camión a un segundo sitio y otra de llevar el paquete al mismo sitio. La regla elige la meta del camión, lo cual no parece lo más conveniente; es preferible transportar el paquete a la vez que el camión. La siguiente regla describe una situación igual, pero las metas entre las que puede elegir son la de que el paquete esté dentro del camión o la de desplazar el camión. En este caso selecciona la opción adecuada, empezar por la meta del paquete. PRODIGY dispara todas las reglas cuyas precondiciones se cumplen en cada punto de decisión. Aunque alguna de las reglas sea incorrecta, si hay otras que son correctas, el comportamiento final de PRODIGY será correcto.

9.5.3. Dominio Logistics

En este dominio ocurre algo similar al dominio anterior. Se generan varías reglas para seleccionar los *bindings* de cada uno de los operadores del dominio: 48 para el operador `unload-truck`, 39 para el operador `load-truck`, 23 para

el operador `load-airplane`, 14 para el operador `unload-airplane`, 5 para el operador `drive-truck` y 4 para el operador `fly-airplane`. También se generan 133 reglas para seleccionar operadores y 140 de selección de metas.

Las reglas para seleccionar los bindings del operador `unload-truck` para conseguir la meta de que un paquete esté en un sitio, representan diferentes situaciones, desde que el paquete esté dentro de un camión en la localidad destino hasta que el paquete se encuentre en un sitio de diferente ciudad y es necesario llevarlo en avión. Muchas de las reglas se podrían agrupar en una sola más general. También hay reglas iguales donde varían las metas pendientes que podrían generalizarse.

Las reglas de selección de metas eligen la meta más adecuada cuando la selección es entre dos metas pendientes (108 reglas). Por ejemplo, si tiene que elegir entre dos metas de llevar paquetes a un sitio selecciona primero la meta en la que el paquete está en la ciudad destino, bien dentro de un camión, de un avión o en un sitio de la misma ciudad. Sin embargo, cuando intervienen 3 metas (32 reglas) la opción elegida por la regla no es siempre la más correcta. Por ejemplo, la regla de la figura 9.8 representa una situación en que hay tres metas pendientes: que un objeto, inicialmente dentro de una camión en una ciudad `c1`, esté dentro de un avión que inicialmente está en otra ciudad `c0`; que ese mismo avión esté en el aeropuerto de la ciudad `c1`; y que un segundo objeto `obj0`, inicialmente dentro de un camión que está en el aeropuerto donde está el avión, esté en el otro aeropuerto de la ciudad `c1`. La regla elige empezar por la meta de desplazar el avión a la otra ciudad y sería más eficiente que antes cargara `obj0` ya que está en la misma ciudad que el avión; es decir, que comenzara por la meta `(at obj0 a1)`. El meta-predicado `some-candidate-goals` se cumple si, al menos, una de las metas de su argumento es una meta pendiente; no es necesario que lo sean todas. Por tanto, si la única meta pendiente es que `obj0` esté dentro del avión, la regla si es correcta.

9.5.4. Dominio Miconic

En este dominio, GEBL genera 13 reglas que resuelven todos los problemas menos uno. HAMLET aprende sólo 5 reglas que son capaces de resolver todos los problemas. Por tanto, para este dominio el aprendizaje realizado por HAMLET es suficiente para resolver los problemas y no es necesario recurrir a la transferencia de conocimiento. De las 5 reglas que aprende HAMLET 2 son de `sub-goal` y las otras 3 GEBL aprende la misma regla u otra muy similar como se muestra en las figuras 9.9, 9.10 y 9.11 (la primera regla es la aprendida por GEBL). Entre las reglas que seleccionan el operador `down`, la regla de HAMLET es más general que la de GEBL ya que tiene un meta-predicado `true-in-state` menos que indica la situación del ascensor. En las otras reglas, son más específicas las que aprende HAMLET ya que añade otra precondición de que exista una meta pendiente más en cada caso. De hecho, las reglas aprendidas por HAMLET para seleccionar las instanciaciones de variables de los operadores `up` y `down` utilizan la palabra

```
(control-rule regla-logistics-aips98-l1-15-s2
  (if (and (target-goal (at <pl0> <a1>))
           (true-in-state (loc-at <a1> <c1>))
           (true-in-state (loc-at <po1> <c1>))
           (true-in-state (inside <ob0> <tr0>))
           (true-in-state (inside <ob1> <tr1>))
           (true-in-state (at <pl0> <a0>))
           (true-in-state (at <tr0> <a0>))
           (true-in-state (at <tr1> <po1>))
           (some-candidate-goals ((inside <ob1> <pl0>)
                                  (at <ob0> <a1>)))
           (type-of-object <pl0> airplane)
           (type-of-object <a0> airport)
           (type-of-object <a1> airport)
           (type-of-object <c1> city)
           (type-of-object <ob0> object)
           (type-of-object <ob1> object)
           (type-of-object <po1> post-office)
           (type-of-object <tr0> truck)
           (type-of-object <tr1> truck)))
  (then select goals (at <pl0> <a1>)))
```

Figura 9.8: Regla de selección de metas en el dominio Logistics.

`rededuced` en el nombre, que indica que al principio por EBL se generó una regla sobre-general.

9.6. Conclusiones y trabajos futuros

En esta sección se ha estudiado la viabilidad de transferir conocimiento de control aprendido en un planificador basado en grafos de planes, TGP, a un planificador bidireccional, PRODIGY. Se han estudiado las decisiones comunes hechas por ambos planificadores, selección de la próxima meta a resolver y selección del operador para alcanzar una meta, generándose reglas de control que explican las circunstancias en las que dichas decisiones se toman en TGP para aplicarlas a PRODIGY en situaciones similares. Se ha adaptado el sistema de aprendizaje GEBL para generar este conocimiento y traducirlo a la sintaxis de PRODIGY. El objetivo es aprender heurísticas de TGP que mejoren la tarea de planificación de PRODIGY. Tal como se dijo en la introducción, PRODIGY presenta una serie de ventajas frente a TGP que hacen que en algunos casos pueda resultar preferible su utilización, siendo muy interesante incrementar su eficiencia durante la resolución de problemas. Además, desde un punto de vista científico, la transferencia de conocimiento entre planificadores tiene interés por sí misma y hasta el momento, no existía ninguna

```
(control-rule GEBL-REGLA-MICONIC-MIXED-F2-R2-E2
  (if (and (current-goal (lift-at <f0>))
           (true-in-state (above <f0> <f1>))
           (true-in-state (lift-at <f1>))
           (type-of-object <f0> floor)
           (type-of-object <f1> floor)))
(then select operators down))

(control-rule HAMLET-INDUCED-SELECT-DOWN-L009-02
  (if (and (current-goal (lift-at <f2>))
           (true-in-state (above <f2> <f1>))
           (type-of-object <f2> floor)
           (type-of-object <f1> floor)))
(then select operators down))
```

Figura 9.9: Reglas similares aprendidas por GEBL (1ª regla) y HAMLET (2ª regla) en el dominio Miconic para seleccionar el operador down.

```
(control-rule GEBL-REGLA-MICONIC-MIXED-F2-R2-E4-BINDS
  (if (and (current-operator up)
           (current-goal (lift-at <f1>))
           (true-in-state (lift-at <f0>))
           (true-in-state (above <f0> <f1>))
           (type-of-object <f1> floor)
           (type-of-object <f0> floor)))
  (then select bindings
        ((<f1> . <f0>) (<f2> . <f1>))))

(control-rule HAMLET-REDEDUCED-SELECT-BIND-UP-L010-02
  (if (and (current-operator up)
           (current-goal (lift-at <f1>))
           (true-in-state (above <f2> <f1>))
           (true-in-state (above <f2> <f3>))
           (true-in-state (lift-at <f3>))
           (true-in-state (origin <p4> <f2>))
           (some-candidate-goals ((boarded <p4>)))
           (type-of-object <p4> passenger)
           (type-of-object <f3> floor)
           (type-of-object <f2> floor)
           (type-of-object <f1> floor)))
  (then select bindings
        ((<f1> . <f2>) (<f2> . <f1>))))
```

Figura 9.10: Reglas similares aprendidas por GEBL (1ª regla) y HAMLET (2ª regla) en el dominio Miconic para seleccionar los *bindings* del operador up.

```
(control-rule GEBL-REGLA-MICONIC-MIXED-F2-R2-E2-BINDS
  (if (and (current-operator down)
           (current-goal (lift-at <f0>))
           (true-in-state (above <f0> <f1>))
           (true-in-state (lift-at <f1>))
           (type-of-object <f0> floor)
           (type-of-object <f1> floor)))
  (then select bindings
        ((<f1> . <f1>) (<f2> . <f0>))))

(control-rule HAMLET-REDEDUCED-SELECT-BIND-DOWN-L009-02
  (if (and (current-operator down)
           (current-goal (lift-at <f1>))
           (true-in-state (above <f1> <f2>))
           (true-in-state (above <f3> <f2>))
           (true-in-state (lift-at <f3>))
           (true-in-state (origin <p4> <f2>))
           (some-candidate-goals ((boarded <p4>)))
           (type-of-object <p4> passenger)
           (type-of-object <f3> floor)
           (type-of-object <f2> floor)
           (type-of-object <f1> floor)))
  (then select bindings
        ((<f1> . <f2>) (<f2> . <f1>))))
```

Figura 9.11: Reglas similares aprendidas por GEBL (1ª regla) y HAMLET (2ª regla) en el dominio Miconic para seleccionar los *bindings* del operador down.

investigación al respecto. De todas formas, como segundo objetivo, se busca verificar experimentalmente si PRODIGY, con las heurísticas aprendidas de TGP, tiene un comportamiento mejor que TGP en algún dominio.

Los experimentos demuestran que la transferencia es factible. En los cuatro dominios estudiados, GEBL aprende un conocimiento del paradigma de planificación basado en grafos, que incrementa la eficiencia de otro paradigma de planificación distinto. En dos de los dominios, Miconic y Zenotravel, el conocimiento aprendido incluso mejoró el tiempo de resolución en encontrar la solución y el número de problemas resueltos con respecto a TGP. Sin embargo, en los problemas que resuelven ambos, TGP y PRODIGY con las reglas aprendidas en GEBL, se observa un empeoramiento de la calidad de las soluciones. El sistema aprende automáticamente, en el dominio Zenotravel, el conocimiento de control necesario para resolver casi todos los problemas. En este caso, se trata sólo de aprender a repostar los aviones cuando no tiene combustible suficiente. Los otros dominios estudiados (Logistics, Driverlog y Miconic) son más complejos y GEBL no es capaz de extraer todo el conocimiento necesario para resolver problemas de mucha mayor complejidad, aunque si incrementa notablemente la eficiencia de PRODIGY.

La mayor dificultad de la transferencia vuelve a ser encontrar los estados de planificación, igual que ocurrió al utilizar GEBL para incrementar la eficiencia del propio planificador TGP. Ahora, los estados de planificación durante las decisiones tomadas por TGP deben corresponderse con estados equivalentes cuando se realicen esas mismas selecciones en PRODIGY. No existe dicha equivalencia por lo que ha sido necesario hacer algunas aproximaciones que en algunos casos son más acertadas que en otros. Las suposiciones adoptadas para obtener los estados de planificación se implementan en el algoritmo de regresión de metas, y por eso se ha añadido un parámetro a GEBL que permita posibilidades variadas. En un futuro, se podrían estudiar otras formas de calcular la regresión y verificar experimentalmente si mejoran los resultados.

Aunque el artículo original donde se explica el planificador TGP, se afirma que ordenan las metas en orden decreciente, igual que los CSP, en la práctica no lo implementan porque dicen que se obtienen mejores resultados con el orden inverso. Al final, optan por no utilizar ninguna ordenación y los experimentos aquí realizados han mantenido su criterio. Sería interesante probar distintas opciones de ordenación de las metas y estudiar cómo afecta al conocimiento transferido. Para generar las reglas de selección de metas es posible que sea mejor ordenar las metas en orden creciente, primero las que se encuentran más cerca del estado inicial. Estas reglas seleccionan la meta que TGP persiste para solucionar primero. Dado que la persistencia es la opción primera intentada por TGP, a PRODIGY le interesa empezar resolviendo las metas que menos cambios requieren del estado. Sin embargo, en los CSP se ha demostrado que es más eficiente empezar a resolver las metas más difíciles. Otra posible opción es probar la ordenación en orden decreciente pero modificando el comportamiento de TGP para que intente aplicar

operadores para resolver las metas antes que persistirlas. O probar con diferentes combinaciones entre ordenaciones de metas y persistir/aplicar operadores.

Por último, como trabajos futuros también sería conveniente aplicar inducción en las reglas generadas por GEBL para generalizarlas y especializarlas de forma similar a HAMLET y estudiar la transferencia de conocimiento entre más planificadores.

Capítulo 10

Transferencia de conocimiento a los sistemas de aprendizaje

El objetivo de esta parte de la tesis es estudiar experimentalmente cómo influye la utilización de conocimiento previo en los sistemas de aprendizaje en planificación. Esto es, determinar si la utilización de algún tipo de conocimiento externo al propio sistema de aprendizaje consigue mejorar su rendimiento; es decir, hace que el sistema obtenga un conocimiento de control que incremente la eficiencia del planificador. Hay que determinar de qué fuentes se puede obtener el conocimiento inicial y cómo integrarlo en los sistemas de aprendizaje. En particular se han estudiado tres posibles fuentes: de una persona, de otro planificador y de otro sistema de aprendizaje diferente. Los sistemas de aprendizaje estudiados son HAMLET y EVOCK que utilizan técnicas de aprendizaje muy diferentes, EBL más inducción en el primero y programación genética en el segundo (ver capítulo 4). GEBL se ha utilizado sólo como fuente de aprendizaje para HAMLET y no como sistema de aprendizaje a estudiar, ya que como se vio en el capítulo anterior, no incrementa la eficiencia del planificador TGP. HEBL no se ha utilizado, en esta parte de la tesis, porque el tipo de dominios para el que está diseñado difiere de los dominios de planificación clásicos utilizados por el resto de sistemas.

El capítulo está organizado de la siguiente manera: primero se describe cómo incorporar conocimiento previo a HAMLET, luego a EVOCK, después se describen los experimentos realizados y por último, las conclusiones obtenidas.

10.1. Conocimiento previo en HAMLET

La primera condición necesaria para que HAMLET pueda generar las reglas de control es que encuentre la solución a un problema y almacene el árbol de búsqueda generado durante el proceso. En algunos dominios, excesivamente difíciles, no es

posible encontrar estas soluciones, ni siquiera para problemas sencillos, lo que imposibilita el aprendizaje de ningún conocimiento. Necesita de una ayuda extra, conocimiento previo, que le permita resolver algún problema.

Se han seguido dos métodos para suministrar conocimiento previo a HAMLET. El primero consiste en proporcionarle un conjunto inicial de reglas de control, bien generadas manualmente por una persona o bien mediante algún proceso de aprendizaje previo, realizado con alguna otra técnica de aprendizaje. El segundo método es recibiendo la solución de un problema alcanzada por otro planificador. Si HAMLET no encuentra la solución al problema en un tiempo razonable se recurre a otro planificador para que la encuentre y ayude a HAMLET.

10.1.1. Conjunto inicial de reglas de control

Cuando se utiliza un conjunto inicial de reglas, HAMLET puede generalizar o quitar algunas de las reglas suministradas pero no puede especializarlas al no tener información sobre el episodio de aprendizaje que las produjo. Pero, puede utilizar este conocimiento previo para resolver problemas de aprendizaje que sin conocimiento de control no era capaz de resolver e iniciar nuevos procesos de aprendizaje completos incluida la especialización.

Como se ha dicho, pueden ser generadas manualmente por un humano o automáticamente por otros sistema de aprendizaje. En la generación manual, un factor determinante es el grado de cualificación de la persona que diseña las reglas, variando desde un experto en planificación hasta alguien novel en el tema. En esta tesis sólo se ha recurrido a expertos.

Cuando se utiliza otra técnica de aprendizaje, puede ser preciso un proceso de adaptación previo del conocimiento extraído para que sea entendido por HAMLET. En particular, se ha utilizado el conocimiento extraído por GEBL y por EVOCK. Con GEBL se tiene el traductor que adapta la sintaxis a la de HAMLET, tal como se explica en la sección 9.3, pero con EVOCK no siempre es factible dicha adaptación.

10.1.2. Utilizando otro planificador

Se ha utilizado el planificador FF para proporcionar las soluciones de los problemas que HAMLET no puede resolver, con el fin de que le sirvan de ayuda para encontrar la soluciones y generar los árboles de búsqueda que precisa para identificar los puntos de decisión, los nodos de éxito y fallo, de los que obtener las reglas de control.

La aproximación es genérica y se podría usar con cualquier planificador. La generación de las instancias de aprendizaje se realiza en dos pasos:

- En el primer paso, FF recibe el problema de planificación que no pudo re-

solver HAMLET y trata de encontrar una solución. Normalmente, para el aprendizaje es suficiente con que se llegue a una solución, aunque no sea la mejor posible en términos de la métrica de calidad (por ejemplo, longitud del plan). El problema ahora, consiste en cómo esa solución puede proporcionar a HAMLET las instancias de aprendizaje que requiere.

- El segundo paso consiste en generar artificialmente un árbol de búsqueda de la solución de FF. Esto no es un tarea trivial ya que lo único que proporciona la solución es un conjunto de operadores instanciados que se deben ejecutar en secuencia. Hay muchos posibles árboles de los que se podría obtener la misma solución. Se ha optado por generar un conjunto de reglas de control, independientemente de las que se van a aprender, que seleccionan como nodos válidos de la búsqueda nueva aquéllos que utilizan alguno de los operadores instanciados en la solución de FF. Siguiendo la sintaxis de PRODIGY esto da lugar a dos tipos de reglas:

  - *SELECT operator*: se generan reglas de seleccionar operador para los operadores que conforman la solución de FF de la siguiente manera: por cada efecto del operador, según su definición en el dominio, se crea una regla cuya meta actual de resolución sea dicho efecto y el consecuente sea seleccionar el operador. Si el plan de FF contuviera todas las acciones definidas en el dominio estas reglas no aportarían nada al proceso de búsqueda ya que para cada meta seguiría considerando todos los operadores relevantes (los que obtienen dicha meta).
  - *SELECT bindings*: para cada operador O que aparezca en la solución de FF genera una regla para seleccionar todos los *bindings* de sus variables que se correspondan con las instancias de O en la solución. Por ejemplo, si una solución del dominio Logistics es:

    load-airplane(package1,airplane1,airport1)
    fly-airplane(airplane1,city1,city2)
    unload-airplane(package1,airplane1,airport2)
    fly-airplane(airplane1,city2,city3)

    se genera una regla de control para seleccionar los siguientes *bindings* del operador `fly-airplane`:

    ((<airplane> . airplane1) (<city-from> . city1) (<city-to> . city2))
    ((<airplane> . airplane1) (<city-from> . city2) (<city-to> . city3))

En este ejemplo se generaría una regla para seleccionar el operador `load-airplane`, otra para seleccionar el operador `unload-airplane` y otra para seleccionar el operador `fly-airplane` ya que estos operadores sólo tienen un efecto cada uno. Es decir, de los cinco posibles operadores del dominio PRODIGY restringiría la búsqueda a tres de ellos, con los que FF resolvió el problema. También, se generarían dos reglas más de selección de los *bindings* del operador `load-airplane` y `unload-airplane` que se corresponden con la

instanciación de esos operadores en la solución de FF, además de la regla de selección de los *bindings* del operador `fly-airplane` comentada.

Utilizando este conjunto de reglas, PRODIGY es capaz de encontrar la solución al problema y generar el árbol de búsqueda correspondiente. Después, PRODIGY continúa buscando otras soluciones para que HAMLET pueda generar reglas de control de las decisiones que llevaron a soluciones mejores durante el límite de tiempo establecido. La figura 10.1 muestra un esquema de esta aproximación.

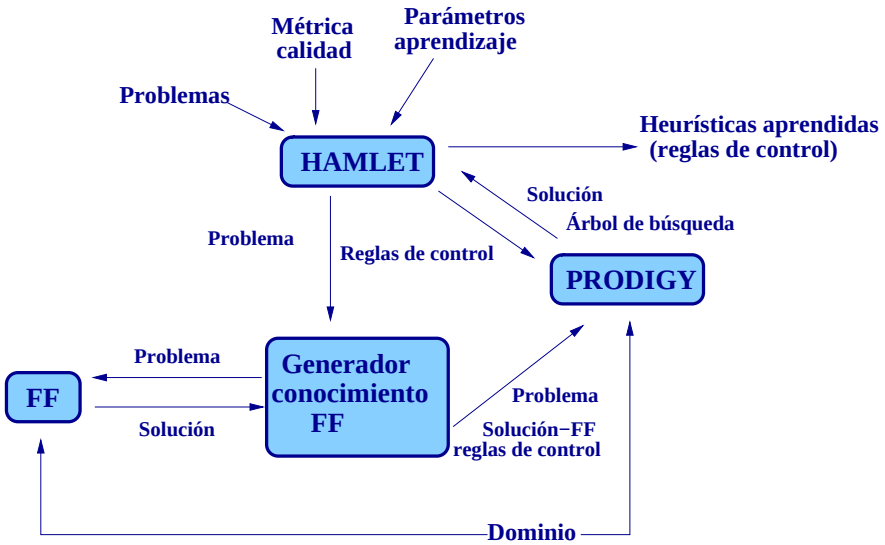

Figura 10.1: Introducción de conocimiento previo en HAMLET utilizando otro planificador, FF.

En algunos casos, ni siquiera con este conocimiento el planificador es capaz de encontrar una solución. Esto es debido a que los posibles árboles de búsqueda generados con este esquema pueden llegar a ser demasiado grandes al no poder podar las alternativas.

10.2. Conocimiento previo en EVOCK

La forma de proporcionar conocimiento previo en EVOCK es generando un conjunto de reglas, bien por una persona, bien por otro sistema de aprendizaje, que constituya la población inicial, en vez de generarla de forma aleatoria. Este conjunto supone un punto de partida que hubiese sido difícil de alcanzar con técnicas puramente evolutivas, además de focalizar el espacio de búsqueda en unas reglas determinadas.

La figura 10.2 muestra cómo las reglas generadas por HAMLET son usadas como conocimiento previo para EVOCK. En primer lugar, HAMLET aprende de un

conjunto aleatorio de problemas, explorando los árboles de búsqueda generados por PRODIGY al resolver esos problemas. Luego, estas reglas, junto a otros individuos generados de forma aleatoria, proporcionan una población inicial para EVOCK y se deja que el sistema evolutivo actúe. Las nuevas reglas obtenidas por EVOCK se proporcionan a PRODIGY para que resuelva otro conjunto de problemas aleatorios diferente al utilizado para entrenar HAMLET y se analizan los resultados.

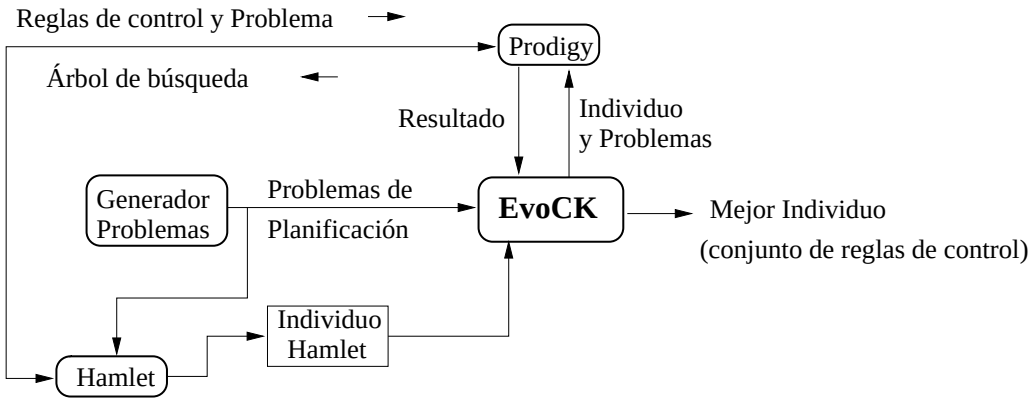

Figura 10.2: Esquema de inserción de conocimiento previo para EVOCK.

10.3. Experimentos y resultados

La experimentación se ha dividido en dos partes: una concerniente, sólo a la utilización de GEBL como fuente de conocimiento previo para HAMLET y la otra, concerniente con el resto de aspectos estudiados. En ambos casos se utilizan los dominios de las competiciones internacionales de planificación. Elegido un dominio, es necesario generar conjuntos de problemas para hacer los experimentos, tanto de aprendizaje como de validación. Pueden ser los mismos propuestos para las competiciones o generados aleatoriamente. Los experimentos consisten en proporcionar problemas a los distintos sistemas en prueba para obtener un plan solución y medir una serie de variables que permitan hacer las comparaciones pertinentes. Se fija un tiempo máximo de resolución dependiendo del tamaño de los problemas. Se ha verificado experimentalmente que aumentar dicho tiempo no mejora los resultados; si no se encuentra una solución en el tiempo establecido, la probabilidad de llegar a una solución en un tiempo con órdenes de magnitud mayor es muy pequeña.

Las variables independientes de la experimentación son el dominio de planificación y el tipo de problemas, variando la complejidad, medida, por ejemplo, por el número de metas a satisfacer. Como única variable dependiente, se mide el porcentaje de problemas de un mismo dominio resueltos en el tiempo límite de ejecución establecido.

10.3.1. GEBL como fuente para HAMLET

Una vez estudiada, en la sección anterior, la viabilidad de transferir conocimiento de GEBL a PRODIGY, en esta sección se comprueba, experimentalmente, cómo el conocimiento generado por GEBL afecta a HAMLET. Para ello se escoge un dominio y dos conjuntos diferentes de problemas, el de entrenamiento y el de validación. El conjunto de entrenamiento se utiliza en los dos sistemas de aprendizaje de la siguiente manera:

1. En HAMLET, sin utilizar conocimiento previo y aprendizaje *lazy*, se genera un conjunto de reglas de control.
2. En GEBL, con aprendizaje *eager*, se generan las reglas de `SELECT goals` y `SELECT operator` y se traducen a la sintaxis de HAMLET, como se explica en la sección 9.3.
3. Otra vez en HAMLET, pero utilizando el conjunto de reglas obtenido en el paso anterior, como conocimiento previo para hacer el entrenamiento, generando un nuevo conjunto de reglas.

Los problemas de validación se presentan a PRODIGY con los diferentes conjuntos de reglas de control y se mide la variable dependiente comentada anteriormente. Se han utilizado los mismos dominios y conjuntos de problemas que en los experimentos realizados en la sección 9.4.2.

En el apéndice C.2 están los resultados detallado y en la tabla 10.1, se muestra un resumen de los resultados obtenidos. Se representa, el porcentaje de problemas resueltos (Res) por PRODIGY, utilizando los diferentes conjuntos de reglas obtenidos por cada sistema de aprendizaje, y el número de reglas generadas (Reglas). La columna PRODIGY indica los resultados cuando se utiliza PRODIGY sin ningún conocimiento de control. La columna HAMLET, utilizando las reglas aprendidas por HAMLET sin conocimiento previo. La columna PRODIGY-GEBL, utilizando las reglas aprendidas por GEBL. La columna HAMLET+GEBL, cuando se entrena HAMLET partiendo de las reglas generadas por GEBL. El tiempo límite de ejecución fueron 30s en todos los dominios, menos en el Miconic que fue de 60s.

Los resultados muestran que la utilización de conocimiento externo en HAMLET, proveniente del sistema de aprendizaje GEBL, no incrementa el número de problemas resueltos por HAMLET. Tampoco incrementa el número de problemas resueltos por PRODIGY comparándolo con la utilización exclusiva de las reglas aprendidas por GEBL. La razón de este empeoramiento es por las técnicas inductivas utilizadas en HAMLET: la generalización y especialización incremental no aseguran la convergencia.

En el dominio Zenotravel, al volver a entrenar partiendo de las reglas generadas por GEBL, HAMLET especializa la regla para seleccionar el operador `refuel`, ex-

Dominio	PRODIGY	HAMLET		PRODIGY-GEBL		HAMLET+GEBL	
	Res	Res	Reglas	Res	Reglas	Res	Reglas
Zenotravel	37 %	40 %	5	98 %	14	40 %	14
Logistics	12 %	25 %	16	57 %	406	23 %	222
Driverlog	26 %	4 %	9	77 %	71	2 %	49
Miconic	4 %	100 %	5	99 %	13	82 %	17

Tabla 10.1: Resultados del porcentaje de problemas resueltos por PRODIGY utilizando distintos conjuntos de reglas.

plicada en el capítulo anterior 9.7, añadiendo el meta-predicado (SOME-CANDIDATE-GOALS NIL). Esto hace que el porcentaje de problemas resueltos disminuya al 40 %. Se quitó dicha condición manualmente de la regla y el planificador resolvió los 100 problemas, resolvió incluso los 2 que las reglas de GEBL solas no resolvían.

10.3.2. HAMLET y EVOCK con diferentes fuentes

Para estos experimentos se han utilizado dos de los dominios de las competiciones internacionales de planificación que son especialmente difíciles para PRODIGY: Logistics y Depots (ver apéndice A.1 y A.8.

Para evaluar cómo le afecta la utilización de conocimiento previo a un sistema de aprendizaje, SA1, se elige un dominio y dos conjuntos de problemas diferentes, el de entrenamiento y el de validación. Se ejecuta SA1 con los problemas de entrenamiento, utilizando el conocimiento previo y sin utilizarlo. Esto genera dos conjuntos de reglas de control, $CSA1P$ y $CSA1$, respectivamente. Los problemas de validación se presentan al planificador dos veces, una utilizando el conjunto de reglas $CSA1P$ y la otra con las reglas $CSA1$. Se computan las variables dependientes de la experimentación y se comparan los resultados. Si el conocimiento previo puede provenir de más de una fuente, se repite el proceso con cada una de ellas.

Para el dominio Logistics el conjunto de entrenamiento lo constituyen 400 problemas generados aleatoriamente de 1 ó 2 metas, y para Depots 200 problemas de 1 ó 2 metas. Para el conjunto de validación se han utilizado problemas generados aleatoriamente de entre 1 y 5 metas, 120 para Logistics y 60 problemas para Depots. El tiempo límite de resolución se ha establecido en 15 segundos.

La tabla 10.2 muestra los resultados del funcionamiento de todos los sistemas de forma autónoma, sin ningún conocimiento de control. También se muestran los resultados de las reglas generadas a mano por un experto en planificación. La columna Resueltos muestra el porcentaje de problemas resueltos por el sistema en los 15 segundos y la columna Reglas el número de reglas generadas.

	Logistics		Depots	
Sistema	Resueltos	Reglas	Resueltos	Reglas
PRODIGY	21 %		12 %	
EVOCK	33 %	7	52 %	2
HAMLET	36 %	32	0 %	4
Humano	100 %	37	55 %	5

Tabla 10.2: Resultados de PRODIGY, EVOCK, and HAMLET sin conocimiento previo.

La principal conclusión que se saca del análisis de estos resultados es que ambos dominios son bastante difíciles para PRODIGY y los sistemas de aprendizaje. Con las reglas proporcionadas por el humano, se consigue resolver el 100 % de los problemas en el dominio Logistics pero sólo un 55 % en el Depots. Incluso en el caso del dominio Depots, HAMLET no es capaz de resolver ninguno de los problemas. Estos resultados llevan a la conclusión de que podrían mejorarse introduciendo conocimiento previo a dichos sistemas.

La tabla 10.3, muestra los resultados de introducir conocimiento de control previo a cada uno de los sistemas de aprendizaje. En la primera columna (Fuente), se muestra la fuente de donde proviene el conocimiento, que puede ser: ninguna (si no se utiliza conocimiento previo), reglas generadas por EVOCK, por HAMLET, por la solución de otro planificador (FF) o por el humano. El resto de columnas, muestran los resultados para cada sistema de aprendizaje, representando en la columna Res el porcentaje de problemas resueltos y en la columna Mejora el porcentaje de mejora que supone el conocimiento previo frente a la no utilización de ningún conocimiento. En el caso de utilizar EVOCK como conocimiento previo para HAMLET, no siempre es posible generarlo ya que ambos sistemas utilizan un lenguaje de representación diferente y no siempre se puede adaptar. Esto se consiguió en el dominio Logistics pero no en el Depots. No se ha desarrollado de momento ninguna forma para utilizar FF como fuente de conocimiento en EVOCK.

	Logistics				Depots			
	EVOCK		HAMLET		EVOCK		HAMLET	
Fuente	Res	Mejora	Res	Mejora	Res	Mejora	Res	Mejora
Ninguna	33 %	-	36 %	-	52 %	-	0 %	-
EVOCK	-	-			-	-	57 %	57 %
HAMLET	33 %	0 %	-	-	43 %	-9 %	-	-
FF			48 %	12 %			43 %	43 %
Humano	83 %	50 %	88 %	52 %	55 %	3 %	55 %	55 %

Tabla 10.3: Resultados para EVOCK y HAMLET con conocimiento previo en los dominios Logistics y Depots. Porcentaje de problemas resueltos.

Los resultados demuestran que en el dominio Logistics, EVOCK sólo mejora

cuando el conocimiento de control viene del humano, en un 50 % y HAMLET mejora en los dos casos, cuando el conocimiento viene de FF, en un 12 % y cuando viene del humano, en un 52 %. En el dominio Depots los resultados son muy dispares en EVOCK y en HAMLET. En el primer sistema, el conocimiento previo no aporta ninguna mejora, es más, incluso empeora los resultados en un 9 % cuando el conocimiento previo viene de HAMLET. Sin embargo, HAMLET mejora notablemente en los tres casos dado que partía de un 0 % de problemas resueltos, un 57 % si el conocimiento previo viene de EVOCK, un 43 % si viene de FF y un 55 % del humano. También se demuestra que los sistemas de aprendizaje no mejoran las reglas generadas por el humano: en el dominio Logistics pasa de resolver el 100 % de los problemas a resolver un 83 % EVOCK y un 88 % HAMLET. En el dominio Depots, el porcentaje de problemas resueltos con las reglas del humano, 55 %, permanece igual cuando se utilizan como conocimiento previo tanto en HAMLET como en EVOCK. Por último, se observa que, en el dominio Depots, EVOCK tiene un rendimiento mucho mejor que HAMLET funcionando de forma independiente. EVOCK resuelve el 52 % de los problemas frente a 0 % resueltos por HAMLET y utilizar HAMLET como fuente para EVOCK hace empeorar su rendimiento (en un 9 %). Al utilizar EVOCK como fuente para HAMLET consigue mejorar, incluso, el rendimiento de utilizar EVOCK solo, aunque en un porcentaje pequeño, 5 %. En el Logistics, HAMLET resuelve el 36 % de los problemas frente al 33 % de EVOCK, apenas hay diferencia, y la utilización de HAMLET como fuente de conocimiento para EVOCK no mejora su rendimiento; sigue resolviendo el 33 % de los problemas.

10.4. Conclusiones y trabajos futuros

En esta sección se ha estudiado cómo influye la transferencia de conocimiento entre los sistemas de aprendizaje en planificación. Es decir, si dado un sistema de aprendizaje, se puede mejorar su rendimiento proporcionándole un conocimiento previo proveniente de una fuente externa al propio sistema de aprendizaje. Los sistemas de aprendizaje estudiados son HAMLET y EVOCK. En el caso de HAMLET, el aprendizaje sólo puede realizarse cuando resuelve los problemas de entrenamiento. En los dominios en que no es capaz de encontrar ninguna solución, incluso a problemas sencillos, no se puede generar ningún conocimiento. Por eso, en estos casos resulta, necesario "ayudar" a HAMLET a resolver problemas, proporcionándole un conocimiento inicial.

Los experimentos realizados demuestran que, en el caso de HAMLET, efectivamente el conocimiento previo mejora el rendimiento del sistema respecto a no utilizar conocimiento externo, en todos los casos, menos cuando se utiliza GEBL. Se han utilizado diferentes fuentes para proporcionar dicho conocimiento: reglas generadas manualmente por un humano, reglas generadas automáticamente por dos sistemas de aprendizaje diferentes (GEBL y EVOCK) y utilizando otro planificador (FF). En el caso de EVOCK, se le ha proporcionado conocimiento previo provenien-

te de dos fuentes: el humano y el sistema de aprendizaje HAMLET. EVOCK mejora su rendimiento sólo en el primer caso, cuando la fuente de conocimiento viene del humano.

Por otro lado, los experimentos también demuestran que, si el conocimiento previo proporcionado se puede aplicar directamente al planificador produciendo un mejor resultado que el obtenido utilizando las reglas aprendidas por el sistema de aprendizaje solo, utilizar dicho conocimiento externo, para entrenar por segunda vez al sistema de aprendizaje, no produce mejores resultados. Por ejemplo, en el caso de HAMLET, las reglas generadas manualmente por un humano y las reglas generadas por GEBL se pueden utilizar directamente en PRODIGY, produciendo mejores resultados que utilizando las reglas generadas exclusivamente con HAMLET. Utilizar ambos conjuntos de reglas, como conocimiento previo en HAMLET, no mejoran el rendimiento inicial conseguido con las reglas externas exclusivamente, incluso lo empeora. Sólo en un caso (cuando se utiliza EVOCK como fuente para HAMLET en el dominio Depots), la transferencia de conocimiento al sistema de aprendizaje mejora ligeramente el resultado. En el caso de EVOCK, ocurre algo similar. Las reglas proporcionadas por el humano incrementan el rendimiento de PRODIGY en un porcentaje mayor que las generadas exclusivamente por EVOCK, y al utilizarlas como conocimiento previo, no mejoran la eficiencia alcanzada con las reglas del humano. En este caso se observa también que si la fuente de conocimiento es un sistema de aprendizaje que produce peores resultados que el propio sistema, utilizar el conocimiento proporcionado es contraproducente.

Resumiendo, la transferencia de conocimiento externo a un sistema de aprendizaje, en general, incrementa la eficiencia del sistema de aprendizaje, Pero, si la fuente de conocimiento previo es otro sistema de aprendizaje diferente, con un comportamiento mejor que el sistema al que se quiere transferir, es preferible la transferencia de conocimiento directamente al planificador que al sistema de aprendizaje. Esto ocurre cuando el segundo sistema de aprendizaje es inductivo, que es el caso de HAMLET y EVOCK. Si fueran sólo deductivos, añadirían nuevas reglas que seguramente mejorarían el rendimiento. Además en el caso de HAMLET, cuando utiliza un conjunto de reglas inicial, no hace inducción sobre las nuevas que genera, sólo las utiliza para resolver más problemas de entrenamiento e iniciar nuevos procesos de búsqueda.

Otra conclusión que se deriva de estos experimentos es que el aprendizaje de conocimiento de control en planificación es complicado. Una pequeña variación en el conocimiento adquirido puede alterar el rendimiento del sistema drásticamente. Así ha ocurrido, por ejemplo, en el domino Zenotravel, al especializar una regla añadiendo un meta-predicado, hace que se resuelvan muchos menos problemas.

Como trabajos futuros se podría buscar en el espacio de estados de conjuntos de reglas con escalada en sistemas inductivos (como HAMLET), de forma que no pueda pasar a un estado que sea peor que el anterior. También, se podrían estudiar otras combinaciones de entrada, como GEBL+EVOCK, Humano+GEBL, FF+EVOCK....

Parte IV

Conclusiones finales y trabajos futuros

Capítulo 11

Conclusiones

Las aportaciones de esta tesis doctoral y las principales conclusiones obtenidas se pueden resumir en:

- Definición de un lenguaje común para la representación del conocimiento de control. Se ha demostrado experimentalmente que el lenguaje utilizado en PRODIGY de reglas de control, con una parte de precondiciones, definidas como una conjunción de meta-predicados, y un consecuente, sirve para representar conocimiento de control (o heurísticas) que incrementa la eficiencia de diferentes paradigmas de planificación. Los meta-predicados básicos de PRODIGY son útiles en los otros paradigmas y sólo es preciso implementarlos según el algoritmo de planificación de donde se aprende. Además, se ha demostrado que permite definir nuevos predicados que obtengan características particulares de la técnica de planificación.

- Desarrollo de un sistema de aprendizaje de conocimiento de control que incrementa la eficiencia resolviendo problemas de un planificador híbrido HTN-POP usado para resolver problemas de manufacturación del mundo real. Pese a que el planificador es muy eficiente, se logra mejorar su comportamiento aprendiendo conocimiento de control para seleccionar métodos de refinamiento jerárquico y para añadir operadores al plan parcial.

- Desarrollo de un sistema de aprendizaje de conocimiento de control aplicable a planificación basada en grafos de planes. El sistema no incrementa la eficiencia del planificador del que aprende, pero sí la de otro planificador que utiliza planificación bidireccional cuando se realiza transferencia de conocimiento. Además se han analizado las características del planificador que hacen que el tipo de conocimiento extraído no mejore su rendimiento para extrapolarlo a otras técnicas de planificación.

- Definición de una metodología de diseño y desarrollo de sistemas de aprendizaje de conocimiento de control basada en la explicación que sistematiza los

pasos necesarios para aplicar EBL a diferentes paradigmas de planificación. Se han analizado algunas características de los planificadores que pueden condicionar la validez de la metodología. También, se ha hecho un análisis del tipo de conocimiento que puede ser aprendido con la metodología.

- Estudio sobre la transferencia de conocimiento de control aprendido en un planificador basado en grafo de planes a otro planificador que utiliza otra técnica diferente, planificación bidireccional. Sin embargo, la transferencia en el otro sentido no ha sido posible porque el sistema de aprendizaje para el planificador basado en grafos de planes no mejora la eficiencia del planificador con conocimiento de control. También se ha hecho un análisis cualitativo del conocimiento transferido. Se ha comprobado experimentalmente que el conocimiento generado incrementa la eficacia del segundo planificador en todos los dominios seleccionados para el estudio e incluso, en dos de ellos, supera el porcentaje de problemas resueltos por el planificador de donde se extrae el conocimiento.

- Estudio sobre la transferencia de conocimiento a los propios sistemas de aprendizaje en planificación. La principal conclusión obtenida es que la transferencia a los sistemas de aprendizaje no mejora a la fuente de donde proviene el conocimiento. Es decir, no hace que el sistema de planificación resuelva más problemas cuando el sistema de aprendizaje se entrena utilizando el conocimiento externo que utilizando directamente el conocimiento proporcionado por la fuente origen. Sin embargo, sí mejora respecto a utilizar el sistema de aprendizaje solo.

Capítulo 12

Futuras líneas de trabajo

Como trabajos futuros se propone profundizar en la especificación de un lenguaje común de representación del conocimiento de control para poder seguir investigando la transferencia de conocimiento entre distintos paradigmas de planificación. Para ello, se propone aplicar la metodología de aprendizaje definida a otros planificadores para intentar la transferencia entre distintas combinaciones de planificadores. También sería interesante desarrollar un método inductivo que permita generalizar y especializar las reglas de control aprendidas, similar al de HAMLET o EVOCK. Así como, realizar algún estudio de utilidad del conocimiento aprendido por HEBL y GEBL que permita reducir la cantidad de reglas generadas a aquellas que realmente incrementen el rendimiento del sistema en que se aplican. El proceso de equiparación y ejecución de reglas se puede mejorar aplicando el algoritmo RETE en vez de la equiparación simple lineal actualmente utilizada. Por último, dentro del marco de la transferencia de conocimiento entre diferentes paradigmas, analizar también las características de los dominios de planificación que determinen el tipo de técnica o técnicas de planificación más apropiada para resolver problemas de cada dominio.

En cuanto a la transferencia de conocimiento a los propios sistemas de aprendizaje, el mayor problema encontrado es que los esquemas propuestos no mejoran el rendimiento de la propia fuente externa por lo que habría que buscar en el espacio de estados de conjuntos de reglas con escalada en sistemas inductivos de forma que no pueda pasar a un estado que sea peor que el anterior. También, sería interesante hacer el estudio sobre la transferencia a sistemas de aprendizaje no inductivos, por ejemplo, a los dos sistemas deductivos implementados. En esta línea, probar otras combinaciones de proporcionar el conocimiento externo, como GEBL+EVOCK, Humano+GEBL, FF+EVOCK..., especialmente las que requieran la intervención del humano que lleven a iniciativas mixtas de aprendizaje, automáticas y manuales, que han demostrado ser muy eficientes.

Respecto a los sistemas de aprendizaje desarrollados, en planificación basada

en grafos de planes investigar otro tipo de conocimiento que se pueda extraer, como la ordenación de las metas antes de cada proceso de búsqueda en un nivel o aprender *memoizations* genéricos que consideren el estado inicial del problema. Dada la rapidez del algoritmo GRAPHPLAN es preciso investigar formas más eficientes para hacer el proceso de equiparación posterior, bien con el ya mencionado algoritmo RETE, con los UB-Tree o cualquier otra técnica. En HEBL convendría encontrar una forma para traducir los dominios y problemas del lenguaje PDDL al utilizado por HYBIS para poder experimentar con más dominios y también hacer un análisis de las reglas aprendidas de cada tipo, las HTN y las POP.

Por último, investigar en la metodología de diseño y desarrollo de sistema de aprendizaje definida la manera de ampliarla para que el conocimiento obtenido también incremente la calidad de los planes y otros aspectos fuera de la denominada planificación clásica. Por ejemplo, de la parte de *scheduling* con que se ha integrado PRODIGY en el sistema IPSS, o de planificación condicional con que se amplió HYBIS.

Capítulo 13

Publicaciones

En esta sección se enumeran las publicaciones a las que ha dado origen la elaboración de esta tesis doctoral.

Título:	*Machine Learning in Hybrid Hierarchical and Partial-Order Planners for Manufacturing Domains*
Autores:	Susana Fernández, Ricardo Aler y Daniel Borrajo
Revista:	Applied Artificial Intelligence
Publicación:	Taylor & Francis. ISSN:0883-9514 print/1087-6545 online. DOI: 10.1080/08839510490964491
Año:	2005

Título:	*On Learning Control Knowledge for a HTN-POP Hybrid Planner*
Autores:	Susana Fernández, Ricardo Aler y Daniel Borrajo
Congreso:	First International Conference on Machine Learning and Cybernetics
Publicación:	Proceedings of the ICMLC-02
Lugar :	Beijing, China
Año:	2002

Título:	*Generación Automática de Conocimiento de Control en un Planificador Híbrido HTN-POP*
Autores:	Susana Fernández, Ricardo Aler y Daniel Borrajo
Congreso:	IBERAMIA-02. I Workshop on Planning, Scheduling and Temporal Reasoning
Publicación:	Proceedings of the IBERAMIA-02 Workshop on Planning, Scheduling and Temporal Reasoning
Lugar :	Sevilla
Año:	2002

Título:	*Using Previous Experience for Learning Planning Control Knowledge*
Autores:	Susana Fernández, Ricardo Aler y Daniel Borrajo
Congreso:	The 17th International FLAIRS Conference. Special Track on Machine Learning for Planning, Problem Solving, and Scheduling
Publicación:	Proceedings of the FLAIRS-04
Lugar:	Miami
Año:	2004

Título:	*On providing prior knowledge for learning relational search heuristics*
Autores:	Ricardo Aler, Daniel Borrajo y Susana Fernández
Congreso:	CAEPIA-03. Taller de Planificación, Scheduling y Razonamiento Temporal
Publicación:	Proceedings of the CAEPIA-03 Taller de Planificación, Scheduling y Razonamiento Temporal
Lugar :	Donostia
Año:	2003

Título:	*Transfer of Learned Heuristics among Planners*
Autores:	Susana Fernández, Ricardo Aler y Daniel Borrajo
Congreso:	AAAI-2006. Workshop on Learning for Search in the AAAI-2006
Publicación:	Por publicar
Lugar:	Boston
Año:	2006

Bibliografía

[Aarup *et al.*, 1994] M. Aarup, M.M. Arentoft, Y. Parrod, I. Stokes, H. Vadon, and J. Stager. Optimum-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. *Intelligent Scheduling*, pages 451–469, 1994.

[AIS, 1974] AISB summer conference. *The virtuous nature of bugs*, 1974.

[Aler *et al.*, 2002] R. Aler, D. Borrajo, and P. Isasi. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, 141(1-2):29–56, October 2002.

[Allen *et al.*, 1992] J. Allen, P. Langley, and S. Matwin. Knowledge and regularity in planning. In *Working notes of the AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification and Reuse*, pages 7–12, Stanford University, 1992.

[Ambite and Knoblock, 2001] J.L Ambite and C.A. Knoblock. Planning by rewriting. *Journal of Artificial Intelligence Research*, 15:207–261, 2001.

[Ambite *et al.*, 2000] J.L Ambite, C.A. Knoblock, and S. Minton. Learning plan rewriting rules. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

[Ambros-Ingerson and Stell, 1996] J.A. Ambros-Ingerson and S. Stell. Integrating planning, execution and monitoring. In *Proceedings of AAAI-88*, 1996.

[ANS, 1995] ANSI/ISA. *Batch Control Part I, Models & Terminology (S88.01)*, 1995.

[Astrom, 1965] K.J. Astrom. Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:403–406, 1965.

[Avesani *et al.*, 1998] P. Avesani, A. Perini, and F. Ricci. The twofold integration of CBR in decision support systems. In *Proceedings AAAI98 Workshop on Case-Based Reasoning Integrations*, 1998.

[Avesani *et al.*, 2000] P. Avesani, A. Perini, and F. Ricci. Interactive case-based planning for forest fire management. *Applied Intelligence*, 13:41–57, 2000.

[Bacchus and Ady, 2001] F. Bacchus and M. Ady. Planning with resources and concurrency: A forward chaining approach. In *In Proc. of IJCAI-2001*, 2001.

[Bacchus and Kabanza, 1996] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222, Portland, Oregon, USA, 1996. AAAI Press / The MIT Press.

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

[Bacchus and Yang, 1994] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1994.

[Barrett and Weld, 1994] A. Barrett and D.S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1), 1994.

[Barto *et al.*, 1995] A. G. Barto, S.J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1(72), 1995.

[Baum, 1995] E. Baum. Toward a model of mind as a laissezfaire economy of idiots. In *In Proc. Int. Conf. on Machine Learning*, 1995.

[Bayardo and Miranker, 1996] R. Bayardo and R. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *In Pro. of the 13th Nat'l conf. on Artificial Intelligence*, 1996.

[Beek, 1994] P. Beek. CSPLIB: A library of CSP routines, 1994. University of Alberta.

[Bellman and Kalaba, 1965] R. Bellman and R. Kalaba. *Dynamic Programming and Modern Control Theory*. Academic Press, 1965.

[Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ., 1957.

[Bergmann and Wilke, 1996] R. Bergmann and W. Wilke. On the role of abstraction in case-based reasoning. In Ian Smith and Bob Faltings, editors, *Proceedings of the Third European Workshop on Advances in Case-Based Reasoning*, volume 1168 of *LNAI*, pages 28–43, Berlin, 1996. Springer.

[Bergmann *et al.*, 1998] R. Bergmann, H. Muñoz-Avila, M. Veloso, and E. Melis. CBR applied to planning. *Lecture Notes in Computer Science*, 1400:169, 1998.

[Bertoli *et al.*, 2001] P. Bertoli, A. Cimatti, and M. Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In *In Proc. IJCAI-01*, 2001.

[Bhatnagar and Mostow, 1994] N. Bhatnagar and J. Mostow. On-line learning from search failures. *Machine Learning*, 15(1):69–117, 1994.

[Bhatnagar, 1992] N. Bhatnagar. Learning by incomplete explanations of failures in recursive domains. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 30–36, Aberdeen, Scotland, 1992. Morgan Kaufmann.

[Biundo and Schattenberg, 2001] S. Biundo and B. Schattenberg. From abstract crisis to concrete relief: A preliminary report on combining state abstraction and htn planning. In *In Proceedings of the European Conference on Planning*, 2001.

[Blum and Furst, 1995] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. In Chris S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, pages 1636–1642, Montréal, Canada, August 1995. Morgan Kaufmann.

[Bonet and Geffner, 2001a] B. Bonet and H. Geffner. Gpt: A tool for planning with uncertainty and partial information. In *In Workshop on Planning with Uncertainty and Partial Information, IJCAI-2001*, 2001.

[Bonet and Geffner, 2001b] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001.

[Borrajo and Veloso, 1997] D. Borrajo and M. M. Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 11(1-5):371–405, February 1997.

[Brachman and Levesque, 1984] R. Brachman and H. Levesque. The tractability of subsumption in frame based description languages. In *In Proc. of AAAI-84*, 1984.

[Breslow and Aha, 1997] L. Breslow and D. W. Aha. NaCoDAE: Navy conversational decision aids environment. Technical Report AIC-97-018, Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, 1997.

[Bylander, 1991] T. Bylander. Complexity results for planning. In Ray Myopoulos, John; Reiter, editor, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 274–279, Sydney, Australia, August 1991. Morgan Kaufmann.

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

[Carbonell *et al.*, 1990] J.G. Carbonell, C.A. Knoblock, and S. Minton. Prodigy: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990. Also Technical Report CMU-CS-89-189.

[Carbonell *et al.*, 1992] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela M. Veloso, and Xuemei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University, 1992.

[Carbonell, 1983] J.G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach*, pages 137–162, Palo Alto, CA, 1983. Tioga Press.

[Carrick *et al.*, 1999] C. Carrick, Q. Yang, I. Abi-Zeid, and L. Lamontagne. Activating CBR systems through autonomous information gathering. *Lecture Notes in Computer Science*, 1650, 1999.

[Castillo *et al.*, 2000] L. Castillo, J. Fdez-Olivares, and A. González. A hybrid hierarchical/operator-based planning approach for the design of control programs. In *ECAI Workshop on Planning and configuration: New results in planning, scheduling and design*, 2000.

[Castillo *et al.*, 2001] Luis Castillo, Juan Fernández-Olivares, and Antonio González. Mixing expressiveness and efficiency in a manufacturing planner. *Journal of Experimental and Theoretical Artificial Intelligence*, 13:141–162, 2001.

[Castillo *et al.*, 2006] L. Castillo, J. Fdez.-Olivares, O. García-Pérez, and F. Palao. Bringing users and planning technology together. experiences in SIADEX. In *16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 2006.

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[Chase *et al.*, 1989] M. P. Chase, M. Zweben, R. L. Piazza, J. D. Burger, P. P. Maglio, and H. Hirsh. Approximating learned search control knowledge. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 218–220, 1989.

[Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *In JAIR*, 13, 2000.

[Cohen, 1990] W. W. Cohen. Learning approximate control rules of high utility. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 268–276, Austin, TX, 1990. Morgan Kaufmann.

[Cross and Walker, 1994] S.E. Cross and E. Walker. *DART:applying knowledge-based planning and scheduling to crisis action planning*. Morgan Kaufmann, 1994.

[Currie and Tate, 1985] K. Currie and A. Tate. O-plan: Control in the open planning architecture. In *BCS Expert systems conference*, 1985.

[Currie and Tate, 1991] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[Dechter, 2003] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Dietterich and Flann, 1997] T.G. Dietterich and N.S. Flann. Explanation-based and reinforcement learning: A unified view. *Machine Learning*, 28(2/3):169–210, 1997.

[Do and Kambhampati, 2001a] M.B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving planning graph by compiling it into a CSP. *AI Journal*, 2001.

[Do and Kambhampati, 2001b] M.B. Do and S. Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *In Proc. ECP-01*, 2001.

[Doherty *et al.*, 1998] P. Doherty, J. Gustafsson, L. Karlsson, and et al. Temporal action logics (TAL): Language specification and tutorial. Linköping Electronic Articles in Computer and Information Science, Vol. 3, October 1998.

[Doorenbos and Veloso, 1993] R.B. Doorenbos and M. M. Veloso. Knowledge organization and the utility problem. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 28–34, Amherst, MA, June 1993.

[Draper *et al.*, 1993] D. Draper, S. Hanks, and D.S. Weld. Probabilistic planning with information gathering and contigent execution. Technical Report TR93-12-04, Department of Computer Science and Engineering, University of Washington, 1993.

[Edelkamp and Helmert, 2000] S. Edelkamp and M. Helmert. The implementation of Mips. In AIPS-Workshop on Model-Theoretic Approaches to Planning, pp. 18– 25, 2000.

[Erol *et al.*, 1994a] K. Erol, J. A. Hendler, and D.S. Nau. HTN planning:complexity and expresisvity. In *Proceedings of the National Conference on Artificial Intelligence*, volume I, pages 1123–1128. AAAI Press/MIT Press, 1994.

[Erol *et al.*, 1994b] K. Erol, J.A. Hendler, and D.S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.

[Estlin and Mooney, 1997] T.A. Estlin and R.J. Mooney. Learning to improve both efficiency and quality of planning. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1227–1233, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[Etzioni and Minton, 1992] Oren Etzioni and Steven Minton. Why EBL produces overly-specific knowledge: A critique of the Prodigy approaches. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 137–143, Aberdeen, Scotland, 1992. Morgan Kaufmann.

[Etzioni *et al.*, 1992] O. Etzioni, S. Hanks, D.S. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *Proceedings of KRR-92*, 1992.

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

[Fernández and Borrajo, 2002] F. Fernández and D. Borrajo. On determinism handling while learning reduced state space representations. In *Proceedings of the European Conference on Artificial Intelligence (ECAI 2002)*, Lyon (France), July 2002.

[Fernández *et al.*, 2002] S. Fernández, I.M. Galván, and R. Aler. A first approach to tackling planning problems with neural networks. In *Workshop on Knowledge Engineering Tools and Techniques for AI Planning in the AIPS-2002*, April 2002.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Fink and Blythe, 2005] Eugene Fink and Jim Blythe. Prodigy bidirectional planning. *Journal of Experimental & Theoretical Artificial Intelligence*, 17(3):161–200(40), September 2005.

[Fink and Yang, 1992] E. Fink and Q. Yang. Formalizing plan justifications. In *In Proceedings of the Nith Conference of the Canadian Sociey for Computational Studies of Intelligence*, pages 9–14, 1992.

[Forgy, 1982] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, 19:17–37, 1982.

[Fox and Long, 2001] M. Fox and D. Long. Hybrid STAN: Identifying and managing combinatorial optimization sub-problems in planning. In *IJCAI*, pages 445–452, 2001.

[Fox and Long, 2002a] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *In Proc. of AIPS-02*, 2002.

[Fox and Long, 2002b] M. Fox and D. Long. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. University of Durham, Durham (UK), February 2002.

[Fox, 1997] M. Fox. Natural hierarchical planning using operator decomposition. In Sam Steel, editor, *Proceedings of the Fourth European Conference on Planning*, Lecture Notes in Artificial Intelligence, pages 195–207, Toulouse, France, September 1997. Springer-Verlag.

[Frost and Dechter, 1994] D. Frost and R. Dechter. Dead-end driven learning. In *In Proc. AAAI-94*, 1994.

[García-Martínez and Borrajo, 1998] R. García-Martínez and D. Borrajo. An integrated approach of learning, planning, and execution. *IEEE Transactions on Systems, Man and Cybernetics*, 1998. Submitted.

[Garrido and Barber, 2001] A. Garrido and F. Barber. Integrating planning and scheduling. *Applied Artificial Intelligence*, 15(5):471–491, 2001.

[Garrido and Long, 2004] A. Garrido and D. Long. Planning with numeric varibles in multiobjective planning. In *Proceedings of ECAI-2004*, pages 662–666, 2004.

[Garrido and Onaindía, 2003] A. Garrido and E. Onaindía. On the application of least-commitment and heuristic search in temporal planning. In *Proceedings of IJCAI-2003*, pages 942–947. Morgan Kaufmann, 2003.

[Geffner, 1999] H. Geffner. Functional strips: a more general language for planning and problem solving. In *Presented at the Logic-based AI Workshop*, 1999.

[Geffner, 2002] H. Geffner. Perspectives on artificial intelligence planning. In *Eighteenth national conference on Artificial intelligence*, pages 1013–1023. American Association for Artificial Intelligence, 2002.

[Georgeff, 1987] M.P. Georgeff. Reactive reasoning and planning. In *Proceedings of AAAI-87*, 1987.

[Ghallab and H., 1994] M. Ghallab and Laruelle H. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings in AIPS-94*, pages 61–67, 1994.

[Godefroid and Kabanza, 1991] P. Godefroid and F. Kabanza. An efficient reactive planner for synthesising reactive plans. In *Proceedings of AAAI91*, 1991.

[Golden *et al.*, 1994] K. Golden, O. Etzioni, and D.S. Weld. Omnipotence without omniscience: Efficient sensor management for planning. In *Proceedings of AAAI-94*, 1994.

[Green, 1969] C.C. Green. Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 219–239, Washington DC, 1969. Also in Readings in Planning.

[Hammond, 1989] K.J. Hammond. *Case-based Planning: Viewing Planning as a Memory Task*. Academic Press, New York, NY, 1989.

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

[Haslum and Geffner, 2001] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Procedings of the 6th European Conference on Planning (ECP-2001)*, 2001.

[Hoffmann, 2001] J. Hoffmann. FF: The fast-forward planning system. *AI Magazine*, 22(3):57 – 62, 2001.

[Howard, 1990] Ronald A. Howard. *Dynamic Programming and Markov Processes*. The M.I.T. Press, Cambridge, 1990.

[Huang *et al.*, 1999] Y. Huang, B. Selman, and H. Kautz. Control knowledge in planning: Benefits and tradeoffs. In *AAAI/IAAI*, pages 511–517, 1999.

[Huang *et al.*, 2000] Y. Huang, B. Selman, and H. Kautz. Learning declarative control rules for constraint-based planning. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, Stanford, CA (USA), June-July 2000.

[Huffman *et al.*, 1992] S.B. Huffman, D.J Pearson, and J.E. Laird. Correcting imperfect domain theories: A knowledge level analysis. In S. Chipman and A. Meyrowitz, editors, Kluwer Academic Press, 1992.

[Iba, 1989] G.A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3(4):285–317, March 1989.

[Ihrig and Kambhampati, 1996a] L. Ihrig and S. Kambhampati. Design and implementation of a replay framework based on a partial order planner. In *AAAI-96*, 1996.

[Ihrig and Kambhampati, 1996b] L. Ihrig and S. Kambhampati. Plan-space vs. state-space planning in reuse and replay. *ASU CSE TR 94-006*, 1996.

[Ihrig and Kambhampati, 1997] L. Ihrig and S. Kambhampati. Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Artificial Intelligence*, 7:161–198, 1997.

[Ilghami *et al.*, 2002] O. Ilghami, D. Nau, H. Muñoz-Avila, and D. W. Aha. Camel: Learning method preconditions for HTN planning. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 131–141, Toulouse (France), 23-17 April 2002. AAAI Press.

[Iwamoto, 1994] Masahiko Iwamoto. A planner with quality goal and its speed-up learning for optimization problem. In K. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS94)*, pages 281–286, Chicago, Illinois, June 1994.

[J.R., 1983] Quinlan J.R. Learning efficient classifiation procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach, Volume I*. Morgan Kaufman, 1983.

[J.W. *et al.*, 1991] Shavilik J.W., Mooney R.J., and Towell G.G. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

[Kambhampati and Hendler, 1989] S. Kambhampati and J. A. Hendler. Flexible reuse of plans via annotation and verification. In *IEEE conf. on applications of AI*, 1989.

[Kambhampati *et al.*, 1996] S. Kambhampati, S. Katukam, and Y. Qu. Failure driven search control for partial order planners: An explanation based approach. *Artificial Intelligence*, 88, 1996.

[Kambhampati, 1989] Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. PhD thesis, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD, 1989.

[Kambhampati, 2000] S. Kambhampati. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in graphplan. *Journal of Artificial Intelligence Research*, 12:1–34, 2000.

[Katukam and Kambhampati, 1994] S. Katukam and S. Kambhampati. Learning explanation-based search control rules for partial order planning. In *Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1*, pages 582–587, Menlo Park, CA, USA, July 31–August 4 1994. AAAI Press.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In Howard Shrobe and Ted Senator, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California, 1996. AAAI Press.

[Kautz and Selman, 1999] Henry Kautz and Bart Selman. Blackbox: Unifying sat-based and graph-based planning. In *Procedings IJCAI-99*, 1999.

[Keller, 1987] R. Keller. *The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance*. PhD thesis, Rutgers University, 1987.

[Kettler *et al.*, 1994] B. P. Kettler, J. A. Hendler, A. W. Andersen, and M. P. Evett. Massively parallel support for case-based planning. *IEEE Expert*, 2:8–14, 1994.

[Khardon, 1999] Roni Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

[Knoblock *et al.*, 1991] C.A. Knoblock, S. Minton, and O. Etzioni. Integrating abstraction and explanation based learning in PRODIGY. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 541–546, 1991.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991. Available as technical report CMU-CS-91-120.

[Knoblock, 1993] C.A. Knoblock. *Generating Abstraction Hierarchies*. Kluwer Academic Publishers, 1993.

[Knoblock, 1995] C.A. Knoblock. Planning, executing, sensing an replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[Knoblock, 1996] C.A. Knoblock. AI Planning systems in the real world. *IEEE Expert*, pages 4–12, 1996.

[Koehler *et al.*, 1997] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In *In proceddings ECP-97*, pages 273–285, 1997.

[Koehler, 1994] Jana Koehler. Flexible plan reuse in a formal framework. In C. Bäckström and E. Sandewell, editors, *Current Trends in AI Planning: Proceedings of the 2nd European Workshop on Planning (EWSP-93)*, pages 171–184, Vadstena, Sweeden, 1994. IOS Press (Amsterdam).

[Koehler, 1998] J. Koehler. Planning under resource constraints. In *In Proc. ECAI-98*, 1998.

[Kokkinaki and Valavanis, 1995] A.I. Kokkinaki and K.P. Valavanis. On the comparison of AI and DAI based planning techniques for automated manufacturing systems. *Intelligent and Robotic Systems:Theory and Applications*, 13:201–245, 1995.

[Kolbe and Brauburger, 1997] T. Kolbe and J. Brauburger. PLAGIATOR: A learning prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 256–259, Berlin, July 13–17 1997. Springer.

[Korf, 1985] Richard E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[Koza, 1992] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[Koza, 1994] J.R. Koza. *Genetic Programming II*. The MIT Press, 1994.

[Kumar, 1986] P. R. Kumar. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice Hall, 1986.

[Kurien *et al.*, 2002] J. Kurien, P.P. Nayak, and D.E. Smith. Fragment-based conformant planning. In *In Proc. of the AIPS-02*, 2002.

[Kushmerick *et al.*, 1995] N. Kushmerick, S. Hanks, and D.S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[Kvarnström and Doherty, 2000] J. Kvarnström and P. Doherty. TALplanner: A temporal logic based forward chaining planner. Annals of Mathematics and Articial Intelligence, Volume 30, Numbers 1-4, Pages 119-169, 2000.

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

[Leckie, 1993] Christopher Leckie. Being correct is not enough - the role of domain knowledge in the utility problem. In *Proceedings of the 6th Australian Joint Conference on AI (AI'93)*, November 1993.

[Liatsos and Richards, 1999] V. Liatsos and B. Richards. Scaleability in planning. In *In Proc. ECP-99*, 1999.

[Littman *et al.*, 1995] M.L. Littman, A. Cassandra, and L.P Kaelbling. Learning policies for partially observable environments: scaling up. In *In Proceedings of International Machine Learning Conference*, pages 362–370, 1995.

[Long and Fox, 1999] Dereck Long and Maria Fox. Automatic synthesis and use of generic types in planning (tr 3/99). Technical report, Durham University, 1999.

[Lotem *et al.*, 1999] A. Lotem, D.S. Nau, and J. Hendler. Using planning graphs for solving HTN planning problems. In *In Proc. AAAI-99*, 1999.

[Luger and Stubblefield, 1993] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison Wesley, 1993.

[Mahadevan, 1996] S. Mahadevan. Average reward reinforcement learing: Foundations, algorithms, and empirical learnig. *Machine Learning*, 22:159–195, 1996.

[Mahavedan and Connell, 1992] S. Mahavedan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[Mali and Kambhampati, 1998] A. Mali and S. Kambhampati. Encoding htn planning in propositional logic. In *In Proc. AIPS-98*, 1998.

[Mali, 1999] A. Mali. Hierarchical task network planning as satisfiability. In *In Proc. ECP-99*, 1999.

[Manna and Waldinger, 1980] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2:90–121, 1980.

[Martin and Geffner, 2000] M. Martin and H. Geffner. Learning generalized policies in planning using concept languages. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS00)*, June 2000.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, 1991.

[McDermott, 1999] Drew McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109:1–361, 1999.

[McDermott, 2000] John McDermott. The AIPS'98 planning competition. *AI Magazine*, 21(2), 2000.

[Minton *et al.*, 1989] Steven Minton, Craig A. Knoblock, Dan R. Kuokka, Yolanda Gil, Robert L. Joseph, and Jaime G. Carbonell. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University, 1989.

[Minton, 1988] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.

[Mitchell *et al.*, 1982] T. Mitchell, T. Utgoff, and R. Banerji. *Machine Learning: An Artificial Intelligence Approach*, chapter Learning problem solving heuristics by experimentation. Morgan Kaufmann, 1982.

[Mitchell *et al.*, 1986] T. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Mitchell, 1977] Tom Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th IJCAI*, pages 305–310, MIT, Cambridge, Massachusetts, 1977.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Mostow and Bhatnagar, 1987] Jack Mostow and Neeraj Bhatnagar. Failsafe – A floor planner that uses EBG to learn from its failures. In John McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 249–255, Milan, Italy, August 1987. Morgan Kaufmann.

[Muscettola *et al.*, 1998] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[Muscettola, 1994] N. Muscettola. HSTS: Integrating planning and scheduling. *Intelligent Scheduling*, pages 169–212, 1994.

[Muñoz-Avila and Huellen, 1995] Hector Muñoz-Avila and Jochem Huellen. Retrieving cases in structured domains by using goal dependencies. In Manuela Veloso and Agnar Aamodt, editors, *Proceedings of the 1st International Conference on Case-Based Reasoning Research and Development*, volume 1010 of *LNAI*, pages 241–252, Berlin, 1995. Springer Verlag.

[Muñoz-Avila *et al.*, 1999] Héctor Muñoz-Avila, David W. Aha, Leonard A. Breslow, Rosina Weber, and Dana Nau. HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 879–885, 1999.

[Myers *et al.*, 2001] K.L. Myers, S.F. Smith, D.W. Hildum, P.A. Jarvis, and R. de Lacaze. Integrating planning and scheduling through adaptation of resource intensity estimates. In Amedeo Cesta and Daniel Borrajo, editors, *Preprints of the Sixth European Conference on Planning (ECP'01)*, pages 133–144, Toledo (Spain), September 2001.

[Nau *et al.*, 1999] D.S Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the IJCAI-99*, pages 968–973, Stockholm (Sweden), August 1999.

[Nebel *et al.*, 1997] Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In *ECP-97*, pages 338–350, 1997.

[Newell and Simon, 1963] Allen Newell and Herbert Simon. GPS: A program that simulates human thought. In E.A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963. Also in Readings in Planning.

[Nigenda *et al.*, 2000] Romeo Sanchez Nigenda, XuanLong Nguyen, and Subbarao Kambhampati. AltAlt: Combining the advantages of graphplan and heuristic state search, 2000.

[Olawsky and Gini, 1990] D. Olawsky and M. Gini. Deferred planning and sensor use. In Katia P. Sycara, editor, *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, California, 5–8 November 1990. Morgan Kaufmann.

[Olawsky *et al.*, 1995] D. Olawsky, K. Krebsbach, and M. Gini. An analysis of sensor-based task planning. Technical report, Dep. Comp. Science. University of Minneapolis, 1995.

[Parr and Russel, 1995] R. Parr and S. Russel. Approximating optimal policies for partially observable stochastic domains. In *In Proceedings of National Conference On Artificial Intelligence*, pages 1088–1093, 1995.

[Pednault, 1989] Edwin Pednault. ADL: Exploring the middle ground between Strips and the situation calculus. In *Proceedings of the Conference on Knowledge Representation and Reasoning*, pages 324–332, 1989.

[Pednault, 1991] E. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 91)*, pages 240–245, 1991.

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

[Penberthy and Weld, 1994] J.S. Penberthy and D.S. Weld. Temporal planning with continuous change. In *In Proc. AAAI-94*, 1994.

[Pereira *et al.*, 1997] C. Pereira, F. Garcia, J. Lang, and R. Martin-Clouaire. Planning with graded nondeterministic actions: a possibilistic approach. *International Journal of Intelligent Systems*, 12:935–962, 1997.

[Pryor and Collins, 1996] L. Pryor and G. Collins. Planning for contingencies: A decision based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[Pérez and Etzioni, 1992] Alicia Pérez and Oren Etzioni. DYNAMIC: A new role for training problems in EBL. In D. Sleeman and P. Edwards, editors, *Machine Learning: Proceeding of the Ninth International Conference (ML92)*. Morgan Kauffmann, 1992.

[Pérez, 1995] M. Alicia Pérez. *Learning Search Control Knowledge to Improve Plan Quality*. PhD thesis, School of Computer Science, Carnegie Mellon University, July 1995. Also available as Technical Report CMU-CS-95-175, School of Computer Science, Carnegie Mellon University.

[Quinlan and Cameron-Jones, 1995] J.R. Quinlan and R.M. Cameron-Jones. Introduction of logic programs: FOIL and related systems. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):287–312, 1995.

[Quinlan, 1993] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[R-Moreno, 2003] MD. R-Moreno. *Representing and Planning Tasks with Time and Resources*. PhD thesis, Dpto. de Automática, Universidad de Alcalá, 2003.

[Raedt and Bruynooghe, 1992] L. De Raedt and M. Bruynooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, March 1992.

[Reddy and Tadepalli, 1997] C. Reddy and P. Tadepalli. Learning Horn definitions using equivalence and membership queries. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*. Springer Verlag, 1997.

[Refanidis and Vlahavas, 1999] Ioannis Refanidis and Ioannis P. Vlahavas. GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables. In *ECP*, pages 347–359, 1999.

[Reiser and Kaindl, 1994] Christian Reiser and Hermann Kaindl. Case-based reasoning for multi-step problems and its integration with heuristic search. In *Proceedings of the AAAI-94 Workshop on Case-Based Reasoning*, pages 101–105. AAAI Press, 1994.

[Rivest, 1987] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, November 1987.

[Rodríguez-Moreno *et al.*, 2004] María Dolores Rodríguez-Moreno, Daniel Borrajo, and Daniel Meziat. An ai planning-based tool for scheduling satellite nominal operations. *AI Magazine*, 25(4):9–27, 2004.

[Ruby and Kibler, 1989] David Ruby and Dennis Kibler. Learning subgoal sequences in planning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 609–614, San Mateo, CA, 1989. Morgan Kaufmann.

[Ruby and Kibler, 1992] David Ruby and Dennis Kibler. Learning episodes for optimization. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 379–384, Aberdeen, Scotland, 1992. Morgan Kaufmann.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Sacerdoti, 1975] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of IJCAI-75*, pages 206–213, 1975.

[Schattenberg and Biundo, 2002] B. Schattenberg and S. Biundo. On the identification and use of hierarchical resources in planning and scheduling. In *In Proceedings of the International Conference on AI Planning and Scheduling*, 2002.

[Schmill *et al.*, 2000] M. Schmill, T. Oates, and P. Cohen. Learning planning operators in real-world, partially observable environments. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS00)*, June 2000.

[Schoppers, 1987] M.J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings IJCAI87*, 1987.

[Shavlik, 1990] Jude W. Shavlik. Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning*, 5(1):39–70, March 1990.

[Simon, 1983] Herbert A. Simon. *Why should machine learn?. Machine Learning: an Artificial Intelligence Approach*. Morgan Kaufmann, Palo Alto, CA, 1983.

[Smith and Weld, 1998] D.E. Smith and D.S. Weld. Conformant graphplan. In *In Proc. AAAI-98*, 1998.

[Smith and Weld, 1999] D.E. Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI*, pages 326–337, 1999.

[Spector, 1994] L. Spector. Genetic programming and AI planning systems. In *Proceedings of Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[Stone *et al.*, 1994] Peter Stone, Manuela M. Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Artificial Intelligence Planning Systems*, pages 164–169, 1994.

[Sutton, 1990] Richard Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.

[Sutton, 1991] R. S. Sutton. Planning by incremental dynamic programming. In *Eighth International Workshop on Machine Learning*, pages 353–357, 1991.

[Tadepalli and Ok, 1998] P. Tadepalli and D. Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 100(1-2):177–223, 1998.

[Tadepalli, 1989] Prasad Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 694–700, San Mateo, CA, 1989. Morgan Kaufmann.

[Tambe *et al.*, 1990] Milind Tambe, Allen Newell, and Paul S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5(3):299–348, August 1990.

[Tate *et al.*, 1994] A. Tate, B. Drabble, and R. Kirby. O-Plan 2. In M. Fox and M. Zweben, editors, *Knowledge Based Scheduling*. Morgan Kaufmann, San Mateo, California, 1994.

[Tate, 1976] Austin Tate. Project planning using a hierarchic non-linear planner. Research Report 25, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1976.

[Tsang, 1993] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[Upal, 1999] Muhammad Afzal Upal. Using mixed-initiative setting to improve planning performance. In *In Proceedings of the Eighth International Conference on Intelligent Systems*, pages 78–81. ISCA Press, Cary, NC, 1999.

[Upal, 2003] Muhammad Afzal Upal. Learning general graphplan memos through static domain analysis. In *In Proceedings of the Sixteenth Canadian Conference on Artificial Intelligent*, 2003.

[Utgoff *et al.*, 1997] P. Utgoff, N.C Berkman, and J.A Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5–44, 1997.

[van Beek and Chen, 1999] Peter van Beek and Xinguang Chen. Cplan: A constraint programming approach to planning. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 585–590, 1999.

[Veloso and Carbonell, 1993] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278, March 1993.

[Veloso *et al.*, 1995] Manuela M. Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81–120, 1995.

[Veloso, 1994a] Manuela M. Veloso. Flexible strategy learning: Analogical replay of problem solving episodes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 595–600, Seattle, WA, August 1994. AAAI Press.

[Veloso, 1994b] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.

[Viswanathan *et al.*, 1998] S. Viswanathan, C. Johnsson, R. Srinivasan, V. Venkatasubramanian, and K-E. Arzen. Procedure synthesis for batch processes: Part I. knowledge representation and planning framework. *Computers and Chemical Engineering*, 22:1673–1685, 1998.

[Warren, 1974] David Warren. WARPLAN: A system for generating plans. Technical report, Department of Computational Logic, University of Edinburgh, 1974. Memo No. 76.

[Watkins and Dayan, 1992] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[Weld and Etzioni, 1994] D.S. Weld and O. Etzioni. The first law of robotics: A call to arms. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington, 1994. AAAI Press.

[Weld, 1994] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[Wilkins, 1984] David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[Williamson and Hanks, 1994] Mike Williamson and Steve Hanks. Optimal planning with a goal-directed utility model. In K. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS94)*, pages 176–181, Chicago, Illinois, June 1994.

[Wolfman and Weld, 1999] S. Wolfman and D.S. Weld. Combining linear programming and satisfiability solving for resource planning. In *In Proc. IJCAI-99*, 1999.

[Wooldridge and Jennings, 1994] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. Hypertext version of Knowledge Engineering Review paper, 1994.

[Yang *et al.*, 1996] Qiang Yang, Josh Tenenberg, and Steve Woods. On the implementation and evaluation of ABTWEAK. *Computational Intelligence*, 12, 1996.

[Young *et al.*, 1994] R. Young, M. Pollack, and J. Moore. Decomposition and causality in partial order planning. In K. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS94)*, Chicago, Illinois, June 1994.

[Zelle and Mooney, 1993] J. Zelle and R. Mooney. Combining FOIL and EBG to speed-up logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1106–1113, Chambery, France, 1993. Morgan Kaufmann.

[Zimmerman and Kambhampati, 2003] Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2):73–96, 2003.

[Zimmerman and Kambhampati, 2005] Terry Zimmerman and Subbarao Kambhampati. Using memory to transform search on the planning graph. *Journal of Artificial Intelligence Research*, 23:533–585, 2005.

[Zimmerman, 1996] T. Zimmerman. Neural network guided search control in partial order planning. In *Student poster paper, AAAI-96*, Portland, Oregon, USA, 1996.

Apéndice A

Principales dominios de planificación

En este apéndice se describen los principales dominios de planificación utilizados para realizar los experimentos que validan las ideas y nuevos sistemas implementados en la tesis.

A.1. Dominio Logistics

El dominio Logistics representa problemas de logística de paquetes. Existen un número determinado de paquetes que deben ser transportados a diferentes lugares. El transporte se realiza por camiones, cuando los desplazamientos ocurren entre localidades de la misma ciudad, o por avión, cuando hay que cambiar de ciudad. Los lugares considerados son oficinas de correo y aeropuertos. Los camiones pueden alcanzar ambos sitios y los aviones sólo los aeropuertos. Hay varias versiones de este dominio y la utilizada en la tesis es la de la competición internacional de planificación del año 1998. La formulación del dominio completo en lenguaje PDDL es la siguiente:

```
(define (domain logistics-aips98)
    (:requirements :strips :equality)
    (:predicates (object ?o) (truck ?t) (at ?o ?l) (inside ?o ?t)
                 (airplane ?p) (airport ?s) (loc-at ?s ?city) (city ?c)
                 (location ?l))

    (:action load-truck
       :parameters (?o ?truck ?loc)
       :precondition (and (object ?o) (truck ?truck) (location ?loc)
                          (at ?o ?loc) (at ?truck ?loc))
       :effect (and (not (at ?o ?loc)) (inside ?o ?truck)))

    (:action load-airplane
       :parameters (?o ?p ?loc)
```

```
      :precondition (and (object ?o) (airplane ?p)
                         (airport ?loc) (at ?o ?loc) (at ?p ?loc))
      :effect (and (not (at ?o ?loc))    (inside ?o ?p)))

   (:action unload-truck
      :parameters (?o ?truck ?loc)
      :precondition (and (object ?o) (location ?loc) (truck ?truck)
                         (inside ?o ?truck) (at ?truck ?loc))
      :effect (and (at ?o ?loc) (not (inside ?o ?truck))))

   (:action unload-airplane
      :parameters (?o ?p ?loc)
      :precondition (and (object ?o) (location ?loc) (airplane ?p)
                         (inside ?o ?p) (at ?p ?loc))
      :effect (and (at ?o ?loc) (not (inside ?o ?p))))

   (:action fly-airplane
      :parameters (?p ?s ?d)
      :precondition (and (airplane ?p) (airport ?s) (airport ?d) (at ?p ?s)
                         (not (= ?s ?d)))
      :effect (and (at ?p ?d)    (not (at ?p ?s))))

   (:action drive-truck
      :parameters (?truck ?s ?d ?city)
      :precondition (and (truck ?truck) (location ?s) (location ?d)
                    (city ?city) (at ?truck ?s) (loc-at ?s ?city)
                    (loc-at ?d ?city) (not (= ?s ?d)))
      :effect (and (at ?truck ?d)    (not (at ?truck ?s)))))
```

A.2. Dominio Zenotravel

En el domino Zenotravel existen ciudades, personas y aviones. Cada persona está inicialmente en una ciudad y las metas de los problemas son desplazarlas a diferentes ciudades. Para ello se deben utilizar los aviones que pueden volar a dos velocidades, consumiendo más o menos combustible, y expresadas mediante dos operadores diferentes: `fly` y `zoom` respectivamente. Este dominio se utilizó en la competición internacional de planificación del año 2002 existiendo diferentes versiones. La utilizada en la tesis es la versión STRIPS en que el nivel de combustible de los aviones viene representado por niveles discretos cada uno expresado por un literal del tipo `(flevel fl0)` (normalmente desde el nivel 0 hasta el 6) y en las definiciones de los problemas se añaden literales del tipo `(next fl0 fl1)` para indicar que el siguiente nivel de combustible del `fl0` es el nivel `fl1`. La formulación completa del dominio en lenguaje PDDL es la siguiente:

```
(define (domain zeno-travel)
    (:requirements :strips :equality)
    (:predicates (at ?x ?c) (in ?p ?a) (fuel-level ?a ?l) (next ?l1 ?l2)
                 (aircraft ?p) (person ?x) (city ?x) (flevel ?x))

    (:action board
       :parameters ( ?p ?a ?c)
```

```
      :precondition (and (person ?p) (aircraft ?a) (city ?c) (at ?p ?c)
                         (at ?a ?c))
      :effect (and (in ?p ?a) (not (at ?p ?c))))

   (:action debark
      :parameters ( ?p ?a ?c)
      :precondition (and (person ?p) (aircraft ?a) (city ?c) (in ?p ?a)
                         (at ?a ?c))
      :effect (and (at ?p ?c) (not (in ?p ?a))))

   (:action fly
      :parameters ( ?a ?c1 ?c2 ?l1 ?l2)
      :precondition (and (aircraft ?a) (city ?c1) (city ?c2) (flevel ?l1)
                         (flevel ?l2) (at ?a ?c1) (fuel-level ?a ?l1)
                         (next ?l2 ?l1))
      :effect (and (at ?a ?c2) (fuel-level ?a ?l2)
                   (not (at ?a ?c1)) (not (fuel-level ?a ?l1))))

   (:action zoom
      :parameters ( ?a ?c1 ?c2 ?l1 ?l2 ?l3)
      :precondition (and (aircraft ?a) (city ?c1) (city ?c2) (flevel ?l1)
                         (flevel ?l2) (flevel ?l3)  (at ?a ?c1)
                         (fuel-level ?a ?l1) (next ?l2 ?l1)
                         (next ?l3 ?l2))
       :effect (and (at ?a ?c2) (fuel-level ?a ?l3)
                    (not (at ?a ?c1)) (not (fuel-level ?a ?l1))))

   (:action refuel
      :parameters ( ?a ?c ?l ?l1)
      :precondition (and (aircraft ?a) (city ?c) (flevel ?l) (flevel ?l1)
                         (fuel-level ?a ?l) (next ?l ?l1) (at ?a ?c))
      :effect (and (fuel-level ?a ?l1) (not (fuel-level ?a ?l))))
)
```

A.3. Dominio Driverlog

En este dominio existen paquetes que deben ser transportados a diferentes sitios en camiones. Los conductores para llevar los camiones pueden estar en otros sitios y deben andar hasta llegar a ellos. Los caminos por los que pueden andar los conductores son diferentes de las carreteras por las que pueden circular los camiones, existiendo conexiones diferentes entre ellos representadas por el literal `link` para las carreteras y `path` para los caminos. Este dominio se utilizó en la competición internacional de planificación del año 2002 existiendo varias versiones. La utilizada en la tesis es la versión STRIPS cuya formulación completa en PDDL es:

```
(define (domain driverlog)
    (:requirements :strips)
    (:predicates (obj ?obj)(truck ?truck) (location ?loc) (driver ?d)
                 (at ?obj ?loc) (in ?obj1 ?obj) (driving ?d ?v)
                 (link ?x ?y) (path ?x ?y)(empty ?v))

    (:action load-truck
       :parameters (?obj ?truck ?loc)
```

```
       :precondition (and (obj ?obj) (truck ?truck) (location ?loc)
                          (at ?truck ?loc) (at ?obj ?loc))
       :effect (and (not (at ?obj ?loc)) (in ?obj ?truck)))

    (:action unload-truck
       :parameters (?obj ?truck ?loc)
       :precondition (and (obj ?obj) (truck ?truck) (location ?loc)
                          (at ?truck ?loc) (in ?obj ?truck))
       :effect (and (not (in ?obj ?truck)) (at ?obj ?loc)))

    (:action board-truck
       :parameters (?driver ?truck ?loc)
       :precondition (and (driver ?driver) (truck ?truck) (location ?loc)
                          (at ?truck ?loc) (at ?driver ?loc) (empty ?truck))
       :effect (and (not (at ?driver ?loc)) (driving ?driver ?truck)
                    (not (empty ?truck))))

    (:action disembark-truck
       :parameters (?driver ?truck ?loc)
       :precondition (and (driver ?driver) (truck ?truck) (location ?loc)
                         (at ?truck ?loc) (driving ?driver ?truck))
       :effect (and (not (driving ?driver ?truck))
                    (at ?driver ?loc) (empty ?truck)))

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

    (:action walk
       :parameters (?driver ?loc-from ?loc-to)
       :precondition (and (driver ?driver) (location ?loc-from)
                          (location ?loc-to) (at ?driver ?loc-from)
                          (path ?loc-from ?loc-to))
       :effect (and (not (at ?driver ?loc-from)) (at ?driver ?loc-to)))
)
```

A.4. Dominio Miconic

En este dominio hay ascensores y personas en diferentes plantas cuya meta es subir o bajar en el ascensor a otro piso. El predicado `(served ?person)` indica que la persona ha llegado al piso que deseaba. Este dominio se utilizó en la competición del 2000 y la formulación completa en PDDL es la siguiente:

```
(define (domain miconic)
    (:requirements :strips)
    (:predicates (origin ?person ?floor) ;; entry of ?person is ?floor
                 (floor ?floor) (passenger ?passenger)
                 (destin ?person ?floor) ;; exit of ?person is ?floor
                 (above ?floor1 ?floor2)
                 ;; ?floor2 is located above of ?floor1
                 (boarded ?person)
```

```
                ;; true if ?person has boarded the lift
                (served ?person)
                ;; true if ?person has alighted as her destination
                (lift-at ?floor)
                ;; current position of the lift is at ?floor
    )
    ;;stop and allow boarding
    (:action board
       :parameters (?f ?p)
       :precondition (and (floor ?f) (passenger ?p) (lift-at ?f) (origin ?p ?f))
       :effect (boarded ?p))

    (:action depart
       :parameters (?f  ?p)
       :precondition (and (floor ?f) (passenger ?p) (lift-at ?f) (destin ?p ?f)
                          (boarded ?p))
       :effect (and (not (boarded ?p)) (served ?p)))
    ;;drive up
    (:action up
       :parameters (?f1 ?f2)
       :precondition (and (floor ?f1) (floor ?f2) (lift-at ?f1)
                          (above ?f1 ?f2))
       :effect (and (lift-at ?f2) (not (lift-at ?f1))))
    ;;drive down
    (:action down
       :parameters (?f1 ?f2)
       :precondition (and (floor ?f1) (floor ?f2) (lift-at ?f1)
                          (above ?f2 ?f1))
       :effect (and (lift-at ?f2) (not (lift-at ?f1))))
)
```

A.5. Dominio Blocks

Es el clásico dominio del mundo de los bloques en los que hay bloques que pueden estar encima de una mesa o sobre otro bloque y un brazo de un robot para moverlos de posición de uno en uno. Los problemas consisten en pasar de una configuración inicial de un conjunto de bloques dada a otra configuración final. La versión utilizada en la tesis utiliza cuatro operadores: dos para apilar/desapilar bloques y dos para coger/dejar de/en la mesa. La formulación completa en lenguaje PDDL ES:

```
(define (domain blocks)
    (:requirements :strips)
    (:predicates (on ?x ?y) (ontable ?x) (clear ?x) (handempty) (holding ?x))

    (:action pick-up
       :parameters (?x)
       :precondition (and (clear ?x) (ontable ?x) (handempty))
       :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty))
                    (holding ?x)))

    (:action put-down
       :parameters (?x)
```

```
      :precondition (holding ?x)
      :effect(and (not (holding ?x)) (clear ?x) (handempty) (ontable ?x)))

   (:action stack
      :parameters (?x ?y)
      :precondition (and (holding ?x) (clear ?y))
      :effect (and (not (holding ?x)) (not (clear ?y)) (clear ?x)
                   (handempty) (on ?x ?y)))

   (:action unstack
      :parameters (?x ?y)
      :precondition (and (on ?x ?y) (clear ?x) (handempty))
      :effect (and (holding ?x) (clear ?y) (not (clear ?x))
                   (not (handempty)) (not (on ?x ?y)))))
```

A.6. Dominio Satellite

En este domino hay satélites con instrumentos a bordo para tomar imágenes de diferentes fenómenos. Para tomar una imagen, el satélite debe encender el instrumento adecuado, teniendo en cuenta que sólo puede haber un instrumento encendido al mismo tiempo, calibrar el instrumento para lo cual debe girar hasta apuntar a la dirección de calibración, volver a girar hasta apuntar al fenómeno y por último tomar la imagen. Este dominio se utilizó en la competición del año 2002 y la versión utilizada en esta tesis es la versión STRIPS que se muestra a continuación:

```
(define (domain satellite)
    (:requirements :equality :strips)
    (:predicates (on_board ?i ?s) (supports ?i ?m) (pointing ?s ?d)
                 (power_avail ?s) (power_on ?i) (calibrated ?i)
                 (have_image ?d ?m) (calibration_target ?i ?d)
                 (satellite ?x) (direction ?x) (instrument ?x) (mode ?x))

    (:action turn_to
       :parameters ( ?s ?d_new ?d_prev)
       :precondition (and (satellite ?s) (direction ?d_new) (direction ?d_prev)
                          (pointing ?s ?d_prev) (not (= ?d_new ?d_prev)))
       :effect (and (pointing ?s ?d_new) (not (pointing ?s ?d_prev))))

    (:action switch_on
       :parameters ( ?i ?s)
       :precondition (and (instrument ?i) (satellite ?s) (on_board ?i ?s)
                          (power_avail ?s))
       :effect (and (power_on ?i)
                    (not (calibrated ?i)) (not (power_avail ?s))))

    (:action switch_off
       :parameters ( ?i ?s)
       :precondition (and (instrument ?i) (satellite ?s) (on_board ?i ?s)
                          (power_on ?i))
       :effect (and (power_avail ?s) (not (power_on ?i))))

    (:action calibrate
       :parameters ( ?s ?i ?d)
```

```
       :precondition (and (satellite ?s) (instrument ?i) (direction ?d)
                          (on_board ?i ?s) (calibration_target ?i ?d)
                          (pointing ?s ?d) (power_on ?i))
       :effect (calibrated ?i))

    (:action take_image
       :parameters ( ?s ?d ?i ?m)
       :precondition (and (satellite ?s) (direction ?d) (instrument ?i)
                          (mode ?m) (calibrated ?i) (on_board ?i ?s)
                          (supports ?i ?m) (power_on ?i) (pointing ?s ?d)
                          (power_on ?i))
       :effect (have_image ?d ?m))

)
```

A.7. Dominio Robocare

Este dominio no ha sido utilizado en ninguna competición pero se utiliza aquí para estudiar la capacidad de los sistemas estudiados de generar planes paralelos. El dominio consiste en unos robots que hacen las camas de un hospital. Para que un robot pueda hacer una cama se debe encontrar en la misma habitación y la cama debe estar limpia.

```
(define ( domain robocare)
    (:requirements :strips :equality)
    (:predicates (agent ?a) (object ?o) (person ?p) (room ?r) (door ?d)
                 (bed ?b) (is_in_agent_room ?a0 ?r0) (is_door ?r0 ?r1)
                 (unmade ?b0) (clear ?b0) (is_in_bed_room ?b0 ?r0)
                 (made ?b0) (not-clear ?b0)
    )

;; mueve el agente de un sitio a otro. debe estar el primero
;; de lo contrario cogerá el operador walk_person y no encuentra
;; la solución.

    (:action goto_room
       :parameters (?a0 ?r0 ?r1)
       :precondition (and (agent ?a0) (room ?r0) (room ?r1)
                          (is_in_agent_room ?a0 ?r0) (is_door ?r0 ?r1))
        :effect (and (is_in_agent_room ?a0 ?r1)
                     (not (is_in_agent_room ?a0 ?r0))))

;;  si la cama no esta hecha, la hace. la cama debe estar libre,
;;  el agente en la misma habitación que en la cama.

    (:action r_make_bed
        :parameters (?a0 ?b0 ?r0)
        :precondition (and (agent ?a0) (bed ?b0) (room ?r0) (unmade ?b0)
                           (clear ?b0) (is_in_bed_room ?b0 ?r0)
                           (is_in_agent_room ?a0 ?r0))
        :effect (and (not (unmade ?b0)) (made ?b0)))

;; para hacer la cama antes hay que limpiarla.
```

```
    (:action r_clear_bed
       :parameters (?a0 ?b0 ?r0)
       :precondition (and (agent ?a0) (bed ?b0) (room ?r0) (not-clear ?b0)
                          (is_in_bed_room ?b0 ?r0)(is_in_agent_room ?a0 ?r0))
        :effect (and (clear ?b0) (not (not-clear ?b0))))
    )
```

A.8. Dominio Depots

El dominio Depots es una mezcla del dominio Logistics y el Blocks. Hay palés que pueden ser colocados en camiones, por medio de grúas que los apilan, y deben ser distribuidos entre diferentes almacenes. Se utilizó en la competición del año 2002 y su formulación en PDDL es.

```
(define (domain depots)
    (:predicates (at ?x ?y) (on ?x ?y) (in ?x ?y) (lifting ?x ?y)
                 (available ?x) (clear ?x) (place ?x) (depot ?x)
                 (distributor ?x) (truck ?x) (hoist ?x)
                 (surface ?x) (pallet ?x) (crate ?x))

    (:action drive
       :parameters ( ?x ?y ?z)
       :precondition (and (truck ?x) (place ?y) (place ?z)  (at ?x ?y))
       :effect (and (at ?x ?z) (not (at ?x ?y))))

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

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

    (:action load
       :parameters ( ?x ?y ?z ?p)
       :precondition (and (hoist ?x) (crate ?y) (truck ?z) (place ?p)
                          (at ?x ?p) (at ?z ?p) (lifting ?x ?y))
       :effect (and (in ?y ?z) (available ?x) (not (lifting ?x ?y))))

    (:action unload
       :parameters ( ?x ?y ?z ?p)
       :precondition (and (hoist ?x) (crate ?y) (truck ?z) (place ?p)
                          (at ?x ?p) (at ?z ?p) (available ?x) (in ?y ?z))
       :effect (and (lifting ?x ?y) (not (in ?y ?z)) (not (available ?x))))

)
```

Apéndice B

Reglas aprendidas por GEBL para transferir a PRODIGY

En este apéndice se muestran las reglas más significativas aprendidas por GEBL para transferir a PRODIGY utilizadas para hacer el análisis del conocimiento de control discutido en la sección 9.5.

B.1. Dominio Zenotravel

A continuación se muestran las 14 reglas aprendidas en este dominio.

```
(control-rule regla-zeno-travel-l154-02-e1-binds
  (if (and (current-operator debark)
           (current-goal (at <person1> <city1>))
           (true-in-state (next <fl0> <fl1>))
           (true-in-state (fuel-level <plane1> <fl0>))
           (true-in-state (at <plane1> <city0>))
           (true-in-state (at <person1> <city0>))
           (type-of-object <plane1> aircraft)
           (type-of-object <city0> city)
           (type-of-object <city1> city)
           (type-of-object <fl0> flevel)
           (type-of-object <fl1> flevel)
           (type-of-object <person1> person)))
  (then select bindings
        ((<p> . <person1>) (<a> . <plane1>) (<c> . <city1>))))

(control-rule regla-zeno-travel-l192-03-e6-binds
  (if (and (current-operator board)
           (current-goal (in <person1> <plane1>))
           (true-in-state (at <plane1> <city0>))
           (true-in-state (at <person1> <city0>))
           (some-candidate-goals ((fuel-level <plane1> <fl1>)
                                  (in <person2> <plane1>)))
           (type-of-object <plane1> aircraft)
           (type-of-object <city0> city)
```

```
          (type-of-object <fl1> flevel)
          (type-of-object <person1> person)
          (type-of-object <person2> person)))
  (then select bindings
        ((<p> . <person1>) (<a> . <plane1>) (<c> . <city0>))))

(control-rule regla-zeno-travel-l198-03-e1-binds
  (if (and (current-operator debark)
          (current-goal (at <person1> <city1>))
          (true-in-state (at <plane1> <city1>))
          (true-in-state (in <person1> <plane1>))
          (type-of-object <person1> person)
          (type-of-object <city1> city)
          (type-of-object <plane1> aircraft)))
  (then select bindings
        ((<p> . <person1>) (<a> . <plane1>) (<c> . <city1>))))

(control-rule regla-zeno-travel-l198-03-e4-binds
  (if (and (current-operator board)
          (current-goal (in <person1> <plane1>))
          (true-in-state (at <person1> <city0>))
          (true-in-state (at <plane1> <city0>))
          (some-candidate-goals ((fuel-level <plane1> <fl1>)))
          (type-of-object <person1> person)
          (type-of-object <fl1> flevel)
          (type-of-object <city0> city)
          (type-of-object <plane1> aircraft)))
  (then select bindings
        ((<p> . <person1>) (<a> . <plane1>) (<c> . <city0>))))

(control-rule regla-zeno-travel-l200-03-e1-binds
  (if (and (current-operator fly)
          (current-goal (at <plane1> <city0>))
          (true-in-state (next <fl0> <fl1>))
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (at <plane1> <city1>))
          (type-of-object <plane1> aircraft)
          (type-of-object <city0> city)
          (type-of-object <city1> city)
          (type-of-object <fl0> flevel)
          (type-of-object <fl1> flevel)))
  (then select bindings
        ((<a> . <plane1>) (<c1> . <city1>) (<c2> . <city0>)
         (<l1> . <fl1>) (<l2> . <fl0>))))

(control-rule regla-zeno-travel-l200-03-e2-binds
  (if (and (current-operator refuel)
          (current-goal (fuel-level <plane1> <fl1>))
          (true-in-state (at <plane1> <city1>))
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (next <fl0> <fl1>))
          (type-of-object <fl1> flevel)
          (type-of-object <fl0> flevel)
          (type-of-object <city1> city)
          (type-of-object <plane1> aircraft)))
  (then select bindings
        ((<a> . <plane1>) (<c> . <city1>) (<l> . <fl0>)
         (<l1> . <fl1>))))

(control-rule regla-zeno-travel-l154-02-e1
  (if (and (current-goal (at <person1> <city1>))
          (true-in-state (next <fl0> <fl1>))
```

```
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (at <plane1> <city0>))
          (true-in-state (at <person1> <city0>))
          (type-of-object <plane1> aircraft)
          (type-of-object <city0> city)
          (type-of-object <city1> city)
          (type-of-object <fl0> flevel)
          (type-of-object <fl1> flevel)
          (type-of-object <person1> person)))
  (then select operators debark))

(control-rule regla-zeno-travel-l192-03-e6
  (if (and (current-goal (in <person1> <plane1>))
          (true-in-state (at <plane1> <city0>))
          (true-in-state (at <person1> <city0>))
          (some-candidate-goals ((fuel-level <plane1> <fl1>)
                                 (in <person2> <plane1>)))
          (type-of-object <plane1> aircraft)
          (type-of-object <city0> city)
          (type-of-object <fl1> flevel)
          (type-of-object <person1> person)
          (type-of-object <person2> person)))
  (then select operators board))

(control-rule regla-zeno-travel-l198-03-e1
  (if (and (current-goal (at <person1> <city1>))
          (true-in-state (in <person1> <plane1>))
          (true-in-state (at <plane1> <city1>))
          (type-of-object <plane1> aircraft)
          (type-of-object <city1> city)
          (type-of-object <person1> person)))
  (then select operators debark))

(control-rule regla-zeno-travel-l198-03-e4
  (if (and (current-goal (in <person1> <plane1>))
          (true-in-state (at <person1> <city0>))
          (true-in-state (at <plane1> <city0>))
          (some-candidate-goals ((fuel-level <plane1> <fl1>)))
          (type-of-object <person1> person)
          (type-of-object <fl1> flevel)
          (type-of-object <city0> city)
          (type-of-object <plane1> aircraft)))
  (then select operators board))

(control-rule regla-zeno-travel-l200-03-e1
  (if (and (current-goal (at <plane1> <city0>))
          (true-in-state (next <fl0> <fl1>))
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (at <plane1> <city1>))
          (type-of-object <plane1> aircraft)
          (type-of-object <city0> city)
          (type-of-object <city1> city)
          (type-of-object <fl0> flevel)
          (type-of-object <fl1> flevel)))
  (then select operators fly))

(control-rule regla-zeno-travel-l200-03-e2
  (if (and (current-goal (fuel-level <plane1> <fl1>))
          (true-in-state (next <fl0> <fl1>))
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (at <plane1> <city1>))
          (type-of-object <plane1> aircraft)
```

```
          (type-of-object <city1> city)
          (type-of-object <fl0> flevel)
          (type-of-object <fl1> flevel)))
  (then select operators refuel))

(control-rule regla-zeno-travel-l004-01-s1
  (if (and (target-goal (in <person1> <plane1>))
          (true-in-state (at <person1> <city0>))
          (true-in-state (at <plane1> <city0>))
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (next <fl0> <fl1>))
          (some-candidate-goals ((at <plane1> <city1>)))
          (type-of-object <person1> person)
          (type-of-object <fl1> flevel)
          (type-of-object <fl0> flevel)
          (type-of-object <city1> city)
          (type-of-object <city0> city)
          (type-of-object <plane1> aircraft)))
  (then select goals (in <person1> <plane1>)))

(control-rule regla-zeno-travel-l192-03-s1
  (if (and (target-goal (at <plane1> <city1>))
          (true-in-state (next <fl0> <fl1>))
          (true-in-state (fuel-level <plane1> <fl0>))
          (true-in-state (at <plane1> <city0>))
          (true-in-state (at <person1> <city0>))
          (true-in-state (at <person2> <city0>))
          (some-candidate-goals ((at <person2> <city1>)
                                 (at <person1> <city1>)))
          (type-of-object <plane1> aircraft)
          (type-of-object <city0> city)
          (type-of-object <city1> city)
          (type-of-object <fl0> flevel)
          (type-of-object <fl1> flevel)
          (type-of-object <person1> person)
          (type-of-object <person2> person)))
  (then select goals (at <plane1> <city1>)))
```

B.2. Dominio Driverlog

Las reglas aprendidas para seleccionar los *bindings* del operador `unload-truck` son:

```
1 (control-rule regla-driverlog-l022-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s2>))
            (true-in-state (in <package1> <truck1>))
            (true-in-state (at <truck1> <s2>))
            (type-of-object <s2> location)
            (type-of-object <package1> obj)
            (type-of-object <truck1> truck)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s2>))))

2 (control-rule regla-driverlog-l004-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package2> <s2>))
```

```
            (true-in-state (at <truck1> <s0>))
            (true-in-state (at <package2> <s0>))
            (true-in-state (driving <driver1> <truck1>))
            (true-in-state (link <s0> <s2>))
            (type-of-object <truck1> truck)
            (type-of-object <package2> obj)
            (type-of-object <s2> location)
            (type-of-object <s0> location)
            (type-of-object <driver1> driver)))
   (then select bindings
    ((<obj> . <package2>) (<truck> . <truck1>)
     (<loc> . <s2>))))

3 (control-rule regla-driverlog-l013-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package2> <s2>))
            (true-in-state (link <s1> <s2>))
            (true-in-state (in <package2> <truck1>))
            (true-in-state (driving <driver1> <truck1>))
            (true-in-state (at <truck1> <s1>))
            (type-of-object <driver1> driver)
            (type-of-object <s1> location)
            (type-of-object <s2> location)
            (type-of-object <package2> obj)
            (type-of-object <truck1> truck)))
   (then select bindings
    ((<obj> . <package2>) (<truck> . <truck1>)
     (<loc> . <s2>))))

4 (control-rule regla-driverlog-l023-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s0>))
            (true-in-state (at <driver1> <s1>))
            (true-in-state (at <truck1> <s1>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (link <s1> <s0>))
            (type-of-object <truck1> truck)
            (type-of-object <package1> obj)
            (type-of-object <s1> location)
            (type-of-object <s0> location)
            (type-of-object <driver1> driver)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s0>))))



5 (control-rule regla-driverlog-l009-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s2>))
            (true-in-state (path <p1-2> <s2>))
            (true-in-state (path <s1> <p1-2>))
            (true-in-state (link <s2> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (at <driver1> <s1>))
            (true-in-state (at <truck1> <s2>))
            (type-of-object <driver1> driver)
            (type-of-object <p1-2> location)
            (type-of-object <s1> location)
            (type-of-object <s2> location)
```

```
            (type-of-object <package1> obj)
            (type-of-object <truck1> truck)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s2>))))

6 (control-rule regla-driverlog-l014-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s0>))
            (true-in-state (at <driver1> <s0>))
            (true-in-state (at <truck1> <s1>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (link <s1> <s0>))
            (true-in-state (path <s0> <p1-0>))
            (true-in-state (path <p1-0> <s1>))
            (type-of-object <truck1> truck)
            (type-of-object <package1> obj)
            (type-of-object <s1> location)
            (type-of-object <s0> location)
            (type-of-object <p1-0> location)
            (type-of-object <driver1> driver)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s0>))))

7 (control-rule regla-driverlog-l017-02-e2-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s2>))
            (true-in-state (link <s0> <s2>))
            (true-in-state (link <s1> <s0>))
            (true-in-state (empty <truck1>))
            (true-in-state (at <driver1> <s1>))
            (true-in-state (at <truck1> <s1>))
            (true-in-state (at <package1> <s0>))
            (some-candidate-goals ((at <driver1> <s2>)))
            (type-of-object <driver1> driver)
            (type-of-object <s0> location)
            (type-of-object <s1> location)
            (type-of-object <s2> location)
            (type-of-object <package1> obj)
            (type-of-object <truck1> truck)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s2>))))

8 (control-rule regla-driverlog-l018-02-e2-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s0>))
            (true-in-state (link <s1> <s0>))
            (true-in-state (link <s2> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (at <driver1> <s2>))
            (true-in-state (at <truck1> <s2>))
            (true-in-state (at <package1> <s1>))
            (some-candidate-goals ((at <driver1> <s0>)))
            (type-of-object <driver1> driver)
            (type-of-object <s0> location)
            (type-of-object <s1> location)
            (type-of-object <s2> location)
            (type-of-object <package1> obj)
            (type-of-object <truck1> truck)))
```

```
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s0>))))

9 (control-rule regla-driverlog-l015-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package2> <s1>))
            (true-in-state (path <p0-2> <s2>))
            (true-in-state (path <s0> <p0-2>))
            (true-in-state (link <s2> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (at <package2> <s2>))
            (true-in-state (at <truck1> <s2>))
            (true-in-state (at <driver1> <s0>))
            (type-of-object <driver1> driver)
            (type-of-object <p0-2> location)
            (type-of-object <s0> location)
            (type-of-object <s1> location)
            (type-of-object <s2> location)
            (type-of-object <package2> obj)
            (type-of-object <truck1> truck)))
   (then select bindings
    ((<obj> . <package2>) (<truck> . <truck1>)
     (<loc> . <s1>))))

10 (control-rule regla-driverlog-l016-02-e1-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s2>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (at <driver1> <s0>))
            (true-in-state (at <truck1> <s0>))
            (true-in-state (at <package2> <s0>))
            (true-in-state (empty <truck1>))
            (true-in-state (link <s0> <s1>))
            (true-in-state (link <s1> <s2>))
            (some-candidate-goals ((at <package2> <s2>)))
            (type-of-object <truck1> truck)
            (type-of-object <package2> obj)
            (type-of-object <package1> obj)
            (type-of-object <s2> location)
            (type-of-object <s1> location)
            (type-of-object <s0> location)
            (type-of-object <driver1> driver)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s2>))))

11 (control-rule regla-driverlog-l016-02-e2-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package2> <s2>))
            (true-in-state (at <package2> <s0>))
            (true-in-state (at <truck1> <s0>))
            (true-in-state (at <driver1> <s0>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (link <s1> <s2>))
            (true-in-state (link <s0> <s1>))
            (some-candidate-goals ((at <package1> <s2>)))
            (type-of-object <truck1> truck)
            (type-of-object <package2> obj)
            (type-of-object <package1> obj)
            (type-of-object <s2> location)
```

```
            (type-of-object <s1> location)
            (type-of-object <s0> location)
            (type-of-object <driver1> driver)))
   (then select bindings
    ((<obj> . <package2>) (<truck> . <truck1>)
     (<loc> . <s2>))))

12 (control-rule regla-driverlog-l008-02-e2-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package1> <s0>))
            (true-in-state (path <p0-2> <s2>))
            (true-in-state (path <s0> <p0-2>))
            (true-in-state (link <s2> <s1>))
            (true-in-state (link <s1> <s0>))
            (true-in-state (empty <truck1>))
            (true-in-state (at <driver1> <s0>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (at <truck1> <s2>))
            (some-candidate-goals ((at <driver1> <s0>)))
            (type-of-object <driver1> driver)
            (type-of-object <p0-2> location)
            (type-of-object <s0> location)
            (type-of-object <s1> location)
            (type-of-object <s2> location)
            (type-of-object <package1> obj)
            (type-of-object <truck1> truck)))
   (then select bindings
    ((<obj> . <package1>) (<truck> . <truck1>)
     (<loc> . <s0>))))

13 (control-rule regla-driverlog-l006-02-e2-binds
   (if (and (current-operator unload-truck)
            (current-goal (at <package2> <s2>))
            (true-in-state (at <package2> <s1>))
            (true-in-state (at <truck1> <s2>))
            (true-in-state (at <driver1> <s0>))
            (true-in-state (at <package1> <s1>))
            (true-in-state (empty <truck1>))
            (true-in-state (link <s2> <s1>))
            (true-in-state (path <s0> <p0-2>))
            (true-in-state (path <p0-2> <s2>))
            (some-candidate-goals ((at <package1> <s2>)))
            (type-of-object <truck1> truck)
            (type-of-object <package2> obj)
            (type-of-object <package1> obj)
            (type-of-object <s2> location)
            (type-of-object <s1> location)
            (type-of-object <s0> location)
            (type-of-object <p0-2> location)
            (type-of-object <driver1> driver)))
   (then select bindings
    ((<obj> . <package2>) (<truck> . <truck1>)
     (<loc> . <s2>))))
```

Las reglas de selección de metas aprendidas son:

```
(control-rule REGLA-DRIVERLOG-L001-02-S1
  (if (and (target-goal (at <package2> <s2>))
           (true-in-state (at <package2> <s0>))
           (true-in-state (at <truck1> <s1>))
```

```
           (true-in-state (at <driver1> <s0>))
           (true-in-state (at <package1> <s0>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s0> <s2>))
           (true-in-state (link <s1> <s0>))
           (true-in-state (path <s0> <p1-0>))
           (true-in-state (path <p1-0> <s1>))
           (some-candidate-goals ((at <package1> <s1>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p1-0> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package2> <s2>)))

(control-rule REGLA-DRIVERLOG-L002-02-S1
  (if (and (target-goal (at <package2> <s0>))
           (true-in-state (at <package2> <s2>))
           (true-in-state (at <truck1> <s1>))
           (true-in-state (at <driver1> <s0>))
           (true-in-state (at <package1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s2> <s0>))
           (true-in-state (link <s1> <s2>))
           (true-in-state (path <s0> <p0-1>))
           (true-in-state (path <p0-1> <s1>))
           (some-candidate-goals ((at <package1> <s1>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p0-1> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package2> <s0>)))

(control-rule REGLA-DRIVERLOG-L003-02-S1
  (if (and (target-goal (at <package2> <s2>))
           (true-in-state (at <package2> <s1>))
           (true-in-state (at <truck1> <s1>))
           (true-in-state (at <driver1> <s2>))
           (true-in-state (at <package1> <s0>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s0> <s2>))
           (true-in-state (link <s1> <s0>))
           (true-in-state (path <s2> <p2-1>))
           (true-in-state (path <p2-1> <s1>))
           (some-candidate-goals ((at <package1> <s1>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p2-1> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package2> <s2>)))
```

```
(control-rule REGLA-DRIVERLOG-L004-02-S1
  (if (and (target-goal (at <package1> <s0>))
           (true-in-state (at <package1> <s1>))
           (true-in-state (at <truck1> <s2>))
           (true-in-state (at <driver1> <s1>))
           (true-in-state (at <package2> <s0>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s2> <s1>))
           (true-in-state (link <s1> <s0>))
           (true-in-state (path <s1> <p1-2>))
           (true-in-state (path <p1-2> <s2>))
           (some-candidate-goals ((at <package2> <s2>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p1-2> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package1> <s0>)))

(control-rule REGLA-DRIVERLOG-L005-02-S1
  (if (and (target-goal (at <package1> <s2>))
           (true-in-state (at <truck1> <s2>))
           (true-in-state (at <package1> <s0>))
           (true-in-state (at <driver1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s2> <s0>))
           (true-in-state (path <s2> <p2-1>))
           (true-in-state (path <p2-1> <s1>))
           (some-candidate-goals ((at <driver1> <s1>)))
           (type-of-object <truck1> truck)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p2-1> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package1> <s2>)))

(control-rule REGLA-DRIVERLOG-L005-02-S2
  (if (and (target-goal (at <package1> <s2>))
           (true-in-state (at <truck1> <s2>))
           (true-in-state (at <package1> <s0>))
           (true-in-state (at <driver1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s2> <s0>))
           (true-in-state (path <s2> <p2-1>))
           (some-candidate-goals ((at <driver1> <p2-1>)))
           (type-of-object <truck1> truck)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s0> location)
           (type-of-object <p2-1> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package1> <s2>)))

(control-rule REGLA-DRIVERLOG-L010-02-S1
  (if (and (target-goal (at <truck1> <s0>))
           (true-in-state (at <driver1> <s1>))
           (true-in-state (at <truck1> <s2>))
```

```
           (true-in-state (at <package1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s2> <s0>))
           (true-in-state (path <s1> <p1-0>))
           (true-in-state (path <p1-0> <s0>))
           (true-in-state (path <s0> <p0-2>))
           (true-in-state (path <p0-2> <s2>))
           (some-candidate-goals ((at <package1> <s0>)))
           (type-of-object <truck1> truck)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p1-0> location)
           (type-of-object <p0-2> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <truck1> <s0>)))

(control-rule REGLA-DRIVERLOG-L010-02-S2
  (if (and (target-goal (in <package1> <truck1>))
           (true-in-state (path <p0-2> <s2>))
           (true-in-state (path <s0> <p0-2>))
           (true-in-state (path <p1-0> <s0>))
           (true-in-state (path <s1> <p1-0>))
           (true-in-state (link <s2> <s0>))
           (true-in-state (empty <truck1>))
           (true-in-state (at <driver1> <s1>))
           (true-in-state (at <truck1> <s2>))
           (true-in-state (at <package1> <s2>))
           (some-candidate-goals ((at <truck1> <s0>)))
           (type-of-object <driver1> driver)
           (type-of-object <p0-2> location)
           (type-of-object <p1-0> location)
           (type-of-object <s0> location)
           (type-of-object <s1> location)
           (type-of-object <s2> location)
           (type-of-object <package1> obj)
           (type-of-object <truck1> truck)))
  (then select goals (in <package1> <truck1>)))

(control-rule REGLA-DRIVERLOG-L011-02-S1
  (if (and (target-goal (at <package2> <s2>))
           (true-in-state (at <package2> <s0>))
           (true-in-state (at <truck1> <s0>))
           (true-in-state (at <driver1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s0> <s2>))
           (true-in-state (link <s2> <s1>))
           (true-in-state (path <s2> <p2-0>))
           (true-in-state (path <p2-0> <s0>))
           (some-candidate-goals ((at <truck1> <s1>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p2-0> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package2> <s2>)))

(control-rule REGLA-DRIVERLOG-L012-02-S1
  (if (and (target-goal (at <package2> <s0>))
```

```
           (true-in-state (at <package2> <s1>))
           (true-in-state (at <truck1> <s1>))
           (true-in-state (at <driver1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s1> <s0>))
           (true-in-state (path <s2> <p1-2>))
           (true-in-state (path <p1-2> <s1>))
           (some-candidate-goals ((at <driver1> <s0>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <p1-2> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package2> <s0>)))

(control-rule REGLA-DRIVERLOG-L013-02-S1
  (if (and (target-goal (at <package1> <s1>))
           (true-in-state (at <package1> <s0>))
           (true-in-state (at <driver1> <s0>))
           (true-in-state (at <truck1> <s0>))
           (true-in-state (at <package2> <s0>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s0> <s1>))
           (true-in-state (link <s1> <s2>))
           (some-candidate-goals ((at <package2> <s2>)))
           (type-of-object <truck1> truck)
           (type-of-object <package2> obj)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <s0> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <package1> <s1>)))

(control-rule REGLA-DRIVERLOG-L022-02-S1
  (if (and (target-goal (at <truck1> <s2>))
           (true-in-state (at <driver1> <s1>))
           (true-in-state (at <truck1> <s1>))
           (true-in-state (at <package1> <s1>))
           (true-in-state (empty <truck1>))
           (true-in-state (link <s1> <s2>))
           (some-candidate-goals ((at <package1> <s2>)))
           (type-of-object <truck1> truck)
           (type-of-object <package1> obj)
           (type-of-object <s2> location)
           (type-of-object <s1> location)
           (type-of-object <driver1> driver)))
  (then select goals (at <truck1> <s2>)))

(control-rule REGLA-DRIVERLOG-L022-02-S2
  (if (and (target-goal (in <package1> <truck1>))
           (true-in-state (link <s1> <s2>))
           (true-in-state (empty <truck1>))
           (true-in-state (at <driver1> <s1>))
           (true-in-state (at <truck1> <s1>))
           (true-in-state (at <package1> <s1>))
           (some-candidate-goals ((at <truck1> <s2>)))
           (type-of-object <driver1> driver)
           (type-of-object <s1> location)
           (type-of-object <s2> location)
```

```
          (type-of-object <package1> obj)
          (type-of-object <truck1> truck)))
 (then select goals (in <package1> <truck1>)))
))
```

B.3. Dominio Logistics

```
(control-rule regla-logistics-aips98-l2-5-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (inside <ob0> <tr1>))
          (true-in-state (at <tr1> <po1>))
          (type-of-object <ob0> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-42-e4-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <a1>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (inside <ob0> <tr1>))
          (type-of-object <tr1> truck) (type-of-object <ob0> object)
          (type-of-object <a1> airport)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <a1>))))

(control-rule regla-logistics-aips98-l3-9-e2-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <po1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob3> <tr1>))
          (some-candidate-goals ((at <ob1> <po0>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob3> object) (type-of-object <ob1> object)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-15-e11-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (inside <ob0> <tr1>))
          (true-in-state (at <tr1> <po1>))
          (some-candidate-goals ((inside <ob2> <tr0>)))
          (type-of-object <ob0> object) (type-of-object <ob2> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-29-e7-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob4> <po0>))
          (true-in-state (inside <ob4> <tr0>))
          (true-in-state (at <tr0> <po0>))
          (some-candidate-goals ((at <pl0> <a0>)))
```

```
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <ob4> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob4>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-2-e2-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob0> <tr1>))
          (some-candidate-goals ((at <ob2> <a1>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob2> object) (type-of-object <ob0> object)
          (type-of-object <a1> airport)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-44-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po0>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (inside <ob2> <tr0>))
          (some-candidate-goals ((at <ob1> <a1>)))
          (type-of-object <tr0> truck)
          (type-of-object <po0> post-office)
          (type-of-object <ob2> object) (type-of-object <ob1> object)
          (type-of-object <a1> airport)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-11-e7-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <po1>))
          (true-in-state (inside <ob3> <tr1>))
          (true-in-state (at <tr1> <po1>))
          (some-candidate-goals ((at <tr0> <a0>) (at <pl0> <a0>)))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <ob3> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-12-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <a0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (inside <ob0> <tr0>))
          (true-in-state (at <tr0> <po0>))
          (type-of-object <a0> airport) (type-of-object <c0> city)
          (type-of-object <ob0> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <a0>))))

(control-rule regla-logistics-aips98-l4-28-e1-binds
 (if (and (current-operator unload-truck)
```

```
          (current-goal (at <ob1> <po0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (inside <ob1> <tr0>))
          (true-in-state (at <tr0> <a0>)) (type-of-object <a0> airport)
          (type-of-object <c0> city) (type-of-object <ob1> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-37-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <ob2> <a0>)) (type-of-object <a0> airport)
          (type-of-object <c0> city) (type-of-object <ob2> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-41-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <a0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <ob3> <po0>))
          (type-of-object <a0> airport) (type-of-object <c0> city)
          (type-of-object <ob3> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr0>) (<loc> . <a0>))))

(control-rule regla-logistics-aips98-l4-7-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <a1>))
          (true-in-state (at <ob0> <po1>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob0> object) (type-of-object <c1> city)
          (type-of-object <a1> airport)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <a1>))))

(control-rule regla-logistics-aips98-l4-23-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po0>))
          (true-in-state (at <pl1> <a0>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (inside <ob0> <pl1>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (type-of-object <tr0> truck)
          (type-of-object <po0> post-office)
```

```
          (type-of-object <ob0> object) (type-of-object <c0> city)
          (type-of-object <a0> airport)
          (type-of-object <pl1> airplane)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-30-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (inside <ob1> <pl0>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <pl0> <a1>))
          (type-of-object <pl0> airplane) (type-of-object <a1> airport)
          (type-of-object <c1> city) (type-of-object <ob1> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l2-29-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <ob0> <a1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <ob0> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l3-43-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <ob1> <a0>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c1> city)
          (type-of-object <ob1> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l3-8-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <ob0> <a1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
```

```
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <ob0> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-46-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob0> <pl0>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob0> object) (type-of-object <c1> city)
          (type-of-object <a1> airport) (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-48-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <po0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (inside <ob3> <pl1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl1> <a1>))
          (type-of-object <pl1> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <ob3> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-49-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po0>))
          (true-in-state (at <ob1> <a1>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (type-of-object <tr0> truck)
          (type-of-object <po0> post-office)
          (type-of-object <ob1> object) (type-of-object <c0> city)
          (type-of-object <a1> airport) (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-8-e3-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <po1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (inside <ob3> <pl0>))
          (true-in-state (at <pl0> <a1>))
```

```
          (true-in-state (at <tr1> <a1>))
          (some-candidate-goals ((at <pl0> <a1>)))
          (type-of-object <pl0> airplane) (type-of-object <a1> airport)
          (type-of-object <c1> city) (type-of-object <ob3> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l3-15-e6-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po1>))
          (true-in-state (at <ob2> <a1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (some-candidate-goals ((at <pl0> <a0>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob2> object) (type-of-object <c1> city)
          (type-of-object <a1> airport) (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-31-e8-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <a0>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (inside <ob2> <tr0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (some-candidate-goals ((at <pl0> <a0>)))
          (type-of-object <tr0> truck)
          (type-of-object <po0> post-office)
          (type-of-object <ob2> object) (type-of-object <c0> city)
          (type-of-object <a1> airport) (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr0>) (<loc> . <a0>))))

(control-rule regla-logistics-aips98-l2-43-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po1>))
          (true-in-state (at <ob2> <a1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob1> <pl0>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (some-candidate-goals ((at <ob1> <po1>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob2> object) (type-of-object <ob1> object)
          (type-of-object <c1> city) (type-of-object <a1> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l3-22-e1-binds
```

```
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob0> <tr0>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob0> object) (type-of-object <c1> city)
          (type-of-object <a1> airport) (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-25-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (inside <ob2> <tr1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <a1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <ob2> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-3-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (inside <ob0> <tr0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob0> object) (type-of-object <c1> city)
          (type-of-object <a1> airport) (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-4-e10-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <a1>))
          (true-in-state (at <ob2> <po1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (inside <ob1> <pl0>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (some-candidate-goals ((inside <ob1> <tr1>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
```

```
          (type-of-object <ob2> object) (type-of-object <ob1> object)
          (type-of-object <c1> city) (type-of-object <a1> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr1>) (<loc> . <a1>))))

(control-rule regla-logistics-aips98-l2-36-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <ob2> <a0>))
          (true-in-state (at <ob0> <a1>))
          (some-candidate-goals ((at <ob2> <po1>)))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c1> city)
          (type-of-object <ob0> object) (type-of-object <ob2> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-34-e13-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <a1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <ob0> <a0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (inside <ob1> <tr1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (some-candidate-goals ((inside <ob0> <pl0>)))
          (type-of-object <tr1> truck)
          (type-of-object <po1> post-office)
          (type-of-object <ob1> object) (type-of-object <ob0> object)
          (type-of-object <c1> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr1>) (<loc> . <a1>))))

(control-rule regla-logistics-aips98-l4-45-e2-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po0>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (inside <ob2> <tr0>))
          (true-in-state (inside <ob3> <pl0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (some-candidate-goals ((at <ob3> <a0>)))
          (type-of-object <tr0> truck)
          (type-of-object <po0> post-office)
          (type-of-object <ob3> object) (type-of-object <ob2> object)
          (type-of-object <c0> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr0>) (<loc> . <po0>))))
```

```
(control-rule regla-logistics-aips98-l4-26-e3-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (inside <ob1> <tr1>))
          (true-in-state (inside <ob0> <tr0>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr1> <po1>))
          (some-candidate-goals ((inside <ob1> <tr0>)))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c1> city)
          (type-of-object <ob0> object) (type-of-object <ob1> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-26-e13-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <a1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (inside <ob0> <tr0>))
          (true-in-state (inside <ob1> <tr1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <tr1> <po1>))
          (some-candidate-goals ((inside <ob0> <pl0>)))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c1> city)
          (type-of-object <ob0> object) (type-of-object <ob1> object)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr1>) (<loc> . <a1>))))

(control-rule regla-logistics-aips98-l4-8-e8-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <a0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (inside <ob3> <pl0>))
          (true-in-state (inside <ob1> <tr0>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <po0>))
          (some-candidate-goals ((at <pl0> <a0>) (inside <ob3> <tr1>)))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <ob1> object) (type-of-object <ob3> object)
          (type-of-object <po0> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <a0>))))

(control-rule regla-logistics-aips98-l1-10-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po0>))
          (true-in-state (at <ob1> <po1>))
          (true-in-state (at <tr1> <a1>))
```

```
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob1> object) (type-of-object <c1> city)
          (type-of-object <c0> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l1-20-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (inside <ob0> <tr1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <po1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob0> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l2-28-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <ob0> <po1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob0> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l2-4-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po0>))
          (true-in-state (at <ob0> <po1>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <pl0> <a1>))
```

```
          (true-in-state (at <tr0> <a0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob0> object) (type-of-object <c1> city)
          (type-of-object <c0> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l3-15-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob3> <po0>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (inside <ob3> <tr1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob3> object) (type-of-object <c1> city)
          (type-of-object <c0> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob3>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l3-41-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob0> <po1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <ob0> <po0>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob0> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob0>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-15-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a0> <c0>))
```

```
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <ob2> <po0>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob2> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-24-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <ob1> <po0>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob1> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-31-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (inside <ob2> <tr0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob2> object) (type-of-object <c1> city)
          (type-of-object <c0> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr1>) (<loc> . <po1>))))

(control-rule regla-logistics-aips98-l4-34-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po0>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (inside <ob1> <tr1>))
```

```
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (type-of-object <tr1> truck) (type-of-object <tr0> truck)
          (type-of-object <po1> post-office)
          (type-of-object <po0> post-office)
          (type-of-object <ob1> object) (type-of-object <c1> city)
          (type-of-object <c0> city) (type-of-object <a1> airport)
          (type-of-object <a0> airport)
          (type-of-object <pl0> airplane)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-4-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob2> <po0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (at <tr0> <a0>))
          (true-in-state (at <pl0> <a1>))
          (true-in-state (at <tr1> <po1>))
          (true-in-state (at <ob2> <po1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob2> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob2>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l4-9-e1-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po0>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl0> <a0>))
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <ob1> <po1>))
          (type-of-object <pl0> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob1> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <po0>))))

(control-rule regla-logistics-aips98-l3-2-e3-binds
 (if (and (current-operator unload-truck)
          (current-goal (at <ob1> <po0>))
          (true-in-state (loc-at <po1> <c1>))
          (true-in-state (loc-at <a1> <c1>))
          (true-in-state (loc-at <a0> <c0>))
          (true-in-state (loc-at <po0> <c0>))
          (true-in-state (inside <ob1> <tr1>))
```

```
          (true-in-state (at <tr1> <a1>))
          (true-in-state (at <tr0> <po0>))
          (true-in-state (at <pl1> <a0>))
          (some-candidate-goals ((at <tr1> <po1>)))
          (type-of-object <pl1> airplane) (type-of-object <a0> airport)
          (type-of-object <a1> airport) (type-of-object <c0> city)
          (type-of-object <c1> city) (type-of-object <ob1> object)
          (type-of-object <po0> post-office)
          (type-of-object <po1> post-office)
          (type-of-object <tr0> truck) (type-of-object <tr1> truck)))
 (then select bindings
  ((<obj> . <ob1>) (<truck> . <tr0>) (<loc> . <po0>))))
```

B.4. Dominio Miconic

```
(control-rule regla-miconic-mixed-f2-r2-e3-binds
 (if (and (current-operator board) (current-goal (boarded <p0>))
          (true-in-state (lift-at <f1>))
          (true-in-state (origin <p0> <f1>))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)))
 (then select bindings ((<f> . <f1>) (<p> . <p0>))))

(control-rule regla-miconic-mixed-f2-r1-e1-binds
 (if (and (current-operator depart) (current-goal (served <p0>))
          (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (destin <p0> <f1>))
          (true-in-state (origin <p0> <f0>))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select bindings ((<f> . <f1>) (<p> . <p0>))))

(control-rule regla-miconic-mixed-f2-r2-e1-binds
 (if (and (current-operator depart) (current-goal (served <p0>))
          (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (destin <p0> <f0>))
          (true-in-state (origin <p0> <f1>))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select bindings ((<f> . <f0>) (<p> . <p0>))))

(control-rule regla-miconic-mixed-f2-r2-e2-binds
 (if (and (current-operator down) (current-goal (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (lift-at <f1>)) (type-of-object <f0> floor)
          (type-of-object <f1> floor)))
 (then select bindings ((<f1> . <f1>) (<f2> . <f0>))))

(control-rule regla-miconic-mixed-f2-r2-e4-binds
 (if (and (current-operator up) (current-goal (lift-at <f1>))
          (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>)) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select bindings ((<f1> . <f0>) (<f2> . <f1>))))

(control-rule regla-miconic-mixed-f2-r2-e3
 (if (and (current-goal (boarded <p0>)) (true-in-state (lift-at <f1>))
          (true-in-state (origin <p0> <f1>))
```

```
          (type-of-object <p0> passenger) (type-of-object <f1> floor)))
 (then select operators board))

(control-rule regla-miconic-mixed-f2-r1-e1
 (if (and (current-goal (served <p0>)) (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (destin <p0> <f1>))
          (true-in-state (origin <p0> <f0>))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select operators depart))

(control-rule regla-miconic-mixed-f2-r2-e1
 (if (and (current-goal (served <p0>)) (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (destin <p0> <f0>))
          (true-in-state (origin <p0> <f1>))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select operators depart))

(control-rule regla-miconic-mixed-f2-r2-e2
 (if (and (current-goal (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (lift-at <f1>)) (type-of-object <f0> floor)
          (type-of-object <f1> floor)))
 (then select operators down))

(control-rule regla-miconic-mixed-f2-r2-e4
 (if (and (current-goal (lift-at <f1>)) (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>)) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select operators up))

(control-rule regla-miconic-mixed-f2-r2-s2
 (if (and (target-goal (lift-at <f1>))
          (true-in-state (origin <p0> <f1>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (lift-at <f0>))
          (some-candidate-goals ((boarded <p0>)))
          (type-of-object <f0> floor) (type-of-object <f1> floor)
          (type-of-object <p0> passenger)))
 (then select goals (lift-at <f1>)))

(control-rule regla-miconic-mixed-f2-r2-s1
 (if (and (target-goal (boarded <p0>)) (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (origin <p0> <f1>))
          (some-candidate-goals ((lift-at <f0>)))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select goals (boarded <p0>)))

(control-rule regla-miconic-mixed-f2-r1-s1
 (if (and (target-goal (boarded <p0>)) (true-in-state (lift-at <f0>))
          (true-in-state (above <f0> <f1>))
          (true-in-state (origin <p0> <f0>))
          (some-candidate-goals ((lift-at <f1>)))
          (type-of-object <p0> passenger) (type-of-object <f1> floor)
          (type-of-object <f0> floor)))
 (then select goals (boarded <p0>)))
```

Apéndice C

Resultados detallados experimentos

En este apéndice se describen los resultados detallados de los experimentos realizados.

C.1. Resultados de GEBL para mejorar TGP

Las tablas muestran el tiempo total de ejecución hasta encontrar la solución en segundos (TE), el tiempo total de equiparación de las reglas en segundos (TM), el número de puntos de decisión (PD), la longitud del plan paralelo de la solución (M) y el número de operadores en la solución (L). Utilizando las reglas de control aprendidas por GEBL(entre paréntesis se muestran el número de reglas aprendidas) y sin utilizar conocimiento de control (TGP). En las filas **TOTALES**, se muestra la suma de los valores de las columnas correspondientes para poder hacer comparaciones entre sistemas. Por eso, estos totales sólo se calculan para los problemos resueltos por los dos sistemas a comporar. En el caso de que las tablas estén divididas en dos (por razones de espacio), los totales se ponen en la última tabla y se consideran también los valores de las tablas anteriores.

En las primeras tablas se utilizaron los problemas definidos para las competiciones internacionales de planificación. Los problemas más sencillos se usaron para el aprendizaje, se representan en negrita en las tablas. En los otros experimentos se utilizaron problemas aleatorios de validación y aprendizaje de diferentes tamaños y complejidad según el dominio:

- Driverlog: 23 problemas de aprendizaje de 2 metas, y 100 de validación entre 1 y 6 metas.
- Logistics: 200 problemas de aprendizaje de 1 y 2 metas, y 100 de validación

entre 1 y 10 metas.

- Zenotravel: 200 problemas de aprendizaje de 1 a 3 metas, y 100 validación de 1 a 13.
- Robocare: 27 problemas de aprendizaje de 1 a 3 metas y 336 validación de 1 a 50 metas.

Problemas	TGP				GEBL (8 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
DLOG-2-2-2	0.01	000042	006	008	0.04	0.00	000042	006	008
DLOG-2-2-3	0.24	011918	009	019	0.25	0.00	011918	009	019
DLOG-2-2-4	0.18	001548	007	012	0.20	0.00	001548	007	012
DLOG-3-2-5	23.77	408766	045	053	23.43	0.03	408766	045	053
DLOG-3-3-6	0.77	000274	011	015	0.80	0.00	000274	011	015
DLOG-3-3-7	3.68	062600	045	054	3.45	0.01	062600	045	054
DLOG-2-3-6-1	5.75	102919	027	038	5.82	0.03	102919	027	038
DLOG-2-3-6-2	8.51	086220	008	020	7.77	0.05	086220	008	020
DLOG-2-3-6-3	8.59	008034	009	021	8.54	0.00	008034	009	021
TOTALES	51.50	682321	167	240	50.30	0.12	682321	167	240

Tabla C.1: Resultados de GEBL aplicados en TGP con los problemas de la IPC en el dominio Driverlog.

Problemas	TGP				GEBL (1 regla)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
ZTRAVEL-1-2-P1	0.02	000004	001	001	0.01	0.00	000004	001	001
ZTRAVEL-2-4-P3	0.30	000077	005	009	0.29	0.00	000077	005	009
ZTRAVEL-2-5-P4	0.32	000315	007	008	0.32	0.00	000314	007	008
ZTRAVEL-1-3-P2	0.16	000057	005	006	0.16	0.00	000055	005	006
ZTRAVEL-2-4-P5	1.22	001156	007	011	1.18	0.02	001156	007	011
ZTRAVEL-2-5-P6	1.05	004340	005	012	1.07	0.01	004340	005	012
ZTRAVEL-3-6-P8	5.75	016490	006	012	5.66	0.05	016490	006	012
ZTRAVEL-3-7-P9	6.76	171169	007	023	7.95	0.87	171169	007	023
TOTALES	15.58	193608	43	82	16.64	0.95	193605	43	82

Tabla C.2: Resultados de GEBL aplicados en TGP con los problemas de la IPC en el dominio Zenotravel.

Problemas	TGP				GEBL(6 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
STRIPS-LOG-X-1	1.61	001059	009	027	-	-	-	-	-
STRIPS-LOG-X-2	15.93	000318	007	032	-	-	-	-	-
STRIPS-LOG-X-5	55.79	1677001	012	024	-	-	-	-	-
STRIPS-LOG-X-11	34.09	000152	009	033	-	-	-	-	-

Tabla C.3: Resultados de GEBL aplicados en TGP con los problemas de la IPC en el dominio Logistics.

Problemas	TGP				GEBL (6 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
STRIPS-SAT-X-1-P1	0.04	000086	008	009	0.08	0.03	000083	008	009
STRIPS-SAT-X-1-P2	0.64	028189	012	013	2.08	1.42	038226	012	013
STRIPS-SAT-X-1-P3	0.67	031454	006	013	2.24	1.63	030763	006	013
TOTALES	1.35	59729	26	35	4.40	3.08	69072	26	35
STRIPS-SAT-X-1-P6	9.35	340499	008	023	-	-	-	-	-

Tabla C.4: Resultados de GEBL aplicados en TGP con los problemas de la IPC en el dominio Satellite.

Problemas	TGP				GEBL(8 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
MIXED-F2-P1-R0	0.01	000010	004	004	0.01	0.00	000010	004	004
MIXED-F2-P1-R1	0.00	000008	003	003	0.00	0.00	000007	003	003
MIXED-F2-P1-R2	0.00	000010	004	004	0.00	0.00	000010	004	004
MIXED-F2-P1-R3	0.01	000010	004	004	0.01	0.00	000010	004	004
MIXED-F2-P1-R4	0.00	000010	004	004	0.00	0.00	000010	004	004
MIXED-F4-P2-R0	0.01	000025	006	007	0.01	0.00	000025	006	007
MIXED-F4-P2-R1	0.01	000046	006	007	0.01	0.00	000039	006	007
MIXED-F4-P2-R2	0.01	000088	006	007	0.01	0.00	000081	006	007
MIXED-F4-P2-R3	0.02	000085	006	007	0.00	0.00	000084	006	007
MIXED-F4-P2-R4	0.01	000042	006	007	0.01	0.00	000042	006	007
MIXED-F6-P3-R0	0.03	000659	008	010	0.04	0.00	000658	008	010
MIXED-F6-P3-R1	0.04	002264	010	011	0.07	0.04	002264	010	011
MIXED-F6-P3-R2	0.05	002636	008	010	0.07	0.02	002636	008	010
MIXED-F6-P3-R3	0.05	001252	009	010	0.07	0.02	001251	009	010
MIXED-F6-P3-R4	0.04	001583	008	010	0.05	0.02	001583	008	010
MIXED-F8-P4-R0	0.26	014695	012	014	0.35	0.10	014695	012	014
MIXED-F8-P4-R1	0.21	011187	011	013	0.32	0.15	011185	011	013
MIXED-F8-P4-R2	0.55	032745	014	015	0.82	0.28	032745	014	015
MIXED-F8-P4-R3	0.56	033123	014	015	0.75	0.24	033123	014	015
MIXED-F8-P4-R4	0.62	037435	014	015	0.85	0.27	037435	014	015
MIXED-F10-P5-R0	2.38	131953	014	017	3.63	1.40	131945	014	017
MIXED-F10-P5-R1	4.09	215629	015	017	5.96	2.12	215628	015	017
MIXED-F10-P5-R2	0.44	016799	010	015	0.68	0.31	016797	010	015
MIXED-F10-P5-R3	2.49	137237	014	017	4.26	1.95	137224	014	017
MIXED-F10-P5-R4	3.05	167577	016	018	4.27	1.39	167577	016	018
MIXED-F12-P6-R0	9.33	453436	014	019	16.95	8.14	453630	014	019
MIXED-F12-P6-R1	7.33	348964	015	019	11.22	4.30	348959	015	019
MIXED-F12-P6-R2	21.43	941063	016	020	30.93	11.71	941051	016	020
MIXED-F12-P6-R3	9.04	424268	016	020	16.74	8.40	424268	016	020
MIXED-F12-P6-R4	16.61	737788	018	021	27.24	12.72	737788	018	021
MIXED-F14-P7-R2	28.94	1104449	017	022	42.23	17.15	1104430	017	022
TOTALES	107.62	4817076	322	366	167.56	70.73	4817190	322	366
MIXED-F14-P7-R0	55.49	2135115	018	023	-	-	-	-	-
MIXED-F14-P7-R3	45.18	1721581	017	022	-	-	-	-	-

Tabla C.5: Resultados de GEBL aplicados en TGP con los problemas de la IPC en el dominio Miconic.

Problemas	TGP				GEBL (8 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
P001-01	0.00	000000	001	000	0.00	0.00	000000	001	000
P002-01	0.01	000000	001	000	0.00	0.00	000000	001	000
P003-02	0.03	000025	005	006	0.03	0.00	000025	005	006
P004-02	0.04	000034	007	007	0.04	0.00	000034	007	007
P005-02	0.04	000027	005	007	0.05	0.00	000027	005	007
P006-02	0.03	000025	005	006	0.03	0.00	000025	005	006
P007-02	0.08	000053	011	011	0.10	0.00	000053	011	011
P008-03	0.15	000034	006	006	0.16	0.00	000034	006	006
P009-03	0.04	000047	009	010	0.04	0.00	000047	009	010
P010-03	0.03	000068	012	012	0.03	0.00	000068	012	012
P011-03	0.04	000049	007	009	0.04	0.00	000049	007	009
P012-03	0.03	000270	012	013	0.03	0.00	000270	012	013
P013-03	0.04	000023	004	005	0.04	0.00	000023	004	005
P014-03	0.05	000127	009	009	0.05	0.00	000127	009	009
P015-03	0.03	000049	007	009	0.04	0.00	000049	007	009
P016-03	0.03	000068	012	012	0.03	0.00	000068	012	012
P017-03	0.04	000047	009	010	0.04	0.00	000047	009	010
P018-03	0.14	000033	006	008	0.16	0.00	000033	006	008
P019-03	0.07	000041	003	007	0.06	0.00	000041	003	007
P020-03	0.13	000184	010	011	0.12	0.01	000184	010	011
P021-03	0.53	000154	008	009	0.56	0.00	000154	008	009
P022-03	0.57	000043	008	008	0.59	0.00	000043	008	008
P023-03	0.86	000054	007	009	0.89	0.00	000054	007	009
P024-03	0.68	000030	005	007	0.70	0.00	000030	005	007
P025-04	0.23	008682	007	010	0.23	0.00	008682	007	010
P026-04	0.08	000025	004	007	0.08	0.00	000025	004	007
P027-04	0.04	000049	005	006	0.03	0.00	000049	005	006
P028-04	0.05	000095	008	008	0.04	0.00	000095	008	008
P029-04	0.04	000028	003	003	0.05	0.00	000028	003	003
P030-04	0.04	000105	010	012	0.05	0.00	000105	010	012
P031-04	0.04	000041	006	009	0.04	0.00	000041	006	009
P032-04	0.03	000107	007	008	0.03	0.00	000107	007	008
P033-04	0.04	000064	007	008	0.03	0.00	000064	007	008
P034-04	0.04	000160	009	011	0.04	0.00	000160	009	011
P035-04	0.03	000199	013	013	0.03	0.00	000199	013	013
P036-04	0.04	000105	010	012	0.04	0.01	000105	010	012
P037-04	0.05	000105	010	012	0.04	0.00	000105	010	012
P038-04	0.04	000105	010	012	0.05	0.00	000105	010	012
P039-04	0.05	000028	003	003	0.05	0.00	000028	003	003
P040-04	0.04	000095	008	008	0.04	0.00	000095	008	008
P041-04	0.03	000049	005	006	0.04	0.00	000049	005	006
P042-04	0.13	000042	005	010	0.14	0.00	000042	005	010
P043-04	0.20	000910	009	010	0.20	0.00	000910	009	010
P044-04	0.03	000020	004	004	0.03	0.00	000020	004	004
P045-04	0.02	000020	004	004	0.02	0.00	000020	004	004
P046-04	0.08	000076	007	008	0.08	0.00	000076	007	008
P047-04	0.25	003360	041	045	0.26	0.00	003360	041	045
P048-04	0.17	000040	005	011	0.17	0.00	000040	005	011
P049-04	0.36	000064	007	010	0.36	0.00	000064	007	010
P050-04	0.31	000041	005	011	0.27	0.00	000041	005	011

Tabla C.6: Resultados de GEBL aplicados en TGP con problemas aleatorios en el dominio Driverlog I.

Problemas	TGP				GEBL (8 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
P051-04	0.36	000024	004	008	0.38	0.00	000024	004	008
P052-04	0.37	000024	004	008	0.42	0.01	000024	004	008
P053-04	0.58	000205	015	022	0.59	0.00	000205	015	022
P054-04	0.68	000066	007	015	0.68	0.00	000066	007	015
P055-04	0.55	000063	008	012	0.51	0.00	000063	008	012
P056-04	0.38	000045	006	013	0.49	0.00	000045	006	013
P057-04	0.96	000749	008	013	0.96	0.00	000749	008	013
P058-04	0.67	000728	010	011	0.69	0.00	000728	010	011
P059-04	1.23	000051	006	013	1.24	0.00	000051	006	013
P060-04	0.26	000023	003	008	0.28	0.00	000023	003	008
P061-04	1.79	000051	006	013	1.84	0.00	000051	006	013
P062-04	1.40	000092	008	015	1.32	0.00	000092	008	015
P063-04	1.32	000051	006	013	1.41	0.00	000051	006	013
P064-04	0.98	000087	009	012	0.99	0.00	000087	009	012
P065-04	0.34	000034	004	012	0.37	0.00	000034	004	012
P066-05	0.34	009774	012	017	0.34	0.01	009774	012	017
P067-05	0.16	000087	008	012	0.16	0.00	000087	008	012
P068-05	0.17	000087	008	012	0.15	0.00	000087	008	012
P069-05	0.20	000212	006	007	0.20	0.00	000212	006	007
P070-05	0.15	000056	007	013	0.15	0.00	000056	007	013
P071-05	0.25	000050	006	010	0.27	0.00	000050	006	010
P072-05	0.14	003394	007	011	0.14	0.00	003394	007	011
P073-05	1.25	077245	016	021	1.24	0.01	077245	016	021
P074-05	0.41	000086	009	015	0.42	0.00	000086	009	015
P075-05	0.41	000086	009	015	0.40	0.00	000086	009	015
P076-05	0.65	005891	007	012	0.66	0.00	005891	007	012
P077-05	0.37	000085	009	015	0.38	0.00	000085	009	015
P078-05	0.42	000073	008	014	0.42	0.01	000073	008	014
P079-05	0.70	000076	006	016	0.69	0.00	000076	006	016
P080-05	0.88	003314	007	017	0.86	0.00	003314	007	017
P081-05	1.04	003314	007	017	0.96	0.00	003314	007	017
P082-05	2.10	048540	016	024	2.22	0.01	048540	016	024
P083-05	1.68	045208	007	017	1.72	0.00	045208	007	017
P084-05	0.71	000997	009	020	0.66	0.00	000997	009	020
P085-05	0.85	000069	007	014	0.88	0.00	000069	007	014
P086-05	0.49	000058	006	014	0.48	0.00	000058	006	014
P087-05	0.45	000058	006	014	0.46	0.00	000058	006	014
P088-05	0.67	000251	016	022	0.69	0.00	000251	016	022
P089-05	0.74	000098	009	018	0.75	0.00	000098	009	018
P090-05	0.65	000178	010	016	0.62	0.00	000178	010	016
P091-05	0.43	000298	009	016	0.41	0.00	000298	009	016
P092-05	0.95	000082	008	017	0.98	0.00	000082	008	017
P093-05	1.00	000082	008	017	0.93	0.00	000082	008	017
P094-05	1.14	000470	009	018	1.19	0.00	000470	009	018
P095-05	0.71	000131	008	018	0.69	0.00	000131	008	018
P096-05	0.45	000026	003	008	0.53	0.00	000026	003	008
P097-05	0.28	000026	003	008	0.27	0.00	000026	003	008
P098-05	0.05	000017	002	006	0.04	0.00	000017	002	006
P099-06	1.57	004475	009	013	1.58	0.00	004475	009	013
P100-06	1.31	001385	007	019	1.26	0.00	001385	007	019
TOTALES	41.79	224672	769	1163	42.24	0.07	224672	769	1163

Tabla C.7: Resultados de GEBL aplicados en TGP con problemas aleatorios en el dominio Driverlog II.

Problemas	TGP				GEBL(1 regla)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
P001-01	0.00	000000	001	000	0.00	0.00	000000	001	000
P002-02	0.02	000013	003	004	0.01	0.00	000013	003	004
P003-02	0.00	000000	001	000	0.00	0.00	000000	001	000
P006-02	0.00	000000	001	000	0.01	0.00	000000	001	000
P007-02	0.02	000013	003	004	0.01	0.00	000013	003	004
P008-02	0.01	000000	001	000	0.01	0.00	000000	001	000
P009-02	0.00	000000	001	000	0.00	0.00	000000	001	000
P010-02	0.01	000000	001	000	0.00	0.00	000000	001	000
P011-02	0.02	000013	003	004	0.02	0.00	000013	003	004
P013-02	0.00	000000	001	000	0.00	0.00	000000	001	000
P014-02	0.03	000013	003	004	0.02	0.00	000013	003	004
P015-02	0.00	000000	001	000	0.00	0.00	000000	001	000
P016-02	0.00	000000	001	000	0.01	0.00	000000	001	000
P017-02	0.02	000013	003	004	0.01	0.00	000013	003	004
P022-04	0.03	000019	003	004	0.01	0.00	000019	003	004
P024-04	0.01	000000	001	000	0.00	0.00	000000	001	000
P025-04	0.03	000019	003	004	0.02	0.00	000019	003	004
P027-05	0.03	000022	003	004	0.02	0.00	000022	003	004
P031-05	0.03	000024	003	006	0.02	0.00	000024	003	006
P032-05	0.02	000022	003	004	0.01	0.00	000022	003	004
P034-05	0.02	000024	003	006	0.02	0.00	000024	003	006
TOTALES	0.30	195.00	43.00	48	0.20	0.00	195.00	43.00	48

Tabla C.8: Resultados de GEBL aplicados en TGP con problemas aleatorios en el dominio Zenotravel.

Problemas	TGP				GEBL(6 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
T-10-5-1	1.08	015240	013	028	2.60	1.47	015239	013	028
T-10-5-2	0.84	007013	007	012	1.60	0.67	007013	007	012
T-10-5-6	0.50	000097	006	019	87.16	86.66	000096	006	019
T-10-5-9	0.93	000137	008	022	58.78	57.99	000137	008	022
TOTALES	3.35	22487	34	81	150.14	146.79	22485	34	81
T-10-5-0	1.06	000223	006	021	-	-	-	-	-
T-10-5-3	1.25	005183	008	029	-	-	-	-	-
T-10-5-4	0.84	000071	005	012	-	-	-	-	-
T-10-5-5	2.71	070611	008	022	-	-	-	-	-
T-10-5-7	1.38	000230	010	031	-	-	-	-	-
T-10-5-8	1.17	000342	008	026	-	-	-	-	-
T-1-10-0	4.33	000005	002	002	-	-	-	-	-
T-1-10-1	0.47	000002	001	001	-	-	-	-	-
T-1-10-2	0.00	000000	001	000	-	-	-	-	-
T-1-10-5	1.08	000002	001	001	-	-	-	-	-
T-1-10-6	0.83	000006	002	003	-	-	-	-	-
T-1-10-8	53.83	000019	005	005	-	-	-	-	-
T-1-10-9	5.99	000010	003	004	-	-	-	-	-
T-1-5-0	0.00	000000	001	000	-	-	-	-	-
T-1-5-1	0.57	000021	005	006	-	-	-	-	-
T-1-5-2	0.10	000005	002	002	-	-	-	-	-
T-1-5-3	0.28	000020	005	006	-	-	-	-	-
T-1-5-4	0.82	000010	003	003	-	-	-	-	-
T-1-5-5	0.24	000013	004	004	-	-	-	-	-
T-1-5-6	0.08	000005	002	002	-	-	-	-	-
T-1-5-7	0.06	000006	002	003	-	-	-	-	-
T-1-5-8	0.02	000002	001	001	-	-	-	-	-
T-1-5-9	0.22	000016	004	005	-	-	-	-	-
T-2-10-0	25.98	000016	003	005	-	-	-	-	-
T-2-10-2	8.45	000012	002	006	-	-	-	-	-
T-2-10-5	0.48	000003	001	001	-	-	-	-	-
T-2-10-6	49.99	000014	003	005	-	-	-	-	-
T-2-10-7	0.51	000003	001	001	-	-	-	-	-
T-2-10-8	0.49	000004	001	002	-	-	-	-	-
T-2-5-0-10	0.27	000037	007	010	-	-	-	-	-
T-2-5-0-11	0.12	000022	005	005	-	-	-	-	-
T-2-5-0-12	0.12	000036	006	009	-	-	-	-	-
T-2-5-0-13	0.44	000360	009	014	-	-	-	-	-
T-2-5-0-14	0.27	000052	008	013	-	-	-	-	-
T-2-5-0-15	0.20	000065	006	008	-	-	-	-	-
T-2-5-0-16	0.35	000371	009	015	-	-	-	-	-
T-2-5-0-17	0.18	000030	005	009	-	-	-	-	-
T-2-5-0-18	0.22	000035	005	008	-	-	-	-	-
T-2-5-0-19	0.29	000056	007	009	-	-	-	-	-
T-2-5-0	0.08	000008	002	003	-	-	-	-	-
T-2-5-1	0.28	000013	003	004	-	-	-	-	-
T-2-5-2	0.10	000010	002	004	-	-	-	-	-
T-2-5-3	0.27	000013	003	005	-	-	-	-	-
T-2-5-4	0.27	000015	004	004	-	-	-	-	-
T-2-5-5	0.03	000003	001	001	-	-	-	-	-
T-2-5-6	0.12	000012	002	006	-	-	-	-	-
T-2-5-7	0.03	000003	001	001	-	-	-	-	-
T-2-5-8	1.04	000026	005	006	-	-	-	-	-
T-2-5-9	3.29	000048	008	011	-	-	-	-	-

Tabla C.9: Resultados de GEBL aplicados en TGP con problemas aleatorios en el dominio Logistics I.

Problemas	TGP				GEBL(6 reglas)				
	TE(s)	PD	M	L	TE(s)	TM(s)	PD	M	L
T-3-7-0-10	0.89	001847	010	020	-	-	-	-	-
T-3-7-0-11	1.07	010539	010	017	-	-	-	-	-
T-3-7-0-12	0.62	000232	008	017	-	-	-	-	-
T-3-7-0-13	0.75	000064	010	012	-	-	-	-	-
T-3-7-0-14	0.65	001951	009	017	-	-	-	-	-
T-3-7-0-15	0.61	000301	009	015	-	-	-	-	-
T-3-7-0-16	0.80	000052	007	013	-	-	-	-	-
T-3-7-0-17	0.77	000053	008	013	-	-	-	-	-
T-3-7-0-18	0.84	000709	009	015	-	-	-	-	-
T-3-7-0-19	0.85	000044	007	010	-	-	-	-	-
T-3-7-0-1	0.73	001755	010	017	-	-	-	-	-
T-3-7-0-2	0.87	000071	008	011	-	-	-	-	-
T-3-7-0-3	0.85	000102	007	013	-	-	-	-	-
T-3-7-0-4	0.73	001582	009	019	-	-	-	-	-
T-3-7-0-5	0.40	000048	007	012	-	-	-	-	-
T-3-7-0-6	0.20	000029	005	008	-	-	-	-	-
T-3-7-0-7	0.79	000208	009	019	-	-	-	-	-
T-3-7-0-8	0.20	000056	005	008	-	-	-	-	-
T-3-7-0-9	0.37	000112	006	012	-	-	-	-	-
T-5-5-0	0.40	000064	006	014	-	-	-	-	-
T-5-5-10	2.58	069755	012	032	-	-	-	-	-
T-5-5-11	0.60	007039	010	027	-	-	-	-	-
T-5-5-12	2.83	060830	011	029	-	-	-	-	-
T-5-5-13	0.19	000126	005	010	-	-	-	-	-
T-5-5-14	6.58	203722	011	031	-	-	-	-	-
T-5-5-15	0.35	000077	007	018	-	-	-	-	-
T-5-5-16	2.28	052566	008	014	-	-	-	-	-
T-5-5-17	0.35	000524	008	014	-	-	-	-	-
T-5-5-18	0.46	000599	009	022	-	-	-	-	-
T-5-5-19	0.35	000088	010	019	-	-	-	-	-
T-5-5-1	1.09	000031	004	006	-	-	-	-	-
T-5-5-20	0.37	000127	009	028	-	-	-	-	-
T-5-5-2	1.47	000076	006	015	-	-	-	-	-
T-5-5-3	0.71	000479	006	021	-	-	-	-	-
T-5-5-4	0.63	000203	007	016	-	-	-	-	-
T-5-5-5	1.29	000047	005	010	-	-	-	-	-
T-5-5-6	0.03	000006	001	001	-	-	-	-	-
T-5-5-7	1.58	000114	008	025	-	-	-	-	-
T-5-5-8	1.57	000060	006	013	-	-	-	-	-
T-5-5-9	2.34	000035	004	010	-	-	-	-	-

Tabla C.10: Resultados de GEBL aplicados en TGP con problemas aleatorios en el dominio Logistics II.

Problemas	TGP			GEBL (12 reglas)		
	TE(s)	PD	M	TE(s)	PD	M
P15-04-001	0.53	014423	004	2.84	000903	004
P15-05-001	53.37	2252174	004	9.96	001924	004
P15-05-002	0.18	001320	003	2.48	000127	003
P15-05-003	0.34	007638	003	5.35	002507	003
P15-06-001	0.19	000163	003	3.08	000198	003
P15-06-002	0.16	000155	003	3.24	000154	003
P15-06-003	0.27	000387	003	3.12	000526	003
P15-07-001	0.34	000132	003	3.20	000310	003
P15-07-002	0.20	000175	003	4.72	000209	003
P15-07-003	0.26	000102	003	2.64	000100	003
P15-08-001	0.38	000128	003	9.41	000419	003
P15-08-002	0.15	000240	003	1.16	000100	003
P15-08-003	0.68	000157	003	8.04	000094	003
P15-09-001	0.11	000078	002	1.16	000071	002
P15-09-002	0.27	000183	003	6.09	000441	003
P15-09-003	0.16	000080	002	2.83	000072	002
P15-10-001	0.30	000103	003	2.95	000098	003
P15-10-002	0.13	000094	003	1.17	000094	003
P15-10-003	0.16	000086	002	3.35	000077	002
P15-20-001	0.21	000076	002	7.01	000099	002
P15-20-002	0.50	000087	002	6.82	001506	002
P15-20-003	0.76	000082	002	8.93	000067	002
P20-05-001	0.15	000101	002	6.72	000092	002
P20-05-002	1.25	033469	004	3.78	000255	004
P20-05-003	0.08	000270	002	2.06	000256	002
P20-06-001	0.23	000453	003	12.95	000213	003
P20-06-002	0.29	002111	003	14.90	001944	003
P20-06-003	0.34	000139	002	8.91	000124	002
P20-07-001	0.54	000996	003	13.14	000382	003
P20-07-002	0.63	000289	004	12.20	000167	004
P20-07-003	0.44	000280	003	14.75	000161	003
P20-08-001	0.77	000675	003	17.46	000212	003
P20-08-002	0.60	000109	002	13.43	000097	002
P20-08-003	0.24	000538	003	6.08	000505	003
P20-09-002	0.48	000143	003	6.54	000151	003
P20-09-003	0.19	000105	002	5.08	000094	002
P20-10-001	0.71	000115	002	20.39	000095	002
P20-10-002	0.19	000113	002	5.16	000100	002
P20-10-003	0.15	000083	002	4.30	000079	002
P20-20-001	0.45	000100	002	12.05	000092	002
P20-20-002	0.62	000103	002	12.91	000092	002
P20-20-003	0.41	000099	002	7.47	000088	002
P30-05-002	0.47	001073	004	13.71	004280	004
P30-06-001	0.49	001918	003	59.83	001076	003
P30-06-003	21.33	590162	003	32.67	000187	003
P30-07-001	0.19	000149	002	12.22	000134	002
P30-07-002	0.83	000188	003	29.17	000185	003
P30-07-003	0.40	000398	003	54.75	000813	003
P30-08-002	0.14	000140	002	9.78	000124	002
P30-08-003	1.93	013538	003	60.83	030155	003
P30-09-002	0.17	000127	002	11.65	000115	002
P30-09-003	0.27	000166	002	25.22	000145	002
P30-10-001	0.46	000173	003	22.79	000173	003
P30-10-002	1.38	001376	002	81.31	001074	002
P30-10-003	0.46	000152	002	27.56	000135	002
P30-20-001	2.18	000134	002	55.03	000121	002
P30-20-002	3.44	000155	002	91.83	000139	002
P30-20-003	0.97	000150	002	44.85	000137	002
P40-09-001	1.47	000259	003	78.48	000251	003
P40-10-001	0.49	000192	002	81.82	000173	002
P40-10-002	0.37	000183	002	34.66	000164	002
P40-10-003	1.29	000237	003	76.83	000233	003
P40-20-001	1.22	000194	002	96.19	000175	002
P40-20-003	1.66	000203	002	102.40	000180	002
P50-07-002	0.37	000238	002	72.09	000212	002
TOTAL	138.20	3745275	693	1771.76	558265	693

Tabla C.11: Resultados de GEBL aplicados en TGP con problemas aleatorios en el dominio Robocare (últimos problemas resueltos de 287).

C.2. Resultados de GEBL y HAMLET para mejorar PRODIGY

En estas tablas se muestran los resultados detallados de los experimentos realizados en el estudio de la transferencia de conocimiento de control. Muestran el tiempo (en segundos) y la calidad de las soluciones (en número de operadores en el plan) de cada sistema: TGP, PRODIGY , PRODIGY utilizando las reglas aprendidas por GEBL con los problemas de la competición y por último PRODIGY con las reglas aprendidas por GEBL con los problemas aleatorios. En las primeras tablas se utilizaron los problemas definidos para las competiciones internacionales de planificación igual que en los experimentos de la sección anterior. En negrita se representan los problemas utilizados para el aprendizaje. En algunas tablas se muestran resultados TOTALES en los que se suman todos los valores de las columnas correspondientes con el fin de comparar los sistemas. Estas sumas sólo se pueden realizar para los problemas que resuelvan todos los sistemas comparados. El modo para hacer la regresión de metas fue el de omisión utilizado por GEBL para la transferencia a PRODIGY(suponiendo que las metas que persisten en el punto de decisión se habrán satisfecho) en todos los dominios menos para el Logistics en que la regresión de metas se hizo asumiendo que el meta-estado es el estado inicial del problema.

Problemas	TGP		PRODIGY		PRODIGY-GEBL (46 r. IPC)		PRODIGY-GEBL (73 r. aleat.)	
	Q	T	Q	T	Q	T	Q	T
DLOG-2-2-2	8	0.01	-	-	8	0.38	8	0.17
DLOG-2-2-4	12	0.18	-	-	-	-	19	0.71
DLOG-2-2-3	19	0.24	-	-	-	-	-	-
DLOG-3-2-5	53	23.77	-	-	-	-	-	-
DLOG-3-3-6	15	0.77	-	-	-	-	-	-
DLOG-3-3-7	54	3.68	-	-	-	-	-	-
DLOG-2-3-6	38	5.75	-	-	-	-	-	-
DLOG-2-3-6	20	8.51	-	-	-	-	-	-
DLOG-2-3-6	21	8.59	-	-	-	-	-	-

Tabla C.12: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Driverlog.

Problemas	TGP		PRODIGY		PRODIGY-GEBL (14 r. IPC)		PRODIGY-GEBL (14 r. aleat.)	
	Q	T	Q	T	Q	T	Q	T
ZTRAVEL-1-2-P1	1	0.02	1	0.02	1	0.02	2	0.01
ZTRAVEL-1-3-P2	6	0.16	-	-	-	-	8	0.03
ZTRAVEL-2-4-P3	9	0.30	6	0.01	6	0.03	8	0.02
ZTRAVEL-2-5-P4	8	0.32	-	-	-	-	14	0.04
ZTRAVEL-2-4-P5	11	1.22	-	-	-	-	25	0.08
ZTRAVEL-2-5-P6	12	1.05	-	-	-	-	34	0.12
ZTRAVEL-3-6-P8	12	5.75	-	-	-	-	27	0.12
ZTRAVEL-3-7-P9	23	6.76	-	-	-	-	32	0.12
TOTALES	82	15.58					150	0.54
ZTRAVEL-2-6-P7	-	-	-	-	-	-	24	0.07
ZTRAVEL-3-8-P10	-	-	-	-	-	-	64	0.30
ZTRAVEL-3-7-P11	-	-	-	-	-	-	34	0.14
ZTRAVEL-3-8-P12	-	-	-	-	-	-	51	0.24
ZTRAVEL-3-10-P13	-	-	-	-	-	-	64	0.33
ZTRAVEL-5-10-P14	-	-	-	-	-	-	88	0.67
ZTRAVEL-5-15-P15	-	-	-	-	-	-	86	0.69
ZTRAVEL-5-15-P16	-	-	-	-	-	-	95	1.16
ZTRAVEL-5-20-P17	-	-	-	-	-	-	172	2.09
ZTRAVEL-5-20-P18	-	-	-	-	-	-	137	1.68
ZTRAVEL-5-25-P19	-	-	-	-	-	-	234	3.92
ZTRAVEL-5-25-P20	-	-	-	-	-	-	238	4.55

Tabla C.13: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Zenotravel.

Problemas	TGP		PRODIGY		PRODIGY-GEBL (1 reglas IPC)	
	Q	T	Q	T	Q	T
STRIPS-SAT-X-1-P1	9	0.04	9	0.02	9	0.02
STRIPS-SAT-X-1-P2	13	0.64	17	0.02	17	0.02
STRIPS-SAT-X-1-P3	13	0.67	14	0.02	14	0.03
STRIPS-SAT-X-1-P4	-	-	23	0.03	23	0.04
STRIPS-SAT-X-1-P5	-	-	20	0.02	20	0.03
STRIPS-SAT-X-1-P6	23	9.35	20	0.03	20	0.14
STRIPS-SAT-X-1-P7	-	-	30	0.04	30	0.06
STRIPS-SAT-X-1-P8	-	-	26	0.04	26	0.05
STRIPS-SAT-X-1-P9	-	-	36	0.06	36	0.09
STRIPS-SAT-X-1-P10	-	-	36	0.06	36	0.09
STRIPS-SAT-X-1-P11	-	-	35	0.06	35	0.09
STRIPS-SAT-X-1-P12	-	-	55	0.11	55	0.14
STRIPS-SAT-X-1-P13	-	-	74	0.20	74	0.23
STRIPS-SAT-X-1-P14	-	-	76	0.17	94	0.33
STRIPS-SAT-X-1-P15	-	-	67	0.17	67	0.21
STRIPS-SAT-X-1-P16	-	-	60	0.15	60	0.19
STRIPS-SAT-X-1-P17	-	-	55	0.23	55	0.26
STRIPS-SAT-X-1-P18	-	-	61	0.11	61	0.15
STRIPS-SAT-X-1-P19	-	-	93	0.30	91	0.33
STRIPS-SAT-X-1-P20	-	-	203	1.21	201	1.19
TOTALES			1010	3.05	1024	3.69

Tabla C.14: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Satellite.

Problemas	TGP		PRODIGY		PRODIGY-GEBL (6 reglas IPC)	
	Q	T	Q	T	Q	T
BLOCKS-8-1	-	-	78	0.24	-	-
BLOCKS-6-2	-	-	38	0.08	-	-
BLOCKS-6-0	-	-	40	17.89	-	-
BLOCKS-4-2	-	-	14	2.29	16	12.01
BLOCKS-4-1	-	-	16	0.02	-	-
BLOCKS-4-0	6	0.08	10	0.02	10	0.02
BLOCKS-12-1	-	-	372	8.77	-	-
BLOCKS-10-2	-	-	180	1.94	-	-
BLOCKS-10-0	-	-	180	2.00	-	-

Tabla C.15: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Blocks.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (13 reglas)	
	Q	T	Q	T	Q	T	Q	T
mixed-f14-r3	22	45.18	-	-	28	0.13	34	0.08
mixed-f14-r2	22	28.94	-	-	26	0.10	32	0.09
mixed-f14-r0	23	55.49	-	-	28	0.13	35	0.09
mixed-f12-r4	21	16.61	-	-	24	0.08	31	0.07
mixed-f12-r3	20	9.04	-	-	25	0.08	29	0.07
mixed-f12-r2	20	21.43	-	-	23	0.08	28	0.07
mixed-f12-r1	19	7.33	-	-	24	0.09	30	0.07
mixed-f12-r0	19	9.33	-	-	23	0.07	29	0.06
mixed-f10-r4	18	3.05	-	-	20	0.05	25	0.05
mixed-f10-r3	17	2.49	-	-	20	0.05	24	0.05
mixed-f10-r1	17	4.09	-	-	19	0.04	22	0.04
mixed-f10-r0	17	2.38	-	-	20	0.05	24	0.05
mixed-f8-r3	15	0.56	-	-	15	0.05	18	0.03
mixed-f8-r2	15	0.55	-	-	15	0.04	17	0.03
mixed-f8-r1	13	0.21	-	-	16	0.04	18	0.03
mixed-f8-r0	14	0.26	-	-	16	0.03	20	0.03
mixed-f6-r3	10	0.05	-	-	12	0.02	14	0.02
mixed-f6-r1	11	0.04	-	-	11	0.01	16	0.02
mixed-f6-r0	10	0.03	-	-	12	0.02	14	0.03
mixed-f10-r2	15	0.44	25	0.02	19	0.04	23	0.05
mixed-f6-r2	10	0.05	12	0.01	12	0.03	12	0.02
mixed-f6-r4	10	0.04	11	7.24	11	0.03	11	0.02
mixed-f8-r4	15	0.62	27	1.55	15	0.05	18	0.03
TOTALES	373	208.21			434	1.31	524	1.10
mixed-f18-r4	-	-	-	-	36	0.30	-	-
mixed-f18-r3	-	-	53	0.31	36	0.28	43	0.24

Tabla C.16: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Miconic I.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (13 reglas)	
	Q	T	Q	T	Q	T	Q	T
mixed-f28-r4	-	-	-	-	55	1.45	69	0.34
mixed-f28-r3	-	-	-	-	54	1.40	69	0.35
mixed-f28-r2	-	-	-	-	55	1.24	67	0.35
mixed-f28-r1	-	-	-	-	56	1.48	69	0.34
mixed-f28-r0	-	-	-	-	55	1.33	70	0.35
mixed-f26-r4	-	-	-	-	53	1.07	63	0.30
mixed-f26-r3	-	-	-	-	52	1.11	63	0.31
mixed-f26-r2	-	-	-	-	52	1.00	65	0.30
mixed-f26-r1	-	-	-	-	52	1.06	64	0.31
mixed-f26-r0	-	-	-	-	52	1.18	65	0.29
mixed-f24-r4	-	-	-	-	48	0.69	58	0.24
mixed-f24-r3	-	-	-	-	47	0.66	59	0.26
mixed-f24-r2	-	-	-	-	48	0.70	59	0.25
mixed-f24-r1	-	-	-	-	49	0.86	60	0.25
mixed-f24-r0	-	-	-	-	48	0.93	59	0.23
mixed-f22-r4	-	-	-	-	43	0.60	56	0.22
mixed-f22-r3	-	-	-	-	44	0.55	55	0.25
mixed-f22-r2	-	-	-	-	45	0.59	54	0.20
mixed-f22-r1	-	-	-	-	42	0.52	53	0.21
mixed-f22-r0	-	-	-	-	44	0.45	55	0.22
mixed-f20-r4	-	-	-	-	41	0.40	49	0.17
mixed-f20-r3	-	-	-	-	39	0.38	49	0.18
mixed-f20-r2	-	-	-	-	40	0.36	48	0.16
mixed-f20-r1	-	-	-	-	40	0.36	48	0.17
mixed-f20-r0	-	-	-	-	40	0.45	48	0.17

Tabla C.17: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Miconic II.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (13 reglas)	
	Q	T	Q	T	Q	T	Q	T
mixed-f46-r4	-	-	-	-	92	9.38	115	1.23
mixed-f46-r3	-	-	-	-	89	9.27	111	1.23
mixed-f46-r2	-	-	-	-	91	10.98	115	1.23
mixed-f46-r1	-	-	-	-	93	9.97	115	1.21
mixed-f46-r0	-	-	-	-	92	8.25	113	1.22
mixed-f44-r4	-	-	-	-	87	7.92	111	1.04
mixed-f44-r3	-	-	-	-	88	8.46	109	1.03
mixed-f44-r2	-	-	-	-	89	7.93	109	1.09
mixed-f44-r1	-	-	-	-	85	7.11	107	1.08
mixed-f44-r0	-	-	-	-	87	7.50	109	1.14
mixed-f42-r4	-	-	-	-	82	7.01	105	0.90
mixed-f42-r3	-	-	-	-	83	6.38	104	0.96
mixed-f42-r2	-	-	-	-	81	7.78	104	0.88
mixed-f42-r1	-	-	-	-	83	7.25	105	0.99
mixed-f42-r0	-	-	-	-	85	7.25	105	0.93
mixed-f40-r4	-	-	-	-	80	5.26	99	0.84
mixed-f40-r3	-	-	-	-	79	5.08	98	0.86
mixed-f40-r2	-	-	-	-	79	5.04	95	0.85
mixed-f40-r1	-	-	-	-	78	4.76	98	0.83
mixed-f40-r0	-	-	-	-	80	4.34	100	0.88
mixed-f38-r4	-	-	-	-	76	4.40	95	0.71
mixed-f38-r3	-	-	-	-	75	4.24	95	0.77
mixed-f38-r2	-	-	-	-	74	4.11	92	0.73
mixed-f38-r1	-	-	-	-	76	5.00	95	0.71
mixed-f38-r0	-	-	-	-	74	4.50	94	0.74
mixed-f36-r4	-	-	-	-	71	2.99	88	0.64
mixed-f36-r3	-	-	-	-	69	3.53	88	0.62
mixed-f36-r2	-	-	-	-	70	3.42	90	0.66
mixed-f36-r1	-	-	-	-	70	3.18	89	0.67
mixed-f36-r0	-	-	-	-	71	3.28	89	0.65
mixed-f34-r4	-	-	-	-	67	2.92	83	0.54
mixed-f34-r3	-	-	-	-	68	2.73	85	0.55
mixed-f34-r2	-	-	-	-	67	2.61	83	0.56
mixed-f34-r1	-	-	-	-	67	2.96	84	0.55
mixed-f34-r0	-	-	-	-	68	2.79	85	0.59
mixed-f32-r4	-	-	-	-	63	2.44	78	0.47
mixed-f32-r3	-	-	-	-	64	2.27	81	0.53
mixed-f32-r2	-	-	-	-	64	2.45	79	0.45
mixed-f32-r1	-	-	-	-	64	2.28	80	0.52
mixed-f32-r0	-	-	-	-	62	2.75	80	0.49
mixed-f30-r4	-	-	-	-	61	1.70	75	0.43
mixed-f30-r3	-	-	-	-	59	1.84	74	0.41
mixed-f30-r2	-	-	-	-	59	1.63	74	0.41
mixed-f30-r1	-	-	-	-	60	1.76	75	0.44
mixed-f30-r0	-	-	-	-	60	1.73	74	0.42
TOTALES					6339	369.86	7885	58.60

Tabla C.18: Resultados de GEBL aplicado a PRODIGY con los problemas de la IPC en el dominio Miconic III.

Problemas	TGP		PRODIGY		HAMLET (9 reglas)		PRODIGY-GEBL (73 reglas)	
	Q	T	Q	T	Q	T	Q	T
P001-01	1	0.00	1	0.00	1	0.01	1	0.01
P002-01	1	0.01	1	0.01	1	0.01	1	0.00
P003-02	6	0.03	6	0.03	-	-	8	0.08
P004-02	7	0.04	7	0.02	-	-	9	0.09
P005-02	7	0.04	10	0.02	-	-	10	0.12
P006-02	6	0.03	6	0.04	-	-	8	0.08
P009-03	10	0.04	13	0.02	-	-	15	0.25
P010-03	12	0.03	12	0.03	-	-	12	0.11
P011-03	9	0.04	11	0.18	-	-	9	0.26
P012-03	13	0.03	13	1.01	-	-	13	0.38
P013-03	5	0.04	5	20.85	-	-	13	0.19
P015-03	9	0.03	11	0.02	-	-	9	0.28
P016-03	12	0.03	12	0.02	-	-	12	0.10
P017-03	10	0.04	13	0.02	-	-	15	0.23
P020-03	11	0.13	15	0.02	-	-	15	0.33
P021-03	9	0.53	13	0.02	-	-	15	0.65
P027-04	6	0.04	6	2.25	-	-	10	7.73
P029-04	3	0.04	3	0.01	-	-	3	27.92
P039-04	3	0.05	3	0.00	-	-	3	18.64
P041-04	6	0.03	6	2.31	-	-	10	7.77
P043-04	10	0.20	25	0.03	-	-	16	0.46
P046-04	8	0.08	25	0.03	-	-	15	0.31
P053-04	22	0.58	23	0.03	-	-	11	0.28
P088-05	22	0.67	29	0.04	-	-	17	0.52
P099-06	13	1.57	26	0.04	-	-	22	1.53
TOTALES	221	4.35	295	27.05			272	68.32
P098-05	6	0.05	33	0.06	-	-	-	-
P044-04	4	0.03	-	-	6	0.02	4	0.06
P045-04	4	0.02	-	-	6	0.01	4	0.07

Tabla C.19: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Driverlog I.

Problemas	TGP		PRODIGY		HAMLET (9 reglas)		PRODIGY-GEBL (73 reglas)	
	Q	T	Q	T	Q	T	Q	T
P008-03	6	0.15	-	-	-	-	10	0.30
P014-03	9	0.05	-	-	-	-	10	8.15
P018-03	8	0.14	-	-	-	-	29	1.06
P022-03	8	0.57	-	-	-	-	12	0.30
P023-03	9	0.86	-	-	-	-	16	0.78
P024-03	7	0.68	-	-	-	-	19	0.64
P025-04	10	0.23	-	-	-	-	38	1.48
P026-04	7	0.08	-	-	-	-	22	0.55
P019-03	7	0.07	-	-	-	-	20	0.47
P032-04	8	0.03	-	-	-	-	12	0.27
P035-04	13	0.03	-	-	-	-	19	3.14
P047-04	45	0.25	-	-	-	-	22	0.73
P049-04	10	0.36	-	-	-	-	35	1.10
P050-04	11	0.31	-	-	-	-	16	0.57
P051-04	8	0.36	-	-	-	-	20	0.59
P052-04	8	0.37	-	-	-	-	20	0.59
P054-04	15	0.68	-	-	-	-	36	2.36
P055-04	12	0.55	-	-	-	-	20	0.94
P056-04	13	0.38	-	-	-	-	15	0.40
P057-04	13	0.96	-	-	-	-	23	0.79
P058-04	11	0.67	-	-	-	-	20	0.54
P059-04	13	1.23	-	-	-	-	26	1.50
P060-04	8	0.26	-	-	-	-	20	0.67
P061-04	13	1.79	-	-	-	-	26	1.98
P063-04	13	1.32	-	-	-	-	26	1.72
P064-04	12	0.98	-	-	-	-	21	1.28
P065-04	12	0.34	-	-	-	-	40	2.33
P067-05	12	0.16	-	-	-	-	26	0.80
P068-05	12	0.17	-	-	-	-	26	0.65
P070-05	13	0.15	-	-	-	-	22	0.72
P072-05	11	0.14	-	-	-	-	29	3.90
P074-05	15	0.41	-	-	-	-	30	1.09
P075-05	15	0.41	-	-	-	-	30	1.04
P077-05	15	0.37	-	-	-	-	29	1.36
P078-05	14	0.42	-	-	-	-	28	2.10
P080-05	17	0.88	-	-	-	-	28	1.24
P081-05	17	1.04	-	-	-	-	28	1.22
P084-05	20	0.71	-	-	-	-	27	1.57
P085-05	14	0.85	-	-	-	-	40	3.29
P086-05	14	0.49	-	-	-	-	34	1.58
P087-05	14	0.45	-	-	-	-	34	1.57
P089-05	18	0.74	-	-	-	-	50	3.94
P090-05	16	0.65	-	-	-	-	25	1.34
P091-05	16	0.43	-	-	-	-	19	0.63
P092-05	17	0.95	-	-	-	-	36	1.26
P093-05	17	1.00	-	-	-	-	36	1.27
P094-05	18	1.14	-	-	-	-	51	2.22
P095-05	18	0.71	-	-	-	-	18	0.63
P096-05	8	0.45	-	-	-	-	20	0.85
P097-05	8	0.28	-	-	-	-	20	0.76
TOTALES	648	26.70					1279	70.26

Tabla C.20: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Driverlog II.

Problemas	TGP		PRODIGY		HAMLET (9 reglas)		PRODIGY-GEBL (73 reglas)	
	Q	T	Q	T	Q	T	Q	T
P007-02	11	0.08	-	-	-	-	-	-
P028-04	8	0.05	-	-	-	-	-	-
P030-04	12	0.04	-	-	-	-	-	-
P031-04	9	0.04	-	-	-	-	-	-
P033-04	8	0.04	-	-	-	-	-	-
P034-04	11	0.04	-	-	-	-	-	-
P036-04	12	0.04	-	-	-	-	-	-
P037-04	12	0.05	-	-	-	-	-	-
P038-04	12	0.04	-	-	-	-	-	-
P040-04	8	0.04	-	-	-	-	-	-
P042-04	10	0.13	-	-	-	-	-	-
P048-04	11	0.17	-	-	-	-	-	-
P062-04	15	1.40	-	-	-	-	-	-
P066-05	17	0.34	-	-	-	-	-	-
P069-05	7	0.20	-	-	-	-	-	-
P071-05	10	0.25	-	-	-	-	-	-
P073-05	21	1.25	-	-	-	-	-	-
P076-05	12	0.65	-	-	-	-	-	-
P079-05	16	0.70	-	-	-	-	-	-
P082-05	24	2.10	-	-	-	-	-	-
P083-05	17	1.68	-	-	-	-	-	-
P100-06	19	1.31	-	-	-	-	-	-

Tabla C.21: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Driverlog III.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (14 reglas)	
	Q	T	Q	T	Q	T	Q	T
P001-01	1	0.00	1	0.01	1	0.01	1	0.00
P002-02	4	0.02	4	0.21	4	0.03	4	0.01
P003-02	1	0.00	1	0.01	1	0.01	1	0.01
P006-02	1	0.00	1	0.01	1	0.01	1	0.01
P007-02	4	0.02	4	0.22	4	0.01	4	0.02
P008-02	1	0.01	1	0.01	1	0.01	1	0.01
P009-02	1	0.00	1	0.01	1	0.01	1	0.01
P010-02	1	0.01	1	0.01	1	0.01	1	0.01
P011-02	4	0.02	4	0.21	4	0.02	4	0.02
P013-02	1	0.00	1	0.01	1	0.01	1	0.01
P014-02	4	0.03	4	0.18	4	0.02	4	0.02
P015-02	1	0.00	1	0.02	1	0.01	1	0.00
P016-02	1	0.00	1	0.01	1	0.01	1	0.01
P017-02	4	0.02	4	0.19	4	0.01	4	0.01
P022-04	4	0.03	4	0.28	4	0.02	4	0.01
P024-04	1	0.01	1	0.01	1	0.01	1	0.00
P025-04	4	0.03	4	0.19	4	0.02	4	0.02
P027-05	4	0.03	4	0.20	4	0.02	4	0.01
P032-05	4	0.02	4	0.23	4	0.04	4	0.02
P031-05	6	0.03	10	3.95	6	0.40	10	0.03
P034-05	6	0.02	10	3.63	6	0.39	10	0.03
TOTALES	58.00	0.30	66.00	9.60	58.00	1.08	66	0.27
P004-02	-	-	6	0.36	6	0.02	6	0.02
P005-02	-	-	8	2.32	8	0.61	8	0.03
P012-02	-	-	6	0.37	6	0.03	6	0.02
P018-03	-	-	8	2.43	8	0.57	8	0.03
P019-03	-	-	6	0.77	6	0.02	6	0.01
P020-03	-	-	6	0.36	6	0.02	6	0.02
P021-04	-	-	10	5.84	8	0.03	10	0.03
P023-04	-	-	6	0.38	6	0.04	6	0.02
P026-05	-	-	8	2.32	8	0.58	8	0.02
P028-05	-	-	10	5.99	8	0.02	10	0.03
P030-05	-	-	8	2.53	8	0.74	8	0.03
P033-05	-	-	12	6.81	8	0.61	12	0.04
P035-05	-	-	6	0.38	6	0.04	6	0.02
P036-05	-	-	6	0.41	6	0.02	6	0.02
P037-05	-	-	8	2.21	8	0.62	8	0.03
P039-06	-	-	12	6.37	8	0.03	12	0.05
P029-05	-	-	-	-	10	5.93	12	0.03
P038-06	-	-	-	-	12	5.85	20	1.38
P040-06	-	-	-	-	10	0.78	18	0.06
P055-11	-	-	-	-	-	-	-	-
P059-11	-	-	-	-	-	-	-	-

Tabla C.22: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Zenotravel I.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (14 reglas)	
	Q	T	Q	T	Q	T	Q	T
P041-09	-	-	-	-	-	-	26	0.11
P042-09	-	-	-	-	-	-	28	0.11
P043-09	-	-	-	-	-	-	32	0.13
P044-10	-	-	-	-	-	-	40	0.16
P045-10	-	-	-	-	-	-	38	0.15
P046-10	-	-	-	-	-	-	26	0.10
P047-10	-	-	-	-	-	-	46	0.19
P048-10	-	-	-	-	-	-	24	0.09
P049-10	-	-	-	-	-	-	48	0.20
P050-10	-	-	-	-	-	-	22	0.08
P051-10	-	-	-	-	-	-	22	0.08
P052-10	-	-	-	-	-	-	36	0.15
P053-10	-	-	-	-	-	-	30	0.11
P054-10	-	-	-	-	-	-	46	0.19
P056-11	-	-	-	-	-	-	44	0.38
P057-11	-	-	-	-	-	-	34	0.17
P058-11	-	-	-	-	-	-	60	0.26
P060-12	-	-	-	-	-	-	46	0.34
R001-07	-	-	-	-	-	-	28	0.21
R002-08	-	-	-	-	-	-	38	0.16
R003-08	-	-	-	-	-	-	32	0.12
R004-08	-	-	-	-	-	-	42	0.19
R005-08	-	-	-	-	-	-	36	0.14
R006-08	-	-	-	-	-	-	44	0.19
R007-08	-	-	-	-	-	-	20	0.07
R008-08	-	-	-	-	-	-	32	0.12
R009-08	-	-	-	-	-	-	32	0.12
R010-09	-	-	-	-	-	-	38	0.15
R011-09	-	-	-	-	-	-	48	0.22
R012-09	-	-	-	-	-	-	42	0.17
R013-09	-	-	-	-	-	-	44	0.19
R014-09	-	-	-	-	-	-	30	0.11
R015-09	-	-	-	-	-	-	38	0.16
R016-09	-	-	-	-	-	-	38	0.15
R017-09	-	-	-	-	-	-	36	0.14
R018-09	-	-	-	-	-	-	38	0.15
R019-10	-	-	-	-	-	-	26	0.11
R020-10	-	-	-	-	-	-	26	0.11
R021-10	-	-	-	-	-	-	50	0.21
R022-10	-	-	-	-	-	-	44	0.18
R023-11	-	-	-	-	-	-	54	0.27
R024-11	-	-	-	-	-	-	60	0.33
R025-11	-	-	-	-	-	-	50	0.25
R026-11	-	-	-	-	-	-	40	0.18
R027-11	-	-	-	-	-	-	40	0.19
R028-12	-	-	-	-	-	-	58	0.31
R029-12	-	-	-	-	-	-	64	0.36
R030-12	-	-	-	-	-	-	56	0.29
R031-12	-	-	-	-	-	-	52	0.27
R032-12	-	-	-	-	-	-	52	0.27
R033-12	-	-	-	-	-	-	66	0.35
R034-12	-	-	-	-	-	-	54	0.30
R035-12	-	-	-	-	-	-	52	0.26
R036-12	-	-	-	-	-	-	48	0.24
R037-13	-	-	-	-	-	-	72	0.42
R038-13	-	-	-	-	-	-	54	0.29
R039-13	-	-	-	-	-	-	58	0.30
R040-13	-	-	-	-	-	-	54	0.28
TOTALES							1832	8.87

Tabla C.23: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Zenotravel II.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (406 reglas)	
	Q	T	Q	T	Q	T	Q	T
T-1-10-0	2	4.33	2	0.01	8	0.02	2	2.59
T-1-10-1	1	0.47	1	0.01	1	0.00	1	1.13
T-1-10-2	1	0.00	1	0.01	1	0.00	1	0.00
T-1-5-0	1	0.00	1	0.01	1	0.01	1	0.00
T-1-10-5	1	1.08	1	0.01	1	0.01	1	1.69
T-1-5-8	1	0.02	1	0.00	1	0.01	1	0.21
T-2-10-5	1	0.48	1	0.02	1	0.01	1	1.25
T-2-10-7	1	0.51	1	0.01	1	0.01	1	1.23
T-2-10-8	2	0.49	2	0.01	2	0.12	2	4.07
T-2-5-5	1	0.03	1	0.00	1	0.01	1	0.38
T-2-5-7	1	0.03	1	0.01	1	0.00	1	0.39
T-5-5-6	1	0.03	1	0.01	1	0.13	1	0.25
T-1-10-9	4	5.99	-	-	4	0.01	4	6.69
T-1-5-2	2	0.10	-	-	2	0.01	2	0.83
T-1-5-6	2	0.08	-	-	6	0.01	2	1.03
T-1-5-7	3	0.06	-	-	7	0.01	3	1.27
T-2-5-0-11	5	0.12	-	-	5	0.01	5	1.65
T-2-5-2	4	0.10	-	-	4	0.01	4	2.30
T-2-5-3	5	0.27	-	-	5	0.01	5	3.22
T-2-5-4	4	0.27	-	-	4	0.01	4	1.94
TOTALES	43	14.46			57	0.41	43	32.12
T-5-5-9	10	2.34	-	-	10	0.02	10	17.28
T-2-5-0	3	0.08	-	-	-	-	3	1.47
T-5-5-1	6	1.09	-	-	-	-	6	4.58

Tabla C.24: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Logistics I.

Problemas	TGP		PRODIGY		HAMLET (5 reglas)		PRODIGY-GEBL (406 reglas)	
	Q	T	Q	T	Q	T	Q	T
T-1-5-4	3	0.82	-	-	4	0.01	4	1.81
T-1-5-5	4	0.24	-	-	4	0.01	4	2.06
T-2-5-0-18	8	0.22	-	-	8	7.35	8	4.30
T-2-5-8	6	1.04	-	-	6	0.01	6	3.26
T-5-5-8	13	1.57	-	-	-	-	13	29.18
T-10-5-2	12	0.84	-	-	-	-	13	15.20
T-10-5-4	12	0.84	-	-	-	-	12	18.75
T-10-5-9	22	0.93	-	-	-	-	20	21.46
T-1-10-6	3	0.83	-	-	-	-	3	5.92
T-1-10-8	5	53.83	-	-	-	-	5	12.11
T-1-5-1	6	0.57	-	-	-	-	6	3.45
T-1-5-3	6	0.28	-	-	-	-	6	4.39
T-2-10-0	5	25.98	-	-	-	-	5	17.26
T-2-10-2	6	8.45	-	-	-	-	6	29.85
T-2-5-0-10	10	0.27	-	-	-	-	11	4.95
T-2-5-0-12	9	0.12	-	-	-	-	9	3.86
T-2-5-0-15	8	0.20	-	-	-	-	7	3.62
T-2-5-0-17	9	0.18	-	-	-	-	10	5.88
T-2-5-0-19	9	0.29	-	-	-	-	11	5.13
T-2-5-1	4	0.28	-	-	-	-	4	1.79
T-2-5-6	6	0.12	-	-	-	-	6	5.30
T-2-5-9	11	3.29	-	-	-	-	12	12.71
T-3-7-0-12	17	0.62	-	-	-	-	19	16.04
T-3-7-0-19	10	0.85	-	-	-	-	10	7.38
T-3-7-0-3	13	0.85	-	-	-	-	14	14.33
T-3-7-0-6	8	0.20	-	-	-	-	10	6.34
T-3-7-0-9	12	0.37	-	-	-	-	12	8.72
T-5-5-0	14	0.40	-	-	-	-	17	14.07
T-5-5-15	18	0.35	-	-	-	-	22	23.64
T-5-5-16	14	2.28	-	-	-	-	16	18.23
T-5-5-19	19	0.35	-	-	-	-	21	13.35
T-5-5-2	15	1.47	-	-	-	-	12	8.91
T-5-5-4	16	0.63	-	-	-	-	15	18.78
T-5-5-5	10	1.29	-	-	-	-	10	9.15
TOTALES	343	110.85					359	371.18

Tabla C.25: Resultados de GEBL aplicado a PRODIGY con problemas aleatorios en el dominio Logistics II.