UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MODELO DE LANDMARKS TEMPORALES PARA PROBLEMAS DE PLANIFICACIÓN TEMPORAL ALTAMENTE RESTRINGIDOS

Autor: Eliseo Jorge Marzal Calatayud

Directores: Dra. Dña. Eva Onaindía de la Rivaherrera

Dra. Dña. Laura Sebastiá Tarín

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

Valencia, España

Diciembre 2015

A mi familia.

[1]

[1] *Realizado con* Elfic.

Tabla de Contenidos

Tabla de Contenidos v

Lista de Tablas viii

Lista de Figuras x

Agradecimientos xiii

Resumen xv

Abstract xvii

Resum xix

1. Introducción 1
- 1.1. Antecedentes . . . 2
  - 1.1.1. Breve reseña histórica . . . 2
  - 1.1.2. Perspectiva actual . . . 4
- 1.2. Motivación y objetivos . . . 5
- 1.3. Aportaciones de la tesis . . . 7
- 1.4. Organización del trabajo . . . 8

2. La planificación en Inteligencia Artificial 11
- 2.1. Definición del problema . . . 11
  - 2.1.1. Representación en planificación clásica . . . 12
  - 2.1.2. Representación de un problema de planificación en PDDL . . . 17
- 2.2. Aproximaciones de planificación clásica . . . 20
  - 2.2.1. Planificación de Orden Parcial . . . 21
  - 2.2.2. Planificación basada en grafos . . . 25
  - 2.2.3. Planificación heurística . . . 28
- 2.3. Planificación Temporal . . . 32
  - 2.3.1. Modelos de acciones con duración . . . 33

2.3.2. Representación en planificación temporal 36
2.3.3. Evolución de la planificación temporal 39
2.3.4. Planificadores temporales en la actualidad 44
2.4. Restricciones temporales 48
2.4.1. PDDL2.2 49
2.4.2. PDDL3.0 50
2.5. Conclusiones 51

3. Grafos de Landmarks STRIPS **53**
3.1. Introducción 53
3.2. Ejemplo de aplicación 54
3.3. Conceptos de landmarks 54
3.3.1. Relaciones de orden entre *landmarks* 56
3.3.2. Grafo de landmaks 59
3.4. Métodos para la generación del grafo de *landmarks* 59
3.4.1. La aproximación LM 60
3.4.2. La aproximación DL 65
3.4.3. La aproximación DTG 68
3.4.4. La aproximación Pro 69
3.4.5. Comparativa entre estas aproximaciones 70
3.5. Aproximación FULL 74
3.6. Conclusiones 80

4. Modelo de Landmarks Temporales **83**
4.1. Introducción 83
4.2. Problema de Planificación Temporal con Restricciones 86
4.2.1. Interferencias entre literales del modelo temporal 90
4.3. Modelo de Landmarks Temporales 92
4.3.1. Definición de landmark temporal 93
4.3.2. Representación de *deadlines* en un modelo de landmarks temporales 97
4.3.3. Relaciones de orden entre landmarks temporales 98
4.3.4. Grafo de Landmarks Temporales 99
4.4. Construcción del Grafo de Landmarks Temporales 100
4.4.1. Paso 1: Extracción de landmarks y órdenes 102
4.4.2. Paso 2: Propagación de la información temporal 108
4.4.3. Paso 3: Análisis de las inconsistencias temporales 120
4.5. Conclusiones 124

5. TempLM: Planificador basado en *Landmarks Temporales* 125
5.1. Introducción . . . 125
5.2. Modelo preliminar de TempLM: aproximación CSP . . . 126
5.3. Proceso de búsqueda de TempLM . . . 128
5.3.1. Función de evaluación . . . 129
5.3.2. Generación de sucesores . . . 131
5.4. Proceso de *feedback* de TempLM . . . 139
5.4.1. Refinamiento del grafo de landmarks temporales . . . 141
5.4.2. *Feedback* V1 . . . 145
5.4.3. *Feedback* V2 . . . 146
5.5. Análisis de la complejidad . . . 147
5.6. Conclusiones . . . 149

6. Resultados experimentales 151
6.1. Introducción . . . 151
6.2. Dominios y configuración de los experimentos . . . 153
6.3. Comparativa I: TempLM versus OPTIC, MIPS-XXL y SGPLAN5 . . . 158
6.4. Comparativa II: beneficios de la utilización de landmarks y *feedback* 161
6.4.1. Problemas originales de las competiciones *IPC* 2004 e *IPC* 2006 161
6.4.2. Problemas clasificados como *tight* . . . 163
6.4.3. Problemas clasificados como irresolubles . . . 166
6.4.4. Problemas no clasificados . . . 169
6.5. Comparativa III: comparativa con una nueva aproximación que utiliza landmarks temporales . . . 172
6.6. Conclusiones . . . 175

7. Conclusiones y trabajo futuro 177
7.1. Introducción . . . 177
7.2. Conclusiones . . . 178
7.2.1. Contribuciones de la tesis . . . 178
7.2.2. Publicaciones de la tesis . . . 182
7.3. Trabajos futuros . . . 183

A. Representación de los dominios en PDDL 185
A.1. Especificación del dominio *depots* . . . 185
A.2. Especificación del problema del dominio *depots* del Capítulo 2 . . . 187
A.3. Especificación del problema del dominio *depots* del Capítulo 3 . . . 188

Bibliografía 191

Índice de tablas

3.1. Comparación de *landmarks* entre las distintas aproximaciones. . . . 72
3.2. Comparación de relaciones de orden entre las distintas aproximaciones. 73
3.3. Comparación de los *landmarks* individuales de la mejor aproximación de la Sección 3.4 con las versiones de la aproximación FULL. 79
3.4. Comparación de los *landmarks* disyuntivos de la mejor aproximación de la Sección 3.4 con las versiones de la aproximación FULL. 79

4.1. Representación de *deadlines* en $\mathcal{D}$ 97
4.2. Restricciones de orden de las restricciones PDDL3.0 99
4.3. Definición de $dis^a_E(l_i, l_j)$ y $dis^a_L(l_i, l_j)$ para secuencias de una única acción a y donde a sólo puede tomar un valor 111

6.1. Resumen de los resultados obtenidos 159
6.2. Resultados para los problemas originales de dominios de *IPC* 2004 e *IPC* 2006. 161
6.3. Resultados de problemas *tight* en dominios de la *IPC* 2002 163
6.4. Resultados de problemas *tight* en dominios de la *IPC* 2006, *IPC* 2008 e *IPC* 2011 164
6.5. Número de nodos generados y expandidos para problemas clasificados como *tight* 166
6.6. Resultados de problemas irresolubles de dominios de la *IPC* 2002 . . 167
6.7. Resultados de problemas irresolubles para dominios de la *IPC* 2004 168
6.8. Resultados de problemas irresolubles para dominios de la *IPC* 2008 e *IPC* 2011 169

6.9. Número de nodos generados y expandidos en los problemas irresolubles 170
6.10. Número de problemas resueltos de cada dominio 170
6.11. Número de problemas *tight* y *loose* resueltos de cada dominio 172
6.12. Número de problemas irresolubles detectados de cada dominio 172
6.13. Instancias de problemas utilizadas con el enfoque de Karpas *et al.* . 175

Índice de figuras

2.1. Ejemplo de un problema de transporte 15
2.2. Tipos para el problema de la Figura 2.1 17
2.3. Predicados para el problema de la Figura 2.1 18
2.4. Acción drive para el problema de la Figura 2.1 18
2.5. Objetos para el problema de la Figura 2.1 19
2.6. Situación inicial para el problema de la Figura 2.1 19
2.7. Objetivo para el problema de la Figura 2.1 20
2.8. Representación gráfica de una acción durativa 35
2.9. Funciones para el problema de la Figura 2.1 37
2.10. Acción durativa drive para el problema de la Figura 2.1 38
2.11. Situación inicial con *fluents* para el problema de la Figura 2.1 38
2.12. Situación inicial con *timed initial literals* para el problema de la Figura 2.1 50
2.13. Restricciones de trayectoria para el problema de la Figura 2.1 51

3.1. Ejemplo del dominio *depots* 55
3.2. Ejemplo de órdenes razonables 58
3.3. Grafo de *landmarks* del dominio *depots* con la aproximación LM . . 65
3.4. Resumen aproximaciones 71
3.5. Aproximación FULL 75
3.6. Tiempo de computo de FULL-v3 vs LM vs DL y DTG. 80

4.1. Un problema del dominio *logistics* temporal 84
4.2. Varios planes para el problema de la Figura 4.1 85

4.3. Situaciones de *mutex* potenciales y *no-mutex* en el modelo temporal 91
4.4. *Mutex* potential cuando l_2 es *mutex* potencial con la precondición p y l_1 es un efecto positivo 92
4.5. Representación de un landmark temporal 95
4.6. Planes temporales el problema del dominio *logística*. 96
4.7. Situación inicial y objetivos para el ejemplo utilizado para la construcción del TLG. 101
4.8. *Landmarks* STRIPS del TLG correspondiente al ejemplo 4.7. 103
4.9. TLG para el ejemplo de la Figura 4.7 107
4.10. Propagación del min_v para el ejemplo 4.7 115
4.11. Propagación del max_g para el ejemplo 4.7 116
4.12. TLG con los intervalos de los landmarks actualizados para el ejemplo 4.7 118
4.13. Extracto de la Figura 4.12 con los *landmarks* involucrados para resolver la *inconsistencia2* TLG 122
4.14. TLG final para el ejemplo 4.7 123

5.1. Esquema de TempLM 127
5.2. Aplicación de acciones reversibles para el ejemplo de la Figura 4.7 . 132
5.3. Cálculo del instante de comienzo más temprano para el ejemplo de la Figura 4.7 134
5.4. Cálculo del instante de comienzo más temprano para el ejemplo de la Figura 4.7 134
5.5. Resultado del cálculo del instante de comienzo más temprano para el ejemplo de la Figura 4.7 135
5.6. Cálculo del nodo resultante para el ejemplo de la Figura 4.7 136
5.7. Plan parcial para el ejemplo de la Figura 4.7 138
5.8. TLG para el ejemplo modificado (conseguir sólo el objetivo g =(at P1 S2)) de la Figura 4.7 139
5.9. Plan parcial para el ejemplo modificado de la Figura 4.7 140
5.10. Refinamiento del $TLG_{\Pi'}$ en dos etapas: añadir/borrar literales y completar el TLG 142

5.11. Plan y TLG tras aplicar la acción (Board D2 T1 S1) 143
5.12. $Plan2$ y su TLG tras el proceso de refinamiento 144

6.1. Pegsol (makespan). 173
6.2. Pegsol (time). 173
6.3. Driverlog (makespan). 173
6.4. Driverlog (time). 173
6.5. Zeno (makespan). 174
6.6. Zeno (time). 174
6.7. Rovers (makespan). 174
6.8. Rovers (time). 174
6.9. Satellite (makespan). 175
6.10. Satellite (time). 175

Agradecimientos

A mis directoras de Tesis, Eva y Laura por ser las mejores directoras que haya podido tener, por toda la ayuda y el esfuerzo que han dedicado durante el desarrollo de esta tesis.

A mis padres y padrinos, por su apoyo incondicional en todas mis decisiones.

A Rosa Ana por su paciencia, por su inmensa paciencia, por su compañía y por escucharme.

A Jorge y a Àlex por el tiempo que les he robado, por el tiempo que no he jugado con ellos y por comprender que "Papá está haciendo la tesis".

A mi abuela, esté donde esté, porque siempre me cuidó y siempre me cuidará.

A mis amigos del grupo de investigación, por recordarme que tenía que acabar la tesis y no dejar de preguntar día tras día "si ya tenía la portada".

A todas las personas que de una u otra forma han creído en mí.

Valencia, España — Eliseo Jorge Marzal Calatayud

Diciembre, 2015

Resumen

El objetivo de la planificación temporal automática es la construcción de planes con acciones de diferente duración que necesitan ser programadas adecuadamente con el fin de conseguir los objetivos del problema. En planificación temporal, la optimalidad se mide como la duración del plan más corto. Sin embargo, en muchos problemas del mundo real es necesario gestionar restricciones temporales asociadas a los objetivos del problema que pueden no satisfacerse con el plan de menor duración.

En la Competición Internacional de Planificación del año 2006 se presentó el lenguaje PDDL3.0 y se realizó la primera y única competición de planificadores con gestión de restricciones de trayectorias de estado y preferencias. Concretamente, en esta *IPC* se probaron cuatro dominios con restricciones temporales donde los objetivos deben satisfacerse dentro de un límite de tiempo establecido o *deadline*. Dos planificadores participaron en esta competición aunque ninguno presentó un buen comportamiento respecto al cumplimiento de los *deadlines*. En este tipo de problemas, especialmente cuando se trata de problemas altamente restringidos, es crucial detectar la irresolubilidad de los mismos lo antes posible, y en este punto es donde se establece nuestro objetivo.

Este trabajo de tesis presenta un modelo de *landmarks temporales* que permite identificar rápida y eficientemente la irresolubilidad de problemas de planificación con restricciones. Nuestro modelo incorpora las restricciones temporales del lenguaje PDDL3.0 y extiende el concepto de *landmark* STRIPS al contexto temporal. A cada *landmark temporal* se le asocian tres tipos de intervalos que se actualizan y propagan de acuerdo a las relaciones de orden y restricciones temporales entre ellos.

Los *landmarks temporales* junto con sus relaciones de orden y restricciones temporales forman un grafo de *landmarks temporales* donde se sintetiza las relaciones que existen entre los literales de un plan solución y, consecuentemente, entre sus correspondientes acciones. Cuando se añade nueva información al grafo, se actualizan y propagan los intervalos de los *landmarks*, reflejando una imagen más precisa del plan solución.

Posteriormente, el modelo de *landmarks temporales* se integra en un planificador temporal heurístico independiente del dominio denominado TempLM. TempLM utiliza toda la información extraída del problema para podar los planes parciales en el árbol de búsqueda que no son compatibles con la información del grafo de *landmarks temporales*. Además, se dispone de un proceso de retroalimentación entre el grafo de *landmarks temporales* y el proceso de búsqueda de un plan solución que permite enriquecer el grafo y, asimismo, acotar el espacio de búsqueda.

Los resultados experimentales muestran que esta aproximación detecta rápidamente problemas irresolubles y también es muy efectiva para resolver problemas con restricciones muy ajustadas.

Abstract

Automated temporal planning deals with the construction of plans for problems specified with durative actions of possibly different duration. The goal of temporal planning is to select and schedule the actions so as to achieve the problem goals, and the optimality criteria is the plan makespan. However, many real-word application define goals with time constraints which may not be satisfied with the plan of optimal makespan.

The 2006 International Planning Competition introduced the PDDL3.0 language and organized the first and only track in planning with state trajectory constraints, including time restrictions, soft constraints and preferences. Particularly, four domains that feature deadline constraints were tested at the *IPC* and two planners participated in such competition although no one exhibited a good performance or fully-correction in the fulfillment of deadlines. An early detection of unsolvability in temporal planning problems, specifically those with strict deadlines, is crucial for avoiding an unfruitful exploration of the search space. And this is precisely the objective of this PhD dissertation.

This work contributes with a *temporal landmark-based* model embedded into a temporal planner that allows for a rapid and efficient identification of unsolvable problems with deadline constraints. Our model incorporates the basic model operators of PDDL3.0 for defining temporal constraints and extends the STRIPS *landmark* concept to a temporal context. A temporal landmark is associated to three intervals that denote the time frame of the generation, validity and necessity of the landmark in the plan, respectively. The set of temporal landmarks of a problem along with their ordering relations and temporal restrictions define a temporal landmark graph

which comprises the relationships that must exist between the literals of a solution plan and, therefore, between the corresponding actions. As long as new data is incorporated in the graph, the landmarks intervals are updated and propagated accordingly, thus reflecting a more accurate picture of a solution plan.

The temporal landmark model is integrated in a domain-independent temporal planner named TempLM. During the search process, TempLM uses the temporal landmark graph to prune those partial plans of the search tree that are not compliant with the information of the graph. Additionally, we present a further improvement by which a feedback routine between the landmarks graph and the plan of a node tree is established. This process is used to refine the information of the graph and likewise narrow the search.

In the experimental evaluation, we show the effectiveness of the proposed approach for detecting unsolvability and solving temporal planning problems with tight deadline constraints.

Resum

L'objectiu de la planificació temporal automàtica és la construcció de plans amb accions de diferent durada que necessiten ser programades adequadament amb la finalitat d'aconseguir els objectius del problema. En planificació temporal, l'optimitat es mesura com la durada del pla més curt. No obstant açò, en molts problemes del món real és necessari gestionar restriccions temporals associades als objectius del problema que poden no satisfer-se amb el pla de menor durada.

En la Competició Internacional de Planificació de l'any 2006 es va presentar el llenguatge PDDL3.0 i es va realitzar la primera i única competició de planificadors amb gestió de restriccions de trajectòries d'estat i preferències. Concretament, en aquesta *IPC* es van provar quatre dominis amb restriccions temporals on els objectius han de satisfer-se dins d'un límit de temps establit o *deadline*. Dos planificadors van participar en aquesta competició encara que cap va presentar un bon comportament respecte al compliment dels *deadlines*. En aquest tipus de problemes, especialment quan es tracta de problemes altament restringits, és crucial detectar la irresolubilitat dels mateixos el més prompte possible, i en aquest punt és on s'estableix el nostre objectiu.

Aquest treball de tesi presenta un model de *landmarks temporals* que permet identificar ràpida i eficientment la irresolubilitat de problemes de planificació amb restriccions. El nostre model incorpora les restriccions temporals del llenguatge PDDL3.0 i estén el concepte de *landmark* STRIPS al context temporal. A cada *landmark temporal* se li associen tres tipus d'intervals que s'actualitzen i propaguen d'acord a les relacions d'ordre i restriccions temporals entre ells. Els *landmarks temporals* juntament amb les seues relacions d'ordre i restriccions temporals formen un

graf de *landmarks temporals* on se sintetitza les relacions que existeixen entre els literals d'un pla solució i, conseqüentment, entre les seues corresponents accions. Quan s'afig nova informació al graf, s'actualitzen i propaguen els intervals dels *landmarks*, reflectint una imatge més precisa del pla solució.

Posteriorment, el model de *landmarks temporals* s'integra en un planificador temporal heurístic independent del domini denominat TempLM. TempLM utilitza tota la informació extreta del problema per a podar els plans parcials en l'arbre de cerca que no són compatibles amb la informació del graf de *landmarks temporals*. A més, es disposa d'un procés de retroalimentació entre el graf de *landmarks temporals* i el procés de cerca d'un pla solució que permet enriquir el graf i, així mateix, limitar l'espai de cerca.

Els resultats experimentals mostren que aquesta aproximació detecta ràpidament problemes irresolubles i també és molt efectiva per a resoldre problemes amb restriccions molt ajustades.

Capítulo 1

Introducción

Este trabajo se enmarca en el área de planificación en Inteligencia Artificial y, más concretamente, en la planificación temporal con restricciones. El objetivo de la planificación temporal con restricciones es la obtención de planes con acciones de distinta duración, las cuales necesitan ser programadas adecuadamente para garantizar el cumplimiento de las restricciones temporales impuestas en el problema.

La resolución de un problema de planificación temporal con restricciones puede verse, básicamente, como un problema de búsqueda en un espacio de estados o de planes (parcialmente resueltos). Por tanto, el objetivo de un planificador consiste en guiar la búsqueda para explorar el espacio de búsqueda de la forma más eficiente posible utilizando para ello la información proporcionada por las restricciones temporales del problema. Las restricciones temporales introducen información muy valiosa que permite podar el espacio de búsqueda, pero para ello es necesario realizar un adecuado proceso de razonamiento a partir de toda la información temporal del problema a resolver.

En la Sección 1.1 se realiza un breve recorrido por las distintas técnicas que han surgido desde el comienzo de esta disciplina. En la Sección 1.2 se exponen los objetivos que nos han llevado a realizar este trabajo de tesis. El apartado 1.3 muestra las principales aportaciones de este trabajo. Finalmente, en la Sección 1.4 se resumen los distintos capítulos que componen este documento.

1.1. Antecedentes

La Inteligencia Artificial puede definirse como "*el arte de crear máquinas con capacidad de realizar funciones que realizadas por personas requieren inteligencia*" [94]. Según Bellman, la Inteligencia Artificial es "*la automatización de actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje, ...*" [4].

La historia de esta disciplina comienza formalmente en 1956, cuando se acuñó el término Inteligencia Artificial (I.A.), aunque ya se investigaba en este área desde hacía varios años. En los inicios de esta disciplina, los científicos perseguían un objetivo muy ambicioso: construir una máquina inteligente. Sin embargo, la I.A. ha resultado ser algo mucho más complejo de lo que muchos imaginaron en un principio. A medida que la investigación fue avanzando, se observó que el razonamiento genérico sólo se podía alcanzar mediante la comprensión de ejemplos más específicos de razonamiento humano. Por ello, la I.A. se dividió en diversos campos, cada uno de ellos dedicado a distintas áreas del razonamiento aplicado a la resolución de problemas concretos. Uno de esos campos es la planificación. Según Russell, "la planificación puede considerarse como una clase de solución de problemas en la que se utilizan las creencias sobre las acciones y sus consecuencias para buscar una solución dentro del más abstracto espacio de planes" [98]. Es decir, la planificación automática se centra en la obtención de una secuencia de acciones calculadas por unos sistemas de toma de decisiones.

1.1.1. Breve reseña histórica

Las raíces de la planificación en I.A. se encuentran en la resolución de problemas mediante la búsqueda en un espacio de estados y otras técnicas similares (*GPS* [83], *QA3* [51], etc.), y en las necesidades de la robótica. De hecho, se considera que la planificación como campo específico surgió hace tres décadas con el primer sistema de planificación importante: STRIPS (*STanford Research Institute Problem Solver* [32]). Este planificador se diseñó para controlar a *Shakey*, un robot que se desplazaba por las instalaciones del *SRI*. STRIPS realizó contribuciones muy importantes, como la definición de un lenguaje de especificación de problemas de planificación que ha servido como base para otros lenguajes considerados ahora como estándar.

A partir de STRIPS se produjo un importante auge dentro del área, del que surgieron numerosos planificadores como NOAH [100], SNLP [78], O-Plan [18], UCPOP [88], etc. También se desarrollaron distintos lenguajes con el objetivo de proporcionar una notación común para modelar problemas de planificación y evaluar el rendimiento de los planificadores, entre estos lenguajes, el de mayor éxito fue PDDL (*Planning Domain Definition Language* [48]).En toda esta labor investigadora se pueden observar dos tendencias claras:

- Por un lado, los planificadores orientados a aplicaciones (también denominados dependientes del dominio) obtienen resultados excelentes en dominios específicos. Sin embargo, las técnicas desarrolladas en estos sistemas no son reutilizables en otros contextos.

- Por otro lado, las aproximaciones no orientadas a aplicaciones (también denominadas independientes del dominio) resuelven un conjunto limitado de problemas, pero las técnicas que se desarrollan son totalmente generales y reutilizables.

Las grandes diferencias entre el rendimiento de los planificadores de ambas tendencias provoca una tensión que contribuye a dirigir la investigación en planificación independiente del dominio hacia la solución de problemas más complejos. A principios de los años 90, la mayoría de planificadores se basan en el modelo de planificación de orden parcial *(POCL)*(que define los planes sin necesidad de establecer un orden entre todas sus acciones), y ninguno de ellos puede generar planes de más de 30 acciones [72]. Esta situación, sin embargo, cambió radicalmente con la aparición en 1995 del planificador Graphplan [7], que seguía una nueva aproximación basada en grafos. El éxito de Graphplan revitalizó la investigación en el área de la planificación independiente del dominio, lo que propició el desarrollo de nuevos sistemas basados en técnicas diferentes de las empleadas tradicionalmente:

- En el trabajo de Kautz y Selman [68] se demostró que un algoritmo de satisfactibilidad era capaz de mejorar el rendimiento de Graphplan y de otros algoritmos especialmente diseñados para trabajar con problemas de planificación.

- En el trabajo de Bonet y Geffner [9] se mostró que la utilización de técnicas de búsqueda heurística en los algoritmos de planificación permitía obtener resultados muy interesantes.

El éxito de Graphplan propició la aparición de nuevos sistemas de planificación basada en grafos como el planificador STAN [71] u otros planificadores introducen nuevas características, como la planificación temporal (TGP [108] y TPSys [40]).

Posteriormente, el rendimiento de las técnicas independientes del dominio experimentó mejoras significativas, desarrollándose planificadores como FF [58], LPG [47] o SGPlan [15], que permiten resolver problemas más complejos, lo que se traduce en importantes avances en el área.

Recientemente, varios planificadores de orden parcial con heurísticas basadas en estados han demostrado que la planificación *POCL* sigue siendo competitiva, por ejemplo OPTIC [5] (última versión del planificador POPF [17]) y FLAP [101].

1.1.2. Perspectiva actual

Actualmente, gracias a los avances en los algoritmos de planificación se ha podido hacer frente a problemas más realistas que involucran una gestión explícita del tiempo, planes temporales que contienen acciones con duración, manejo de restricciones temporales, modelos más expresivos de acciones, etc.

En el caso de las extensiones temporales, la motivación es evidente ya que en la mayoría de problemas del mundo real el tiempo juega un papel crucial. Esto, junto con una nueva versión de PDDL, denominada PDDL2.1 [34], que permitía la especificación de acciones con duración, supuso el lanzamiento de la planificación temporal. Un problema de planificación con acciones de distinta duración es un problema más complejo porque el planificador no sólo debe encontrar las acciones apropiadas para alcanzar los objetivos sino también el instante de ejecución adecuado para cada una de ellas. Otro factor importante en la planificación temporal es que se incrementa considerablemente la posibilidad de ejecución de acciones en paralelo. Esto tiene importantes implicaciones en la duración total del plan y, consecuentemente, en la calidad de los planes solución. Entre los planificadores temporales actuales más relevantes se puede destacar a SGPlan5 [16], LPG [44], OPTIC [5], TFD [31], YAHSP2 [109] y DAEYAHSP [69].

En muchas aplicaciones del mundo real es más importante que un plan satisfaga determinadas restricciones temporales que el hecho de obtener el plan de duración mínima. buena calidad. Este es el caso de, por ejemplo, algunos procesos industriales o algunos problemas de logística donde es necesario disponer de los productos antes de un determinado instante de tiempo.

Con la introducción de los *timed initial literals* en PDDL2.2 [60] y de las restricciones de trayectoria de estados en PDDL3.0 [43] se incorpora la posibilidad de establecer restricciones temporales a los problemas. La definición de restricciones temporales en el problema implica garantizar el cumplimiento de todas las restricciones para que el plan sea válido. En la *Competición Internacional de Planificación* de 2006 sólo dos planificadores participaron en la sección de restricciones temporales: MIPS-XXL [28] y SGPLAN5 [16], aunque no presentaron buenos resultados en cuanto al cumplimiento de las restricciones. Posteriormente, OPTIC [5] superó a MIPS-XXL y SGPLAN5. Es importante considerar el hecho de que la inclusión de restricciones temporales en un problema puede implicar que éste sea irresoluble, es decir, que no sea posible encontrar un plan que alcance los objetivos y al mismo tiempo, satisfaga las restricciones. Este es uno de los puntos débiles de los planificadores temporales actuales.

1.2. Motivación y objetivos

La **principal motivación** de este trabajo de tesis reside en diseñar un mecanismo que permita determinar de una forma rápida y eficiente de la irresolubilidad de problemas de planificación con restricciones ya que es en este punto donde los planificadores temporales que incorporan restricciones presentan las mayores dificultades.

El **principal objetivo** de esta tesis se centra en el desarrollo de un modelo de *landmarks temporales* para problemas altamente restringidos, es decir, detectar aquellos problemas de planificación temporal con restricciones que son irresolubles y resolver los problemas que contienen restricciones muy ajustadas y para los cuales sí existe una solución. Este modelo incorporará las restricciones temporales expresadas en PDDL3.0 y se utilizará el concepto de *landmark* en el contexto de la planificación temporal con restricciones. Adicionalmente, el objetivo incluye integrar el

modelo de *landmarks temporales* en un proceso de búsqueda de una solución.

Los *landmarks* STRIPS se han utilizado ampliamente en problemas de planificación demostrando su gran utilidad en el diseño de funciones heurísticas para guiar el proceso de búsqueda. Así, nuestro **primer objetivo** consiste en diseñar un proceso de extracción de *landmarks* STRIPS que mejore los resultados de las distintas técnicas existentes con el objetivo de construir un grafo de *landmarks* STRIPS que posteriormente se empleará para guiar el proceso de búsqueda.

Como este trabajo se centra en la planificación temporal, un **segundo objetivo** de esta tesis es extender el concepto de *landmarks* STRIPS a un contexto temporal. De este modo, se desarrolla un modelo de *landmarks temporales* donde un *landmark* se caracteriza por un conjunto de intervalos temporales que se actualizan y propagan de acuerdo a las relaciones de orden y restricciones temporales entre ellos. Estos *landmarks*, junto con sus relaciones de orden y restricciones temporales, forman el grafo de *landmarks temporales* donde se sintetiza la información temporal del problema.

Durante la construcción del grafo de *landmarks temporales* se puede detectar que un problema es irresoluble por el incumplimiento de alguna restricción del problema; pero hay situaciones de irresolubilidad que involucran a varios *landmarks* y no son fáciles de detectar utilizando únicamente la información temporal. Por ello, el **tercer objetivo** de este trabajo de tesis es desarrollar un planificador temporal heurístico independiente del dominio donde integrar el modelo de *landmarks temporales*. El planificador utilizará el grafo de *landmarks temporales* para podar los planes parciales del espacio de búsqueda que no satisfagan la información del grafo.

La información contenida en el grafo de *landmarks temporales* es de gran ayuda durante el proceso de búsqueda pero el grafo representa únicamente la información del estado inicial. Tras la aplicación de una acción de un plan parcial, el estado cambia, de modo que el grafo de *landmarks* se puede actualizar de acuerdo al nuevo estado para que contenga información más precisa. Por este motivo, el **cuarto objetivo** de esta tesis consiste en desarrollar un mecanismo mediante el cual se asocia un grafo de *landmarks temporales* a cada nodo o plan parcial del espacio de búsqueda, de modo que la información del nodo se utiliza para actualizar el grafo y, a su vez, la información del grafo se utiliza para determinar en cada momento si el plan contenido en el nodo satisface las restricciones del problema. Este mecanismo define un

proceso de retroalimentación entre el árbol de búsqueda y los grafos de landmarks temporales que resulta muy beneficioso a la hora de detectar la irresolubilidad de un problema.

Finalmente, el trabajo de esta tesis también ha venido motivado por las necesidades surgidas en diversos proyectos de investigación, realizados en el grupo de investigación al que pertenece el autor (y en los que él mismo ha participado). La investigación desarrollada en esta tesis está enmarcada, entre otros, en los siguientes proyectos:

- *Interacción multiagente para planificación*, Ministerio de Economía y Competitividad TIN2011-27652-C03-01 (Enero 2012 - Diciembre 2014).
- *Planning, execution and learning architecture*, Ministerio de Educación TIN2008-06701-C03-01 (Enero 2009 - Enero 2012).
- *Adaptación basada en aprendizaje, modelado y planificación para tareas complejas orientadas al usuario*, Ministerio de Educación TIN2005-08945-C06-06 (Diciembre 2005 - Diciembre 2008).

1.3. Aportaciones de la tesis

Las principales aportaciones de este trabajo de tesis son las siguientes:

1. Revisión del estado del arte en el área de planificación, prestando especial atención a los planificadores temporales independientes del dominio que son capaces de manejar restricciones temporales.
2. Desarrollo de una técnica para la extracción de *landmarks* y de sus relaciones de orden. Esta técnica surge de la adecuada combinación de las aproximaciones existentes para la extracción de *landmarks*, tras un estudio minucioso de las fortalezas y debilidades cada una de ellas. La técnica desarrollada mejora los resultados obtenidos con cada aproximación individual.
3. Diseño de un modelo de *landmarks temporales* que permite la detección de problemas irresolubles. Se construye un grafo formado por *landmarks temporales* junto con sus relaciones de orden y restricciones temporales entre ellos.

Los *landmarks temporales* están asociados a un conjunto de intervalos cuya actualización y propagación a través de las relaciones de orden o restricciones del problema, permite el ajuste de dichos intervalos y la detección de inconsistencias entre los mismos. Mientras que la detección de una inconsistencia que se pueda resolver permite enriquecer la información del grafo, la detección de una inconsistencia irresoluble es una indicación de la irresolubilidad del problema.

4. Diseño de un planificador temporal heurístico independiente del dominio donde se integra el modelo de *landmarks temporales*. El planificador aprovecha la información del grafo para podar los planes parciales del espacio de búsqueda que no son compatibles con la información del grafo. También se incluye un nuevo mecanismo que permite la retroalimentación entre el grafo de *landmarks temporales* y el proceso de búsqueda. Mediante la utilización de este mecanismo se actualiza la información del grafo a partir del proceso de búsqueda y, asimismo, se limita el espacio de búsqueda al disponer de una información más exacta en el grafo.

5. Evaluación del planificador desarrollado utilizando distintos dominios de problemas de planificación con incorporación de restricciones temporales. Se compara el rendimiento de este planificador con otros planificadores temporales actuales que manejen restricciones temporales. También se comparan dos versiones del planificador desarrollado: una donde solo se utiliza el modelo de *landmarks temporales* y otra donde se integra el mecanismo de retroalimentación entre el grafo y el proceso de búsqueda. Se analiza tanto la respuesta proporcionada por el planificador como el tiempo requerido para proporcionar una respuesta. En la comparativa de las dos versiones del planificador desarrollado también se analiza el número de nodos generados y expandidos.

1.4. Organización del trabajo

Este trabajo se organiza como sigue:

- **Capítulo 1. Introducción**

En este capítulo se realiza un breve resumen histórico de la disciplina de planificación en I.A. Se concreta también la motivación y los objetivos de este trabajo de tesis, junto con sus principales aportaciones. Por último, se incluye el presente apartado en el que se muestra la organización de esta memoria.

- **Capítulo 2. La planificación en Inteligencia Artificial**

  Este capítulo presenta la definición del problema de planificación y los distintos lenguajes utilizados para la especificación de problemas de planificación, incluyendo los lenguajes que permiten la especificación de las restricciones temporales. Asimismo, se resumen las principales tendencias existentes dentro de la planificación clásica y de la planificación temporal.

- **Capítulo 3. Grafos de Landmarks STRIPS**

  En este capítulo se describe el concepto de *landmark* STRIPS. Se realiza un estudio de las fortalezas y debilidades de las distintas aproximaciones que existen para la extracción de *landmarks* y se presenta una comparativa entre estas aproximaciones. Por último, se describe la aproximación FULL como una combinación de las distintas aproximaciones existentes y se presenta una comparativa entre las aproximación que presentan los mejores resultados en cada dominio frente a la aproximación FULL.

- **Capítulo 4. Modelo de Landmarks Temporales**

  Este capítulo se centra en la arquitectura del modelo de *landmarks temporales* desarrollado. Se definen los distintos elementos que se necesitan, extendidos a un contexto temporal. Se formaliza la definición de *landmark temporal* junto con los elementos que lo caracterizan. Se describe la incorporación de las restricciones temporales al modelo. Por último, se presenta la construcción del grafo de *landmarks temporales* detallando cada paso del proceso e indicando los puntos del proceso que permiten la detección de problemas irresolubles.

- **Capítulo 5. TempLM: Planificador basado en *Landmarks Temporales***

  Este capítulo describe el algoritmo de planificación propuesto. Se describen los pasos del algoritmo y se detalla cómo se utiliza la información del grafo de *landmarks* durante la búsqueda del plan solución. También se detalla el proceso

de retroalimentación entre el grafo de *landmarks temporales* y los nodos del espacio de búsqueda para acotar el espacio de búsqueda.

- **Capítulo 6. Resultados experimentales**

  En este capítulo se analiza el comportamiento del planificador desarrollado de acuerdo con las características que ofrece: detección de problemas irresolubles y resolución de problemas muy restringidos. Se describe los distintos entornos donde se realiza la comparativa y se presenta dicha comparativa frente a los planificadores temporales actuales que permiten el manejo de restricciones temporales.

- **Capítulo 7. Conclusiones y trabajo futuro**

  En este capítulo se exponen las aportaciones más relevantes de este trabajo de tesis, y se indican algunas de las futuras líneas de investigación por las que puede continuarse este trabajo.

Capítulo 2

La planificación en Inteligencia Artificial

En este capítulo se aborda el concepto de planificación clásica en Inteligencia Artificial (IA). En la primera sección del capítulo se explican los elementos necesarios para la definición de un problema de planificación clásica y se comentan algunos lenguajes de especificación de problemas. El resto de secciones abordan las diferentes técnicas de planificación existentes para resolver este tipo de problemas.

2.1. Definición del problema

Un problema de planificación es un problema de búsqueda cuyo objetivo es encontrar una secuencia de acciones que conducen un sistema desde un estado inicial hasta un estado objetivo. Desde el punto de vista de agentes inteligentes, el campo de la planificación aborda la construcción de algoritmos de control que permiten a un agente sintetizar una secuencia de acciones que le conduce a alcanzar sus objetivos [113].

Así, el objetivo de la planificación en IA es el control de sistemas dinámicos. Un sistema dinámico es cualquier entorno que incluye un agente cuyo *estado* evoluciona en el tiempo. Esta evolución puede modelarse como una función de transición que toma como entrada el estado anterior del sistema, que incluyen información acerca del entorno y del agente, y decide cuál será el próximo estado. Se dice que un sistema dinámico es *controlable* si el agente puede influir en las transiciones a través de *acciones*, las cuales pueden afectar al entorno del agente. Por tanto, el agente debe

controlar el comportamiento del sistema dinámico para satisfacer las propiedades de su objetivo. El estado del sistema en el instante *t* dependerá de su estado en *t-1* y de la acción que se aplique en *t-1* [65].

Un problema de planificación se describe mediante un conjunto de acciones, un estado inicial del entorno y una descripción del objetivo a conseguir. Esta visión del problema de planificación, totalmente centrada en el concepto de acción, hereda muchos aspectos del *cálculo de situaciones* desarrollado por McCarthy [79]. En este marco se describe cómo las acciones que ejecuta un agente afectan a las situaciones del entorno, las cuales se especifican en un lenguaje de primer orden. En el cálculo de situaciones, las acciones se componen de:

- *axiomas efecto*: describen cómo las acciones modifican una situación en otra nueva situación.
- *axiomas marco*: describen los aspectos que no cambian en una situación como resultado de la aplicación de una acción.

Los axiomas se especifican mediante relaciones y predicados que describen la configuración de los objetos del entorno y variables estado que permiten distinguir los hechos asociados a una situación de aquellos asociados con otra situación distinta. Dado un conjunto completo de axiomas, es posible (a) deducir la situación resultante de aplicar de una secuencia de acciones y (b) determinar las acciones que es necesario aplicar para producir un cambio en una situación.

Un inconveniente importante del cálculo de situaciones es la dificultad de definir un conjunto completo de axiomas marco para un entorno no trivial. Esta dificultad se debe a que el número de propiedades que no cambian en una situación cuando se aplica una acción es mucho mayor que el número de las propiedades que sí cambian. Este problema se conoce como el **problema marco**.

2.1.1. Representación en planificación clásica

Los modelos de planificación clásica realizan una serie de asunciones sobre el modelo del mundo con el objetivo de restringir la complejidad del problema. Estas asunciones, que se muestran en el trabajo de Ghallab *et al.* [50], son:

1. El sistema tiene un número finito de estados (situaciones).

2. El sistema es completamente observable, i.e., se tiene un conocimiento completo del estado del sistema.

3. El sistema es determinista, i.e., la aplicación de una acción a un estado conduce siempre a un mismo estado.

4. El sistema es estático, i.e., el sistema permanece en el mismo estado hasta que se aplique una acción.

5. Los objetivos son conocidos antes de comenzar la planificación, es decir, los objetivos no cambian.

6. Un plan solución de un problema de planificación es una secuencia de acciones finita y linealmente ordenada.

7. No se contempla el razonamiento temporal y numérico, por lo que la calidad del plan se determina por el número de acciones del mismo.

8. La tarea de planificación consiste en construir un plan completo que satisface el objetivo antes de la ejecución de cualquier parte del mismo.

A pesar de estas simplificaciones, la resolución de un problema de planificación independiente del dominio es un problema muy complejo *PSPACE-completo* [14].

Un problema de planificación clásica se formula a través de siguientes elementos:

1. Un conjunto de fórmulas atómicas, denominadas hechos o literales, que representa la información relevante del problema.

2. Un conjunto de operadores definidos en el dominio del problema.

3. Un conjunto inicial de hechos que forman la situación inicial del problema.

4. Un conjunto final de hechos que deben formar parte de la situación final del problema.

Todos estos elementos se deben especificar en un lenguaje formal. Uno de los primeros lenguajes de modelización fue el lenguaje STRIPS. STRIPS *(STanford Research Institute Problem Solver [32])* ha condicionado la gran mayoría de trabajos sobre planificación desde comienzos de los años 70, gracias a su efectiva solución del problema marco [79] y a su soporte para las estrategias de divide-y-vencerás [42].

A partir del lenguaje STRIPS se han desarrollado distintas extensiones que han dado lugar a otros lenguajes de planificación. Entre estos lenguajes, el de mayor éxito ha sido PDDL (*Planning Domain Definition Language* [48]). PDDL se desarrolló para la competición internacional de planificación (*IPC*) del año 1998 [81] con el objetivo de proporcionar una notación común para modelar problemas de planificación y evaluar el rendimiento de los planificadores. Aparte de STRIPS, PDDL ha recibido influencias de muchos otros formalismos, como por ejemplo ADL (*Action Description Language* [87]).

Desde su creación, PDDL ha supuesto un punto de referencia como lenguaje de modelado para la inmensa mayoría de los planificadores. El objetivo de PDDL es expresar la *física* de un dominio, es decir, los predicados que definen un estado, las acciones que se pueden realizar en el dominio, y los efectos de las mismas, sin proporcionar ningún conocimiento adicional sobre el dominio. PDDL ofrece una gran variedad de características, entre las que destacan:

- Modelo de acciones basado en STRIPS.
- Efectos condicionales y cuantificación universal, tal como se propone en ADL.
- Especificación de acciones jerárquicas. Las acciones se descomponen en subacciones y subobjetivos que permiten abordar problemas más complejos.
- Definición de axiomas del dominio. Los axiomas son fórmulas lógicas que establecen relaciones entre los hechos que se satisfacen en un estado (al contrario que las acciones, que definen relaciones entre sucesivos estados).
- Especificación de restricciones de seguridad. Estas restricciones permiten definir un conjunto de objetivos que deben cumplirse durante todo del proceso de planificación.

Dado el gran número de características que se pueden expresar en PDDL, no existe prácticamente ningún planificador que sea capaz de manejarlas todas. PDDL agrupa estas características en una serie de etiquetas (*requirements*) para que los planificadores puedan comprobar rápidamente si son capaces de manejar un determinado dominio.

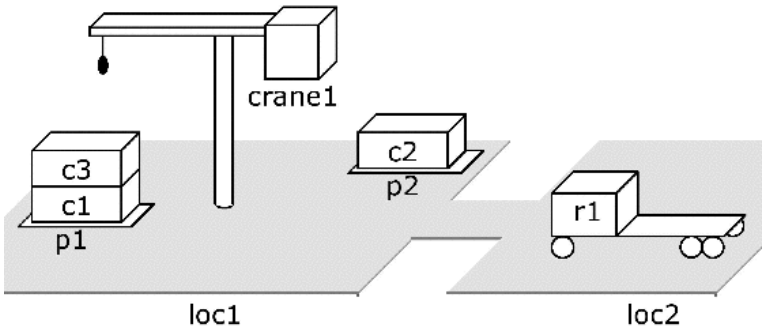

Figura 2.1: Ejemplo de un problema de transporte

Las siguientes definiciones se ha realizado tomando como referencia las expuestas en [50].

Un lenguaje para la modelización de problemas de planificación es un lenguaje lógico de primer orden $\mathcal{L}$ que permite definir completamente el problema de planificación. Los átomos se modelan con símbolos de predicado y sus correspondientes argumentos. Por ejemplo, si disponemos de un dominio de transporte (ver Figura 2.1) se puede representar que un camión se encuentra en una localización con el símbolo de predicado at y obtendríamos el átomo at (t, l). Si sustituimos los argumentos de un átomo por sus posibles valores se obtiene un átomo instanciado que representa una hecho o literal en el sistema. Por ejemplo, en la Figura 2.1 se puede observar que se cumple el literal at (r1, loc2), donde r1 denota al camión del ejemplo y loc2 una de las dos localizaciones existentes. Todos los literales relevantes de un problema constituyen el conjunto denominado $\mathcal{H}$.

Definición 2.1. *Un **estado** $S \subseteq \mathcal{H}$ es un conjunto finito de literales del sistema.* ∎

Un estado puede tener literales positivos, como at (r1, loc2), que representa que r1 se encuentra en la localización loc2, y literales negativos, como ¬at (r1, loc1), que representa que r1 no se encuentra en loc1. En este trabajo asumimos que los estados sólo contienen literales positivos, adoptando así la asunción de mundo cerrado en la especificación de los problemas de planificación.

Definición 2.2. *Un **enunciado de un problema de planificación** es una tripleta $P = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ donde:*

- $\mathcal{I}$, $\mathcal{I} \subseteq \mathcal{H}$, *es un estado que especifica los literales que son verdaderos en la* ***situación inicial*** *del problema.*
- $\mathcal{G}$, estado objetivo, *es una conjunción de literales que representa las condiciones que se desean alcanzar. Un estado $S \subseteq \mathcal{H}$ es un estado objetivo si S satisface $\mathcal{G}$, es decir, si $\mathcal{G} \subseteq S$.*
- $\mathcal{O}$ *es un conjunto de* operadores de planificación. *Cada operador $o \in \mathcal{O}$ es una tripleta $\langle pre(o),\ add(o),\ del(o) \rangle$, donde:*
  - *pre(o), la* lista de precondiciones, *es un conjunto finito de átomos. pre(o) indica las condiciones que un estado debe satisfacer para que el operador se pueda aplicar en ese estado.*
  - *add(o), la* lista de efectos añadidos, *es un conjunto finito de átomos formado por los efectos positivos, es decir, las condiciones que el operador añade al estado cuando se aplica en ese estado.*
  - *del(o), la* lista de efectos borrados, *es un conjunto finito de átomos compuesto por los efectos negativos, es decir, las condiciones que son eliminadas al aplicar el operador.* ∎

Una acción a se obtiene tras la sustitución de los parámetros de un operador por unos valores concretos. Una acción a, por lo tanto, es una instancia concreta de un operador.

Definición 2.3. *Una* ***acción*** *a es una tripleta $a = \langle pre(a),\ add(a),\ del(a) \rangle$ donde $pre(a)$ son los literales que representan las precondiciones de la acción, $add(a)$ son los literales que representan los efectos que se añaden, y $del(a)$ son los literales que representan los efectos que se borran.* ∎

El resultado de aplicar una única acción a a un estado S es:

$$Result(S, \langle a \rangle) = \begin{cases} (S - del(a)) \cup add(a) & pre(a) \subseteq S \\ S & \text{en caso contrario} \end{cases}$$

Si las precondiciones de una acción se satisfacen, los efectos positivos de la acción se añaden al estado, mientras que los efectos negativos se eliminan.

El resultado de aplicar una secuencia de más de una acción a un estado, se define recursivamente como:

$Result(S, \langle a_1, \ldots, a_n \rangle) = Result(Result(S, \langle a_1, \ldots, a_{n-1} \rangle), \langle a_n \rangle)$.

Cuando se aplica una secuencia de acciones vacía, no se produce ningún cambio, i.e., $Result(S, \langle\rangle) = S$.

Definición 2.4. *Un **problema de planificación** $\mathcal{P}$ sobre un enunciado de un problema de planificación P= $(\mathcal{O}, \mathcal{I}, \mathcal{G})$, es una tripleta $(\mathcal{A}, \mathcal{I}, \mathcal{G})$ donde $\mathcal{A}$ es el conjunto de acciones obtenidas a partir de la instanciación de los operadores de $\mathcal{O}$, $\mathcal{I}$ es el estado inicial y $\mathcal{G}$ son los objetivos del problema.* ∎

Definición 2.5. *Un plan $\pi = \langle a_1, \ldots, a_n \rangle$ es un **plan solución** para un problema de planificación $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$, si $\mathcal{G} \subseteq S_n$, donde $S_n = Result(\mathcal{I}, \pi)$.* ∎

Es decir, un plan π representa una secuencia de acciones que permiten conseguir el estado objetivo desde la situación inicial.

2.1.2. Representación de un problema de planificación en PDDL

En PDDL se puede especificar un enunciado de un problema de planificación mediante la declaración de dos bloques: dominio (`domain`) y problema (`problem`). En el bloque del dominio se definen tipos de objetos, predicados y operadores, mientras que en el bloque del problema se definen los objetos, la situación inicial y el objetivo. A continuación se muestra cada uno de estos elementos para un problema de un dominio de transporte cuya situación inicial se representa gráficamente en la Figura 2.1. El objetivo del problema es transportar contenedores entre los pallets de las localizaciones. Los contenedores se cargan y descargan de los camiones utilizando grúas. Los camiones se pueden mover entre pallets o entre localizaciones.

Aunque PDDL permite especificar los elementos de un problema de planificación sin utilizar tipificación, utilizaremos la versión tipificada por ser la más extendida en la actualidad. De esta forma, los tipos de objetos de la Figura 2.1 se definirían del siguiente modo:

```
(:types place locatable - object
        depot distributor - place
        truck hoist surface - locatable
        pallet crate - surface)
```

Figura 2.2: Tipos para el problema de la Figura 2.1

Los tipos `place` y `locatable` son de tipo `object`. El tipo `object` es el tipo base para todos los objetos definidos en un problema especificado en PDDL. De forma análoga se definen el resto de tipos del dominio.

Una vez definidos los tipos de los objetos, se pueden definir los predicados del problema:

```
(:predicates
             (at ?x - locatable ?y - place)
             (on ?x - crate ?y - surface)
             (in ?x - crate ?y - truck)
             (lifting ?x - hoist ?y - container)
             (available ?x - hoist)
             (clear ?x - surface)
             (adjacent ?x - place ?y - place))
```

Figura 2.3: Predicados para el problema de la Figura 2.1

Cada predicado se define por un nombre y unos argumentos, de los cuales se indica su nombre y su tipo. Por ejemplo, `(at ?x - locatable ?y - place)` indicaría que un objeto de tipo `locatable` se encuentra en un objeto de tipo `place`.

Por último, en el bloque del dominio, se definen los operadores. A continuación se muestra la codificación del operador `drive` para este dominio (la definición completa de este dominio se puede ver en el Anexo A):

```
(:action drive
         :parameters (?x - truck ?y - place ?z - place)
         :precondition (and (at ?x ?y) (adjacent ?y ?z))
         :effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Figura 2.4: Acción drive para el problema de la Figura 2.1

En este operador se indica que para mover un camión entre un origen $?y$ y un destino $?z$ necesitamos que el camión se encuentre en $?y$, y que el origen y el destino sean adyacentes. El resultado de la aplicación de este operador daría lugar a un estado en el que el camión está en el destino ?z y ya no está en el origen ?y.

Dentro del bloque del problema, el primer elemento que se define son los objetos disponibles y el tipo de los mismos. En la Figura 2.1 se puede observar que se dispone de dos objetos `loc1` y `loc2` que son de tipo `place`, se dispone de un objeto `r1` del tipo `truck`, etc.

```
(:objects
          loc1 loc2 - place
          r1 - truck
          p1 p2 - pallet
          c1 c2 c3 - crate
          crane1 - hoist)
```

Figura 2.5: Objetos para el problema de la Figura 2.1

El siguiente elemento a definir es el estado inicial. En esta sección se especifican todos los literales que describen la situación inicial. Por ejemplo, el camión `r1` se encuentra en la localización `loc2`, el contenedor `c1` está en el pallet `p1`, representado por el literal `(in c1 p1)`, y ese contenedor es el que está sobre la base del pallet `p1`, representado por el literal `(on c1 p1)`.

```
(:init
      (at p1 loc1)
      (at p2 loc1)
      (in c1 p1)
      (in c3 p1)
      (clear c3)
      (on c3 c1)
      (on c1 p1)
      (in c2 p2)
      (clear c2)
      (on c2 p2)
      (at crane1 loc1)
      (available crane1)
      (adjacent loc1 loc2)
      (adjacent loc2 loc1)
      (at r1 loc2) )
```

Figura 2.6: Situación inicial para el problema de la Figura 2.1

El último elemento que se define son los objetivos. Para este problema el objetivo a conseguir es que el contenedor `c3` se sitúe sobre el contenedor `c2`.

```
(:goal (and (on c3 c2))
```

Figura 2.7: Objetivo para el problema de la Figura 2.1

2.2. Aproximaciones de planificación clásica

En esta sección se describen las principales aproximaciones de planificación clásica independiente del dominio, aquellas que no utilizan información del dominio codificada por el usuario para resolver el problema. La principal ventaja que ofrecen estas técnicas es que son generales y reutilizables.

Las técnicas de planificación que utilizan una representación basada en estados desarrollan un árbol de búsqueda donde un nodo representa un estado del problema, cada arco corresponde a una transición entre estados, y una rama del árbol de búsqueda representa un plan. Habitualmente, en un espacio de estados se emplea una búsqueda hacia delante que consiste en encontrar un estado que satisfaga los objetivos partiendo del estado inicial. Uno de los principales inconvenientes de la búsqueda hacia delante es el elevado factor de ramificación de los árboles de búsqueda que se generan para la resolución de un problema. Para mejorar la eficiencia de los planificadores basados en estados, es necesario utilizar mecanismos que permitan reducir el espacio de búsqueda o bien guiar la búsqueda dentro de este espacio. STRIPS fue uno de los primeros planificadores que utilizó una representación basada en estados pero utilizando una estrategia de búsqueda hacia atrás [32]. La aplicación de una búsqueda regresiva partiendo de los objetivos del problema permite reducir el factor de ramificación pero STRIPS presentaba otros problemas derivados de la dificultad de aplicar una correcta ordenación entre los subobjetivos.

A principios de los 90, la utilización de planificadores basados en estados se relega en favor de otras aproximaciones como las técnicas de planificación de orden parcial. Este es el caso de planificadores como, por ejemplo, UCPOP [88] y VHPOP [117]. Posteriormente surgen otras aproximaciones como la planificación basadas en grafos, iniciada por el planificador Graphplan [7]. En los últimos años, se ha investigado en otras aproximaciones de planificación como *planificación basada en restricciones*, *model-checking* [27], *satisfacibilidad* [67], *aproximaciones CSP* [21], etc. En particular, ha habido un resurgimiento de los planificadores de búsqueda

hacia delante basados en estados como HSP [11], uno de los primeros exponentes de este tipo de aproximación ó FF [58], uno de los planificadores de búsqueda hacia delante más influyentes en el área de planificación.

2.2.1. Planificación de Orden Parcial

A principios de los años 90, muchos de los trabajos en planificación independiente del dominio se centraron en los planificadores de orden parcial *POCL* (*Partial Order Causal Link*). Las aproximaciones *POCL* expanden un árbol de búsqueda mediante la aplicación de una estrategia de razonamiento hacia atrás sobre los objetivos del problema. Cada nodo en el árbol de búsqueda representa un plan de orden parcial. En cada paso del proceso de búsqueda se selecciona el plan más prometedor (según un conjunto de criterios) a expandir. El proceso finaliza cuando se encuentra un plan en el que todos los objetivos han sido resueltos y no existen conflictos entre las acciones del mismo [84].

Los planificadores *POCL* se rigen por el principio de *menor compromiso*, el cual consiste en retrasar la decisión de ordenación entre dos acciones si no existe información suficiente para determinar una relación de orden entre ellas [112]. Un orden parcial entre dos acciones indica que no existe un orden establecido entre ellas y que, por tanto, podrían ejecutarse simultáneamente. Durante el proceso de planificación, se pueden añadir restricciones de orden entre pares de acciones de la forma $a_1 \prec a_2$, indicando que la acción a_1 se debe ejecutar antes que a_2, aunque esto no implica que a_1 tenga que ser necesariamente anterior a a_2. En POCL, el principio de menor compromiso se extiende, además, a la asignación de valores a las variables (parámetros) de las acciones, de forma que un parámetro de una acción no se resuelve hasta que se conoce el valor concreto que se asignará a dicha variable.

2.2.1.1. Elementos de la planificación *POCL*

Las tres operaciones clave de la aproximación *POCL* son (1) introducir o utilizar una acción ya existente en el plan para resolver la precondición de otra acción, (2) resolver un conflicto entre dos acciones, y (3) asignar valores a las variables de las acciones pendientes de resolver, esto es, asignar un valor concreto a un parámetro de una acción. La resolución de la precondición de una acción se realiza mediante la introducción de un *enlace causal*.

Definición 2.6. *Un* ***enlace causal*** *($a_i \xrightarrow{l} a_j$) representa la relación entre dos acciones con respecto a un literal l y determina que la acción a_i ($l \in add(a_i)$) resuelve la precondición l de la acción a_j ($l \in pre(a_j)$).* ∎

Los enlaces causales se utilizan para registrar el propósito de cada acción en el plan. Añadir un enlace causal entre a_i y a_j implica establecer un orden entre ambas acciones: la acción productora a_i se debe ordenar delante de la acción consumidora a_j, $a_i \prec a_j$. Esto no implica que la acción a_i deba ser inmediatamente anterior a la acción a_j, sino que el efecto de a_i debe poder ser utilizado por a_j sin ninguna interferencia o, más concretamente, amenaza.

Definición 2.7. *Una acción a_k se dice que es una* ***amenaza*** *sobre un enlace causal $a_i \xrightarrow{l} a_j$ si:*

- *la acción a_k se puede ordenar entre las acciones a_i y a_j, y*
- *la acción a_k elimina l ($l \in del(a_k)$)* ∎

Una amenaza se puede resolver de tres modos respecto al enlace causal [91]:

- **Por promoción:** la acción conflictiva, a_k, se ordena detrás de la acción consumidora del enlace causal ($a_j \prec a_k$).
- **Por democión:** la acción conflictiva, a_k, se ordena antes de la acción productora del enlace causal ($a_k \prec a_i$).
- **Por separación de variables:** se introducen restricciones para que las variables de las acciones involucradas en el conflicto no puedan tomar los mismos valores.

Las estrategias de promoción o democión no se pueden aplicar si esto supone la violación de los órdenes ya existentes. Lo mismo sucede con el mecanismo de separación de variables si las dos variables en conflicto pueden tomar únicamente un valor que además es compartido. Cuando esto ocurre, el plan en cuestión es descartado.

En la aproximación *POCL* un plan puede contener acciones total o parcialmente instaciadas, i.e., acciones cuyos parámetros son variables. Por este motivo, el tercer tipo de operación que se puede realizar cuando se selecciona un plan es resolver

una variable, que consiste en asignar a la variable un único valor de entre el posible conjunto de valores que puede tomar. Inicialmente, el posible conjunto de valores que puede tomar una variable x, también llamado dominio de x, D_x, se corresponde con los objetos del problema que pertenecen al mismo tipo que la variable x. Durante el proceso de planificación se pueden añadir restricciones de asignación a las variables que pueden ser de la forma $x = y$ o $x \neq y$, donde x es una variable que se corresponde con un parámetro de una acción e y es una constante u otra variable. Por ejemplo, si tenemos una variable $x \in D_{container}$, de acuerdo con la Figura 2.1 x tomará uno de los valores del conjunto {`c1, c2, c3`}. Por otra parte, si durante el proceso de planificación se dispone de la acción `lift ?c ?x ?z ?p` (se puede ver su definición en el Anexo A) donde $?c =$`crane1`, $?z =$`c1` y $?p =$`loc1`, se puede restringir el conjunto de valores que puede tomar la variable $?x$ y el resultado sería $?x =$`c3`. En este caso, el dominio de valores de la variable $?x$ se restringe a $D_x =$`{c3}`.

Formalmente, un plan de orden parcial en de la aproximación *POCL* se define por la tupla $\pi = \langle A, O, B, C, F \rangle$ [115], donde:

- A es el conjunto de acciones del plan $\{a_0, a_1, \ldots, a_n\}$. Cada acción es una versión total o parcialmente instanciada de un operador del problema.
- O es un conjunto de restricciones de orden sobre las acciones del plan, de la forma $a_i \prec a_j$.
- B es el conjunto de restricciones de asignación de valor a las variables de las acciones del plan.
- C es el conjunto de enlaces causales de la forma $(a_i \xrightarrow{l} a_j)$.
- F es el conjunto de tareas pendientes de resolver en el plan parcial. F estará formado por las precondiciones pendientes de resolver de las acciones del plan, las amenazas y las variables de las acciones que aún no tienen un valor concreto asignado. ∎

Definición 2.8. *Un plan de orden parcial $\pi = \langle A, O, B, C, F \rangle$ es **solución** para un problema de planificación $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ si F está vacío y O y B son conjuntos consistentes.* ∎

2.2.1.2. Proceso de planificación *POCL*

El proceso de búsqueda en *POCL* consiste en realizar una búsqueda regresiva sobre los objetivos del problema construyendo un árbol de búsqueda donde cada nodo representa un plan de orden parcial. Este proceso comienza con un plan inicial que contiene únicamente dos acciones, una acción inicial, a_0, y una acción final, a_n, donde $a_0 = \langle \emptyset, \mathcal{I}, \emptyset \rangle$ y $a_n = \langle \mathcal{G}, \emptyset, \emptyset \rangle$. La acción inicial está siempre ordenada antes que el resto de acciones ($a_0 \prec a_i, \forall a_i \in A - \{a_0\}$), mientras que la acción final siempre se ordena detrás ($a_i \prec a_n, \forall a_i \in A - \{a_n\}$). En cada iteración del proceso de búsqueda, se selecciona una tarea pendiente de resolver del conjunto F.

1. Si la tarea pendiente es resolver una precondición de una acción ($l \in pre(a_j)$) se añade un enlace causal ($a_i \xrightarrow{l} a_j$) donde a_i puede ser una acción nueva que se inserta en el plan o bien una acción que ya existe en el plan. En el primer caso, las precondiciones de a_i se añaden al conjunto F de subobjetivos pendientes de resolver; así como las amenazas detectadas al añadir el enlace causal.

2. Si la tarea pendiente es una amenaza, se resolverá mediante promoción, democión o mediante separación de variables si las acciones involucradas en la amenaza tienen variables.

3. Si el planificador trabaja con operadores parcialmente instanciados, en cuyo caso las acciones del plan contendrán variables, otra de las tareas pendientes de resolver sería la resolución de una variable x; en este caso, se generará una ramificación por cada posible valor en D_x.

Este proceso se repite hasta que se encuentre un plan solución.

2.2.1.3. Planificadores *POCL*

El planificador más representativo de la aproximación *POCL* es UCPOP (*Universal quantification Conditional effects Partial Order Planner*) [88], el cual definió el marco formal de los planificadores de orden parcial. UCPOP es un planificador completo y correcto capaz de manejar una sofisticada representación de acciones: además de disyunciones, negaciones y cuantificación existencial, trabaja con efectos condicionales, efectos universalmente cuantificados y precondiciones y objetivos también

universalmente cuantificados. Otro planificador de orden parcial es SPLIN (Sistema de PLanificación INteligente) ([102], [103], [104]), planificador desarrollado en nuestro grupo de investigación, y cuyo rendimiento es superior a UCPOP en dominios estrictamente STRIPS. Debido a que muchos sistemas *POCL* sólo eran capaces de resolver problemas que requerían como máximo una docena de acciones [107], algunos planificadores de orden parcial optaron por realizar mejoras en esta aproximación. Entre ellos, destaca VHPOP (*Versatile Heuristic Partial Order Planner*) [116], que participó en la tercera competición internacional de planificación utilizando operadores completamente instanciados y se le otorgó el premio *Best newcomer*. Además, VHPOP permite elegir entre diversas heurísticas de selección de subobjetivos o amenazas, como *LCFR* [64] y *ZLIFO* [46].

Las técnicas *POCL* también se pueden adaptar fácilmente para permitir el manejo de información temporal y recursos. Esta extensión la han realizado planificadores como parcPLAN [29], ZENO [89] y IxTeT [49].

Actualmente varios planificadores de orden parcial utilizan heurísticas basadas en estados para estimar la distancia al objetivo del problema así como el análisis de alcanzabilidad para mejorar la eficiencia de los algoritmos *POCL*. Los nuevos planificadores que aplican estas mejoras han sido capaces de demostrar que la planificación *POCL* puede ser tan competitiva como las aproximaciones basadas en estados. OPTIC [5] (última versión del planificador POPF [17]) y FLAP [101] son planificadores de orden parcial que realizan una búsqueda hacia delante y combinan la representación de planes de orden parcial con la utilización de estados del problema, lo cual les permite aplicar heurísticas basadas en estados que son mucho más informativas que las clásicas heurísticas de *POCL*.

2.2.2. Planificación basada en grafos

Una de las tendencias que relegó la planificación de orden parcial en favor de otras aproximaciones fue la planificación basada en grafos. Graphplan [7] fue el principal exponente de esta aproximación; además de presentar un enfoque totalmente distinto, superó ampliamente el rendimiento de los planificadores existentes hasta el momento. Aunque el rendimiento de Graphplan ya ha sido superado por nuevas aproximaciones, algunas de sus contribuciones siguen siendo importantes actualmente. Graphplan construye un grafo de planificación para codificar la información del

problema y luego aplica un proceso de búsqueda sobre dicho grafo para construir el plan solución.

Definición 2.9. *Un **grafo de planificación (PG**[1]**)** sobre un problema de planificación $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ es un grafo dirigido multinivel que alterna niveles de nodos literal y niveles de nodos acción:*

(i) *$L_0 = \mathcal{I}$ y $A_0 = \{a | \forall p \in pre(a) : p \in L_0\}$, son el conjunto de nodos literal del nivel 0 y el conjunto de nodos acción del nivel 0, respectivamente. El primero contiene los literales del estado inicial y el segundo las acciones que son aplicables en dicho estado.*

(ii) *En general:*

- *L_i es un conjunto de nodos literal que aparecen tras i niveles o puntos de tiempo[2], es decir $L_i = L_{i-1} \cup \{add(a) | \forall a \in A_{i-1}\} \cup \{del(a) | \forall a \in A_{i-1}\}$*
- *A_i es el conjunto de nodos acción que son aplicables tras i niveles o puntos de tiempo, es decir $A_i = A_{i-1} \cup \{a | \forall p \in pre(a) : p \in L_i\}$* ∎

Denotaremos con el índice g el primer nivel literal del PG donde se consiguen todos los objetivos de $\mathcal{G}$ ($\mathcal{G} \subseteq L_g$). Denotaremos con el índice f el nivel literal del punto fijo; concretamente, L_f representa un nivel en el que no se han añadido nuevos literales respecto al nivel anterior ni es posible añadir nuevas acciones a partir de los literales contenidos en él.

El primer nivel de acción del grafo en el que aparece una acción a indica que es el primer instante en el que a podría ejecutarse. Todas las acciones contenidas en el mismo nivel de acción que a podrían, asimismo, ejecutarse en dicho instante de tiempo. Sin embargo, esto no implica que todas las acciones de un mismo nivel puedan ejecutarse simultáneamente. Dos acciones o dos literales que no pueden ocurrir al mismo tiempo se dice que son *mutuamente excluyentes* (*mutex*)[7].

Definición 2.10. *Dos **acciones** a_i **y** a_j **son mutex** en un nivel t si:*

- *Tipo 1 (efectos inconsistentes): el efecto de la acción a_i es la negación del efecto de la acción a_j.*
- *Tipo 2 (interferencia): la acción a_i elimina una precondición de la acción a_j.*

[1] *Las siglas provienen de Planning Graph*

[2] *Un nivel t de un grafo de planificación está formado por L_t y A_t asociados al punto de tiempo t. Por este motivo, se puede hablar indistintamente de nivel o punto de tiempo t.*

- *Tipo 3 (necesidades competitivas): las acciones a_i y a_j tienen precondiciones que son mutex en el nivel $t-1$.* ∎

Definición 2.11. *Dos **literales** l_i **y** l_j **son mutex** en un nivel t si cada acción que consigue l_i en el nivel $t-1$ es mutex con cada acción que consigue l_j en $t-1$, es decir, si todas las formas de obtener l_i son mutex con todas las formas de conseguir l_j en el nivel $t-1$.* ∎

Como se ha comentado, el planificador basado en grafos más conocido es Graphplan. Su proceso de planificación alterna dos fases diferenciadas:

- **Fase de expansión:** genera un grafo de planificación hasta alcanzar las condiciones necesarias (no suficientes) para encontrar un plan. Es decir, se genera un PG hasta alcanzar el nivel L_g en el que se encuentran todos los objetivos, y ninguno de ellos es *mutex* con otro objetivo.

- **Extracción de la solución:** realiza una búsqueda regresiva en el PG a partir de los objetivos del problema alcanzados en el nivel L_g.

  Dado un conjunto de literales objetivos en el nivel t, el objetivo es encontrar un conjunto de acciones no *mutex* entre sí en el nivel $t-1$ que alcancen dichos literales. Las precondiciones de dichas acciones forman entonces un nuevo conjunto de subobjetivos en el nivel de literal $t-1$. De este modo, si los nuevos subobjetivos se pueden conseguir en $t-1$ niveles sin ser *mutex*, entonces los objetivos originales se pueden conseguir en t niveles. En cambio, si el conjunto de objetivos en el nivel $t-1$ no se puede conseguir sin ser *mutex* entre ellos, Graphplan intenta encontrar una combinación diferente de acciones. Este proceso continúa hasta que se obtiene un plan válido. En caso contrario, queda demostrado que el conjunto original de objetivos no es resoluble en el nivel t y, por lo tanto, se vuelve a la fase de expansión donde el PG se extiende un nivel de literal y un nivel de acción.

Si tras varias iteraciones de las fases de expansión del grafo y extracción de la solución se alcanza el nivel L_f y no se ha obtenido un plan válido, el proceso finaliza indicando que el problema no tiene solución.

El éxito de Graphplan propició la aparición de nuevos sistemas de planificación basada en grafos. El planificador STAN [71], por ejemplo, mejora la eficiencia de

Graphplan introduciendo nuevas técnicas como la explotación de la simetría [36] o el análisis automático de dominios [35]. Otros planificadores introducen nuevas características, como el soporte para las extensiones ADL (IPP [70]), el manejo de incertidumbre y de acciones de sensorización (GP [114]), o planificación temporal (TGP [108] y TPSys [40]).

2.2.3. Planificación heurística

Las aportaciones realizadas dentro de la planificación basada en grafos han supuesto una importante contribución en el campo de la planificación durante los últimos años. Entre estos avances se puede citar la aparición de nuevas técnicas heurísticas que han mejorado significativamente el rendimiento de los planificadores [72]. La planificación heurística utiliza una representación basada en estados con búsqueda hacia delante y hace uso de funciones de estimación para explorar el espacio de búsqueda de una forma más inteligente. Por otro lado, la planificación basada en grafos y, concretamente, la utilización de los grafos de planificación, ha dado lugar a numerosos trabajos e investigaciones en el diseño de heurísticas independientes del dominio, suficientemente informadas para guiar el proceso de planificación de forma eficiente.

El principio general para diseñar una función heurística es formular una versión simplificada (o relajada) del problema, de modo que la solución al problema relajado se puede emplear como heurística en la resolución del problema original para estimar la distancia al objetivo desde un estado del problema; es decir, se calcula una estimación de la distancia que separa un estado del espacio de búsqueda del estado objetivo.

Definición 2.12. *Un* ***problema de planificación relajado*** $\mathcal{R}$ *sobre un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ *es una tripleta* $(\mathcal{A}_\mathcal{R}, \mathcal{I}, \mathcal{G})$ *donde* $\mathcal{A}_\mathcal{R}$ *es un conjunto de acciones donde se eliminan los efectos que se borran :*
$\mathcal{A}_\mathcal{R} = \{(pre(a), add(a), \emptyset) | (pre(a), add(a), del(a)) \in A\}$. ∎

A pesar de esta simplificación, la resolución óptima del problema relajado sigue siendo un problema intratable [14]. Sin embargo, a partir de un problema relajado se puede construir un grafo de planificación relajado cuya información resulta muy útil para el cálculo de heurísticas.

Definición 2.13. *Sea $\mathcal{R} = (\mathcal{A}_\mathcal{R}, \mathcal{I}, \mathcal{G})$ un problema de planificación relajado. El* ***grafo de planificación relajado (RPG[3])*** *es un grafo de planificación (PG) sobre $\mathcal{R}$.* ∎

El *RPG* [58] es similar a un *PG* pero no contempla los efectos borrados de las acciones y, por tanto, no se calcula ningún tipo de *mutex*. El valor de la heurística para un estado se obtiene estimando el coste de alcanzar una solución desde dicho estado en un *RPG*. Existen varias aproximaciones heurísticas para estimar el coste de alcanzar los objetivos a partir de un estado s [11]. Estas heurísticas se basan en la estimación del coste individual de alcanzar un literal l desde un estado s, que se denota como $g_s(l)$, y se calcula a partir de la siguiente fórmula:

$$g_s(l) = \begin{cases} 0 & si\ l \in s \\ min_{a \in A(l)}[1 + g_s(pre(a))] & \text{en otro caso} \end{cases} \tag{2.2.1}$$

donde $A(l)$ representa las acciones que generan l, es decir toda acción a tal que $l \in add(a)$ y $g_s(pre(a))$ representa el coste de conseguir las precondiciones de la acción a desde s. Cada vez que se añade una acción a en un nivel L_i del *RPG* que añade el literal l, el valor $g_s(l)$ se actualiza de la siguiente forma:

$$g_s(l) = min_{a \in A(l)}[g_s(l), 1 + g_s(pre(a))] \tag{2.2.2}$$

La expresión $g_s(pre(a))$ en la Ecuación 2.2.1 y en la Ecuación 2.2.2 representa el coste de un conseguir el conjunto de literales de $pre(a)$ desde un estado s. Sea C un conjunto de literales; el coste de alcanzar C desde s, $g_s(C)$, se puede estimar de distintas formas, lo que da lugar a diferentes heurísticas. La heurística h^{add} se calcula sumando la estimación de conseguir independientemente cada objetivo contenido en $\mathcal{G}$. Esta heurística estima el coste de alcanzar un conjunto C de literales del siguiente modo:

$$g_s^{add}(C) = \sum_{l \in C} g_s(l) \tag{2.2.3}$$

Por tanto, el valor de la estimación de la heurística aditiva del conjunto de objetivos del problema, $\mathcal{G}$, a partir de un estado s se calcula:

$$h^{add}(s) = g_s^{add}(\mathcal{G}) \tag{2.2.4}$$

[3] *Del inglés Relaxed Planning Graph*

Por otro lado, la heurística h^{max} utiliza la estimación de conseguir el objetivo más costoso. Esta heurística estima el coste de alcanzar un conjunto C de literales de la siguiente forma:

$$g_s^{max}(C) = \max_{l \in C} g_s(l) \tag{2.2.5}$$

Por tanto, el valor de la estimación de la heurística h^{max} del conjunto de objetivos del problema, $\mathcal{G}$, a partir de un estado s se calcula como:

$$h^{max}(s) = g_s^{max}(\mathcal{G}) \tag{2.2.6}$$

El valor de la heurística h^{max} coincide con el primer nivel del RPG en el que aparecen simultáneamente todos los literales de $\mathcal{G}$.

La heurística h^{add} es no admisible, aunque suele ser una heurística muy informada. Además, su cálculo puede realizarse muy rápidamente en cada estado del espacio de búsqueda. La heurística h^{add} fue originalmente utilizada en los planificadores UNPOP [80] y HSP [13]. HSP la utilizó en la primera competición internacional de planificación consiguiendo unos resultados muy buenos. En cambio, la heurística h^{max}, introducida por el planificador *HSP2* [10], es admisible pero no es una heurística muy informada.

Existe otro conjunto de heurísticas llamadas h^m que asumen que el coste de alcanzar un conjunto de literales se puede estimar calculando el coste de alcanzar sólo un subconjunto de tamaño m [53]. Este subconjunto de literales debe ser, sin embargo, el subconjunto de los m literales más costosos de alcanzar. En el caso de la heurística h^1, sólo se tendría en cuenta el coste de alcanzar el objetivo más costoso. Esta heurística coincide con h^{max}. Para subconjuntos de tamaño dos (h^2), la heurística es similar a la que se encuentra implícitamente en Graphplan, el cual se puede ver como un planificador heurístico que utiliza una función de estimación h^{PG} y un algoritmo de búsqueda estándar. La heurística $h^{PG}(s)$ se corresponde con el índice del primer nivel del grafo que contiene los literales de s, y éstos no son *mutex* dos a dos [53]. Las heurísticas h^m son admisibles y de coste polinomial. Para valores de $m \geq 2$, la heurística proporciona valores heurísticos más aproximados al coste real aunque la función es bastante más costosa de calcular.

Uno de los planificadores más eficientes basados en esta aproximación es FF (*Fast Forward*) [59], que comparte las ideas básicas de HSP. La técnica que utiliza FF calcula una solución, denominada plan relajado, para un problema relajado sobre

un *RPG*. La longitud o número de acciones del plan relajado proporciona una estimación de la dificultad de la solución del problema y esta estimación se utiliza para guiar un proceso de búsqueda local similar a *Hill-climbing*, que se combina con una búsqueda sistemática en anchura para evitar los mínimos locales. Además, la información contenida en el plan relajado se utiliza como guía para la selección de las acciones durante el proceso de planificación, reduciendo el factor de ramificación de la búsqueda sistemática. Este método, tal y como demuestra su participación en la segunda competición internacional de planificación, mejora los resultados de HSP. Una diferencia fundamental entre ambos sistemas es que distintas versiones de HSP utilizan las heurísticas h^{max} o h^2 que son admisibles mientras que la heurística de FF no lo es, por lo que no existe garantía de que el algoritmo utilizado para extraer el plan obtenga el plan más corto.

Como se ha visto, los *RPG* proporcionan información muy útil para el cálculo de distintas funciones heurísticas pero también se pueden utilizar para extraer el esqueleto de un plan, el cual representa una versión incompleta ó versión parcial de un plan solución que se puede completar o refinar. Este es el caso de 4SP([85], [86], [105]) donde se combinan las técnicas basadas en grafos con técnicas *POCL*. 4SP es un planificador que integra una técnica de preproceso basada en grafos mediante la cual se construye el esqueleto de un plan y un planificador de orden parcial (SPLIN [103]) que toma el esqueleto del plan como entrada y lo refina hasta obtener un plan solución. Los resultados obtenidos con 4SP demuestran que una integración adecuada de una técnica de preproceso basada en grafos y un *POCL* puede obtener buenos resultados en términos de reducción del espacio de búsqueda.

Entre las técnicas utilizadas para mejorar la eficiencia de los planificadores están los trabajos centrados en encontrar órdenes entre objetivos de alto nivel u objetivos del problema y que, posteriormente, se extendieron para encontrar órdenes entre los subobjetivos que aparecen en el problema. Estos subobjetivos, llamados *landmarks*[93], son literales que deben ser ciertos en algún punto de todos los planes solución. Los *landmarks* se han utilizado para planificar en entornos STRIPS y se ha demostrado su eficacia para guiar el proceso de búsqueda hacia la solución. En el Capítulo 3 se introducirá con más detalle esta aproximación.

2.3. Planificación Temporal

Una de las asunciones de la planificación clásica es que las acciones son instantáneas y no tienen duración. Los avances en los algoritmos de planificación en IA han permitido hacer frente a problemas más realistas que involucran una gestión explícita del tiempo, planes temporales que contienen acciones con duración, manejo de restricciones temporales, modelos más expresivos de acciones para especificar problemas del mundo real (modelos conservativos de acciones frente a los modelos no conservativos), utilización de técnicas heurísticas para mejorar el rendimiento (estrategias para el cálculo de las estimaciones y orientación de la búsqueda), etc.

En el caso de las extensiones temporales, la motivación es evidente ya que en la mayoría de problemas del mundo real el tiempo juega un papel crucial. Esto se traduce en dominios de planificación donde la ejecución de cada acción no repercute de igual forma en la duración del plan. Por ejemplo, en la modelización de un problema real como el que se muestra en la Figura 2.1, una acción para cargar un paquete en un camión puede no tener la misma duración que una acción de movimiento; incluso acciones de transporte entre distintas ciudades de origen y destino no tendrán la misma duración porque dependerá de la distancia que separa ambas ciudades. Un problema de planificación con acciones de distinta duración es un problema más complejo porque el planificador no sólo debe encontrar las acciones apropiadas para alcanzar los objetivos sino también el instante de ejecución adecuado para cada una de ellas, lo cual implica espacios de búsqueda más grandes. En otras palabras, la planificación debe incluir un razonamiento temporal para asignar las acciones en el tiempo. Otro factor importante en la planificación temporal es que se incrementa considerablemente la posibilidad de ejecución de acciones en paralelo. Esto tiene importantes implicaciones en la duración total del plan ya que en la planificación temporal la optimalidad del plan no se mide en términos del número de las acciones sino en términos del *makespan* (duración) del plan: la duración de un plan determinado puede ser mayor que la de otro plan con más acciones si las acciones de éste último son de menor duración. En consecuencia, una cuestión importante en la planificación temporal es garantizar que el planificador devuelve el plan con menor duración total, es decir el plan con *makespan* óptimo. Con el fin de obtener planes de

mejor calidad y más realistas, es necesario una utilización más precisa de la concurrencia entre las acciones, aprovechando así al máximo las posibilidades de solape de las acciones con duración o acciones durativas. Encontrar planes temporales óptimos es una tarea compleja y que resulta intratable en muchos problemas. Por tanto, la aplicación de heurísticas eficientes para encontrar buenos planes temporales es un tema de actualidad en la investigación en planificación temporal.

En esta sección se analizan las características que definen un problema de planificación temporal y se presenta una revisión de las aproximaciones más relevantes.

2.3.1. Modelos de acciones con duración

Algunos planificadores incorporan acciones durativas sobre un modelo de acciones clásico conocido como modelo *conservativo* [108]. El resultado es una modesta extensión donde dos acciones no pueden solapar *de ninguna forma* si tienen conflictos entre sus precondiciones o efectos. En cambio, los modelos de acciones *no-conservativos* permiten un mayor número de combinaciones temporales de las acciones durativas, dando así lugar a un mayor nivel de concurrencia entre las acciones.

2.3.1.1. Modelo de acciones conservativo vs. no conservativo

La alternativa más sencilla para incorporar el tiempo en un modelo de planificación clásica es añadir duración a las acciones, sin ninguna modificación adicional en el modelo de acciones. Esto se corresponde con una planificación temporal bajo un modelo *conservativo* de acciones. Se denomina de este modo porque *conserva* la misma estructura de acciones propuesta en PDDL, pero contemplando la duración de las acciones. En el modelo conservativo se asume que:

- Una acción se ejecuta desde que comienza hasta que finaliza.
- Todas las precondiciones deben satisfacerse en el comienzo de la acción y durante todo el periodo de ejecución de la acción.
- Los efectos de la acción sólo se garantizan al final de la ejecución de la acción.

Estas asunciones restringen la forma en que las acciones se pueden ejecutar. Concretamente, dos acciones no pueden solapar de ninguna forma si un efecto o

precondición de una acción es la negación de un efecto o precondición de la otra. La principal desventaja del modelo conservativo es que resulta demasiado limitado para algunos dominios temporales ya que no permite diferenciar sintáctica ni semánticamente entre precondiciones y condiciones *invariantes*, ni entre efectos que se producen al principio o final de la acción [33, 34].

Para hacer frente a las limitaciones de expresividad del modelo conservativo de acciones, Fox y Long propusieron una versión extendida de PDDL denominada PDDL+ [33]. PDDL+ permite especificar dominios de planificación en una organización de cinco niveles que involucran tiempo y recursos numéricos. De los cinco niveles, sólo los cuatro primeros fueron aceptados para formar una nueva versión de PDDL, denominada PDDL2.1 [34], que se utilizó por primera vez en la competición internacional de planificación del año 2002 [63].

El modelo propuesto en el nivel 3 de PDDL2.1, denominado modelo no conservativo, permite gestionar un mayor grado de concurrencia entre las acciones del problema que el modelo conservativo. Un modelo de acciones no conservativo permite un mayor grado de detalle en la especificación de las condiciones y efectos de las acciones, pudiendo así definir un modelado más preciso de las transiciones entre estados durante el intervalo de ejecución de las acciones. Todo ello permite obtener planes de mejor calidad, adaptados a las necesidades del problema real, dando así un mayor grado de concurrencia. Sin embargo, esto también implica una tarea de planificación más compleja debido a que dos acciones que interactúan negativamente pueden no comenzar en el mismo instante de tiempo pero podrían solaparse posteriormente en el tiempo y, por tanto, el espacio de búsqueda se incrementa. De acuerdo con el nivel 3 de PDDL2.1 una acción durativa se puede definir del siguiente modo.

Definición 2.14. *Una **acción durativa** $a \in \mathcal{A}$ está formada por los siguientes componentes (ver Figura 2.8):*

- Condiciones $Cond(a)$. Los tres tipos de condiciones de una acción durativa a son: $SCond(a)$, conjunto de condiciones que se deben cumplir al comienzo de la acción; $Inv(a)$, conjunto de condiciones que se deben cumplir durante la ejecución de la acción y $ECond(a)$, conjunto de condiciones que se deben cumplir al final de la acción. Es decir, $Cond(a) = SCond(a) \bigcup Inv(a) \bigcup ECond(a)$.
- Duración. La duración de una acción a es un valor positivo representado por $dur(a) \in \Re+$.

- Efectos $Eff(a)$. Los dos tipos de efectos de una acción durativa a son $SEff(a) = SAdd(a) \bigcup SDel(a)$, efectos positivos y negativos, respectivamente, que se añaden o borran al comienzo de la acción, y $EEff(a) = EAdd(a) \bigcup EDel(a)$, efectos positivos y negativos, respectivamente, que se añaden o borran al final de la acción. Además, $Eff(a) = SEff(a) \bigcup EEff(a)$. ∎

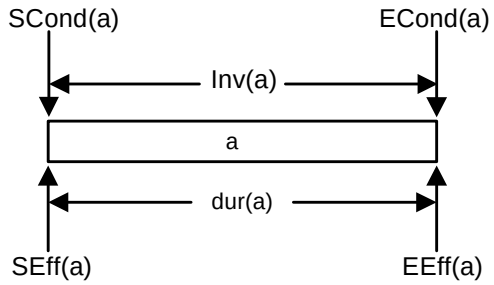

Figura 2.8: Representación gráfica de una acción durativa

A pesar del éxito de PDDL2.1 para el modelado de acciones durativas, esta extensión temporal del lenguaje PDDL todavía presenta una importante limitación: las condiciones y efectos están asociadas a los puntos extremos de las acciones, independientemente de las condiciones invariantes, las cuales deben persistir durante todo el intervalo de ejecución de la acción. Por el contrario, otros planificadores temporales tales como ZENO [89], Sapa [23] o VHPOP [117], trabajan con condiciones y efectos que pueden cuantificarse temporalmente dentro del intervalo de ejecución de la acción. En estos casos, las condiciones se pueden requerir en cualquier instante de tiempo a lo largo del intervalo de ejecución de la acción, o durante un subintervalo específico. Análogamente, un efecto puede generarse en un punto intermedio dentro del intervalo de ejecución de la acción y puede ser válido instantáneamente o durante cierto tiempo. Este tipo de modelos temporales son más genéricos, más expresivos y permiten razonar con restricciones definidas sobre los objetivos, ventanas temporales y eventos exógenos (hechos que se convierten en verdadero o falso en puntos de tiempo predeterminados, con independencia de las acciones en el plan) [26, 108].

2.3.2. Representación en planificación temporal

Los diferentes tipos de modelos temporales mencionados anteriormente difieren en expresividad y en la semántica de los modelos de acciones. En general, la expresividad está determinada por la información temporal asociada a los literales y acciones del modelo. Por tanto, un literal estará asociado al período de tiempo en el cual el literal es cierto o válido y las acciones estarán asociadas con su duración o intervalo de ejecución.

En concreto, asumiremos la semántica del modelo temporal definido en PDDL2.1 [37] ya que resulta apropiado para la representación de problemas de planificación con referencias explícitas de tiempo. A continuación se presentan algunos de los elementos del modelo temporal PDDL2.1:

Definición 2.15. *Un* ***problema de planificación temporal*** $P = \langle \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *es una tripleta donde* $\mathcal{A}$ *es el conjunto de acciones durativas que pueden aplicarse en el dominio e* $\mathcal{I}$ *y* $\mathcal{G}$ *son los conjuntos de literales que representan la situación inicial y objetivo, respectivamente.* ∎

Definición 2.16. *Un* ***plan temporal*** Π *es un conjunto de pares de la forma* (a, t)*, donde* $a \in \mathcal{A}$ *y* t *es el instante de comienzo de la ejecución de la acción durativa* a*.* ∎

La duración (*makespan*) de un plan temporal Π es:

$$dur_{\Pi} = max_{\forall (a,t) \in \Pi} \big(t + dur(a)\big) \tag{2.3.1}$$

Es decir, la duración de un plan temporal es equivalente al instante de finalización de la última acción del plan.

Definición 2.17. *Un* ***estado temporal*** (S_t) *es un conjunto de literales resultado de aplicar el conjunto de acciones de* Π *en* $\mathcal{I}$ *(se denota por* $\mathcal{I} \rightarrow_{\Pi} S_t$*), asumiendo que las acciones en* Π *son aplicables en el estado correspondiente. Formalmente :*

$$S_{ts} = \mathcal{I} \quad - \quad \left(\bigcup_{\substack{\forall (a,t') \in \Pi: \\ t' \leq t}} SDel(a) \right) \cup \left(\bigcup_{\substack{\forall (a,t') \in \Pi: \\ t' \leq t}} SAdd(a) \right)$$

$$S_t = S_{ts} \quad - \quad \left(\bigcup_{\substack{\forall (a,t') \in \Pi: \\ t' + dur(a) \leq t}} EDel(a) \right) \cup \left(\bigcup_{\substack{\forall (a,t') \in \Pi: \\ t' + dur(a) \leq t}} EAdd(a) \right)$$

■

El estado S_{ts} es equivalente al estado $S_{t+\epsilon}$ donde se considera la presencia de un epsilon ($\epsilon > 0$) para resolver el problema de la simultaneidad. El valor de ϵ se utiliza en el nivel 3 de PDDL2.1 para expresar la duración de una acción instantánea y es un valor tan pequeño que no afecta a la correctitud del proceso [39]. Dado un estado S_t y un par (a, t), la acción a es **aplicable** en S_t si:

- $SCond(a) \subseteq S_t \wedge$
- $Inv(a) \subseteq S_t \wedge Inv(a) \subseteq S_{t+1} \wedge \ldots \wedge Inv(a) \subseteq S_{t+dur(a)} \wedge$
- $ECond(a) \subseteq S_{t+dur(a)}$.

Definición 2.18. *Un plan temporal* Π *es un* ***plan solución para un problema de planificación temporal*** $P = \langle \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *si* $\mathcal{I} \rightarrow_\Pi \mathcal{G}$*, donde* $S_0 = \mathcal{I}$ *y* $\mathcal{G} \subseteq S_{dur_\Pi}$

■

2.3.2.1. Representación temporal con el lenguaje PDDL2.1

En PDDL2.1 la información temporal de un problema se representa mediante funciones y operadores que se declaran en el bloque del dominio y *fluents* que se declaran en el bloque del problema. Para ilustrar cómo se definen estos elementos se utilizará el problema del dominio de transporte cuya situación inicial se representa gráficamente en la Figura 2.1.

```
(:functions (distance ?x - place ?y - place)
            (speed ?t - truck))
```

Figura 2.9: Funciones para el problema de la Figura 2.1

En este caso se definen dos funciones numéricas `distance` y `speed` que devuelven un valor en función de los valores de los parámetros. Estas funciones se pueden utilizar para la definición de la duración en un operador como puede verse en el operador de la Figura 2.10.

Dada una acción durativa a, las condiciones $SCond(a)$ se representan como condiciones `at start`, $ECond(a)$ se representan como `at end` e $Inv(a)$ se representa como `over all`. De la misma forma, los efectos $SEff(a)$ se representan como `at`

```
(:durative-action drive
         :parameters (?x - truck ?y - place ?z - place)
         :duration (= ?duration (/ (distance ?y ?z) (speed ?x)))
         :condition (and (at start (at ?x ?y))
                         (over all (adjacent ?y ?z)))
         :effect (and (at start (not (at ?x ?y)))
                      (at end (at ?x ?z))))
```

Figura 2.10: Acción durativa drive para el problema de la Figura 2.1

`start` y los efectos $EEff(a)$ se representan como `at end`. El operador `drive` de la Figura 2.10 indica que para mover un camión entre un origen $?y$ y un destino $?z$ es necesario que el camión se encuentre en $?y$ al comienzo de la acción, y que el origen y el destino sean adyacentes durante todo el intervalo de duración de la acción. El resultado de la aplicación de este operador dará lugar a un estado al comienzo de la acción en el cual el camión ya no estará en el origen $?y$, y otro estado al final de la acción donde el camión se encontrará en el destino $?z$. La duración del operador `drive` se calcula dividiendo el resultado de la función `distance` entre dos lugares de tipo `place` por el resultado de la función `speed` de un `truck`.

Dentro del bloque del problema, en la sección donde se define el estado inicial, se definen los *fluents* que determinan el valor que devuelve una función de acuerdo a los parámetros de la misma. Los *fluents* son literales numéricos o asignaciones de valores a funciones que pueden cambiar en el tiempo. La situación inicial es la misma que se ha definido en la Sección 2.1.2 y únicamente se añade la información referida a los *fluents*.

```
(:init
      ...
      (= (speed r1) 2)
      (= (distance loc1 loc1) 0)
      (= (distance loc1 loc2) 20)
      (= (distance loc2 loc1) 20)
)
```

Figura 2.11: Situación inicial con *fluents* para el problema de la Figura 2.1

En este ejemplo, la velocidad del camión `r1` es 2 y la distancia entre dos localizaciones toma distintos valores en función de los parámetros de la función.

2.3.3. Evolución de la planificación temporal

En esta sección se presenta un resumen de la evolución de los planificadores temporales, excepto los más recientes y representativos que se presentan en la Sección 2.3.4. Comenzaremos con la aproximación basada en grafos, aunque es más reciente que otros paradigmas, por las siguientes razones:

- Ha generado un renovado interés en la planificación automática para hacer frente a capacidades más realistas (planificación neoclásica [50]).
- Se puede adaptar fácilmente para trabajar con acciones durativas (tanto en un modelo conservativo como no conservativo) [39, 108].
- Garantiza ciertas propiedades teóricas relevantes para el diseño de un planificador temporal óptimo [7, 39, 108].
- Se ha utilizado como base para el cálculo de estimaciones heurísticas en muchos planificadores basados en estados y también en planificadores *POCL*.

2.3.3.1. Planificación basada en grafos

Un grafo de planificación temporal mantiene una noción implícita del tiempo mediante la generación de los niveles de acción y literal: las acciones se ejecutan en los niveles $A_{[t]}$, $A_{[t+1]}$, $A_{[t+2]}$ y así sucesivamente, mientras que los literales se generan en los niveles $L_{[t+1]}$, $L_{[t+2]}$, $L_{[t+3]}$, etc., como consecuencia de la ejecución de las acciones. La información contenida en los niveles de acción y literal proporcionan una medida optimista de accesibilidad. Es decir, si una acción no está presente en el nivel $A_{[t]}$, se sabe con certeza que esta acción no es alcanzable en el instante temporal t, y lo mismo ocurre con los literales.

El éxito de Graphplan, en el ámbito de la planificación clásica motivó el uso de técnicas de planificación basada en grafos aplicadas a la planificación temporal. TGP es el primer planificador temporal que generaliza la representación basada en grafos de planificación para manejar, sustituyendo los niveles tradicionales del grafo por niveles temporales, y utiliza el grafo *temporal* resultante para realizar la búsqueda de

un plan [108]. TGP utiliza un modelo conservativo de acciones durativas e introduce algunas claves para trabajar con acciones durativas:

- Utiliza un grafo de planificación temporal donde los niveles se etiquetan con marcas de tiempo (el tiempo se modela por $\Re$ y su orden cronológico).
- Extiende el razonamiento de *mutex* distinguiendo entre *mutex* condicionales, relaciones de exclusión mutua en un intervalo temporal, y *mutex* permanentes, relaciones de exclusión mutua que se mantienen a lo largo de todo el grafo de planificación temporal. En concreto, TGP calcula *mutex* proposición-proposición, proposición-acción y acción-acción.
- El grafo de planificación temporal se construye añadiendo nuevos niveles de acciones y proposiciones y propagando la información de las relaciones *mutex*. Cuando una acción a_i se inserta en el nivel $A_{[t]}$ genera sus efectos $SEff(a_i)$ en el nivel de proposición $P_{[t+\epsilon]}$ y sus efectos $EEff(a_i)$ en el nivel $P_{[t+dur(a_i)]}$ en lugar del nivel de $P_{[t+1]}$, es decir, los niveles en el grafo de planificación temporal no son equidistantes y dependen de la duración de las acciones generadas.
- TGP extrae un plan realizando una búsqueda en el grafo de planificación temporal.
- Las propiedades de Graphplan se mantienen en un entorno temporal, y se garantiza el plan de *makespan* óptimo. Los planes pueden contener acciones en paralelo, y el grado de paralelismo es mayor que en la planificación clásica, ya que dos acciones pueden solaparse de diferentes formas.
- Al gestionar el tiempo de manera explícita se pueden tratar otros aspectos temporales en un grafo de planificación temporal tales como: i) eventos exógenos no condicionales (restricciones temporales en forma de ventanas temporales), ii) objetivos con restricciones, iii) especificación de una duración máxima para el plan y otros.

Otro planificador que combina las ideas de Graphplan y TGP para manejar acciones durativas es TPSYS [41]. Aunque inicialmente TPSYS sólo manejaba el modelo de acciones conservativo como TGP, posteriormente se extendió para manejar el

modelo de acciones no conservativo de PDDL2.1. Las aportaciones clave de este planificador son las siguientes:

- El razonamiento sobre *mutex* es más complejo que en TGP debido a la semántica del modelo de acciones no conservativo. Se distinguen cuatro tipos de *mutex* acción-acción: tipo 1 (comienzo-comienzo) y tipo 2 (final-final); estos *mutex* representan que dos acciones no pueden comenzar o terminar, respectivamente, al mismo tiempo; tipo 3 (final-comienzo) que representa que una acción no puede terminar y otra comenzar en el mismo instante de tiempo; y tipo 4 (durante-durante) que representa que una acción no puede comenzar o terminar mientras se está ejecutando la otra.

- TPSYS realiza un preproceso de cálculo de *mutex* antes de la generación del grafo de planificación temporal y calcula *mutex* eternos, aquellos que sólo dependen de la definición de las acciones.

- En la generación del grafo de planificación temporal se tiene en cuenta distintos elementos. En primer lugar, se propagan los distintos tipos de *mutex* comentados anteriormente. En segundo lugar, para que una acción a comience en un nivel t se debe comprobar que $SCond(a) \subseteq S_t$ y $Inv(a) \subseteq S_t$, y lo mismo ocurre con el criterio de terminación, ya que debemos garantizar que para cualquier acción a_i de un plan solución se cumple que $ECond(a_i) \subseteq S_{t+dur(a_i)}$. En tercer lugar, los efectos de una acción a se deben añadir en sus niveles correspondientes: $SEff(a)$ se añade en el nivel $t + \epsilon$ y $EEff(a)$ se añade en el $t + dur(a)$.

- TPSYS también realiza cambios en la etapa de búsqueda porque ahora una acción a puede conseguir sus efectos al comienzo o final de la acción ($SEff(a)$ y $EEff(a)$). Además, para incluir una acción en el plan se deben satisfacer sus condiciones $SCond(a)$, $Inv(a)$ y $ECond(a)$ lo que implica realizar comprobaciones hacia adelante y hacia atrás en el grafo de planificación temporal.

2.3.3.2. Planificación heurística

Las diferentes funciones heurísticas que se han presentado en la Sección 2.2.3 se extendieron rápidamente a la planificación temporal, dando lugar a interesantes

resultados [22, 54]. Las heurísticas basadas en estados se adaptan a la definición de estado temporal (Definición 2.17) para estimar la distancia temporal (*makespan*) desde un estado a otro en el que se satisfacen todos los objetivos del problema. Las aproximaciones que utilizan este tipo de heurísticas construyen un RPG temporal donde se ignoran los efectos borrados de las acciones durativas. Un planificador representativo de esta aproximación es TP4 [52, 54]. TP4 es un planificador temporal óptimo que utiliza una heurística admisible $h^*(S)$ que se define como el último instante en el cual un plan puede conseguir el estado S. Para calcular $h^*(S)$, TP4 utiliza un grafo de planificación temporal en el que aplica una relajación equivalente al RPG pero teniendo en cuenta que ahora las acciones son sobre un modelo no-conservativo Por lo tanto, la idea subyacente es que las estimaciones calculadas se utilizan para evaluar los estados y acelerar la búsqueda regresiva.

2.3.3.3. Planificación temporal en aproximaciones POCL

Las características de las aproximaciones POCL hacen de este tipo de aproximación una forma natural y adecuada de incluir y manejar el tiempo en planificación. El enfoque POCL ofrece varias ventajas a la hora de incorporar el tiempo: i) el paradigma POCL es equivalente cuando se aplica a acciones instantáneas o acciones con la misma duración; ii) la definición de orden parcial permite manejar acciones de distinta duración siempre y cuando las condiciones que determinan las interferencias entre las acciones estén bien definidas [107]; y iii) la gran ventaja de la aproximación POCL es que ofrece un alto grado de flexibilidad gracias a la aplicación del principio de menor compromiso.

La mayoría de los planificadores temporales POCL utilizan intervalos temporales para representar las acciones y los literales. La idea de utilizar una representación del tiempo mediante intervalos la introdujo y popularizó Allen [1]. Las restricciones entre intervalos describen las relaciones entre acciones y literales (relaciones de precondición y efecto) o entre acciones (relaciones de causalidad o de orden). Los intervalos se definen generalmente como una tupla de puntos de tiempo, correspondiente al inicio y fin del intervalo, y las restricciones temporales se traducen en simples restricciones de igualdad y desigualdad entre los puntos extremos de los intervalos. HSTS [82], por ejemplo, utiliza un lenguaje de especificación de restricciones orientado a objetos basado en la lógica temporal de Allen y los dominios se

especifican utilizando una lógica temporal cualitativa con restricciones cuantitativas sobre la duración [90].

La representación mediante intervalos es muy flexible y ofrece una mayor expresividad que el modelo no conservativo de PDDL2.1 presentado en la Sección 2.3.1.1. Esta mayor expresividad fue explotada por ZENO [89], un planificador que aplica el principio de menor compromiso y que soporta una amplia variedad de métricas temporales, incluyendo restricciones temporales sobre objetivos y eventos exógenos. Existen otros planificadores temporales que utilizan lenguajes más expresivos. IxTeT [49], por ejemplo, permite acceder a los puntos de tiempo dentro del intervalo de una acción durativa e incluye otras funcionalidades a costa de una mayor complejidad. IxTeT busca un compromiso entre eficiencia y expresividad y permite manejar restricciones de concurrencia e interferencia entre acciones así como asociar eventos y objetivos a instantes de tiempo [50].

La dificultad de la aproximación POCL reside en que la detección y resolución de amenazas es más costosa porque la representación de las restricciones temporales es más expresiva y su resolución es más compleja. Se distinguen tres métodos generales para la resolución de restricciones temporales en una aproximación POCL:

1. Extender el algoritmo POCL clásico para resolver las restricciones entre acciones y proposiciones. Este método consiste en separar los aspectos temporales de los aspectos causales de las acciones de un plan, y utilizar una estructura de datos especializada para manejar cada uno de estos dos aspectos. Entre los planificadores que utilizan este enfoque podemos destacar FORBIN [19], O-PLAN2 [25] y TRIPTIC [99].

2. Cooperación entre un planificador POCL y un resolutor CSP, resolviendo las restricciones de cada nodo del espacio de planes mediante un CSP. En este tipo de enfoques, una red de restricciones temporales se representa como una *Simple Temporal Network (STN)* [20]. Ejemplos de planificadores que utilizan este enfoque son IxTeT [49] y VHPOP [117].

3. Convertir un problema de planificación temporal en un CSP y aplicar un resolutor de satisfacción de restricciones para obtener una solución. Por ejemplo, en el trabajo presentado en [38], todas las restricciones del problema temporal

(relaciones causales, ordenaciones, amenazas, relaciones temporales entre puntos de tiempo, etc.) se traducen a un CSP y se aplican técnicas de satisfacción de restricciones para su resolución.

2.3.3.4. Aproximaciones híbridas

Dada la complejidad de la planificación temporal y que ninguna de las aproximaciones presentadas ofrece un rendimiento superior al resto de enfoques, varias investigaciones estudiaron el desarrollo de planificadores híbridos que combinan varias estrategias. Por otra parte, cabe mencionar que la mayoría de las investigaciones en planificación temporal se han destinado al desarrollo de planificadores temporales subóptimos. En general, la gran mayoría de planificadores temporales actuales son subóptimos a excepción, por ejemplo, del planificador TP4 [52].

2.3.4. Planificadores temporales en la actualidad

En esta sección se describen los planificadores temporales actuales más relevantes, los cuales han destacado por su buen rendimiento en las competiciones internacionales de planificación. Concretamente, se presentan los siguientes planificadores: SGPlan [16], LPG [44] y sus posteriores versiones; OPTIC [5] y sus versiones predecesoras; TFD [31], basado en el planificador Fast Downward [55]; y YAHSP2 [109] y DAEYAHSP [69], ambos basados en la aproximación YAHSP [111], la cual fue la aproximación ganadora de la competición de Satisfacibilidad Temporal de la *IPC* 2011 e *IPC* 2014 [63].

2.3.4.1. SGPlan

SGPlan [16] es un planificador secuencial que está diseñado para resolver problemas de planificación clásica y temporal especificados en PDDL3.0, incluyendo objetivos no estrictos (*soft goals*), predicados derivados o características ADL. SGPlan resuelve un gran número de problemas de las competiciones de planificación y ofrece un excelente rendimiento pero el *makespan* de los planes que devuelve se aleja mucho de la calidad óptima. SGPlan fue el ganador de la sección de satisfacibilidad temporal en la *IPC* 2008.

SGPlan aplica un análisis basado en *landmarks* para descomponer el problema en subproblemas y resuelve cada uno de estos subproblemas con una versión modificada de Metric-FF [57]. En caso de no encontrar un plan solución dentro un límite temporal, la ejecución de Metric-FF se detiene e intenta realizar una descomposición adicional. Cuando se ha calculado un subplan para cada subobjetivo del problema, SGPlan fusiona dichos planes en un único plan.

El proceso de partición-resolución que utiliza SGPlan es muy útil en una amplia gama de dominios, aunque su rendimiento empeora en problemas en los que existen fuertes interacciones entre los subobjetivos. Esto es debido a que las acciones que consiguen los objetivos están muy relacionadas entre sí por lo que resulta más difícil obtener una partición significativa de las restricciones del problema.

2.3.4.2. YAHSP2

YAHSP [111] es un planificador heurístico subóptimo para dominios STRIPS que extiende el cálculo de la heurística del plan relajado de FF. Dado un estado S que se está evaluando, se añaden las acciones del plan relajado que son sucesivamente aplicables en S mientras que la aplicación de estas acciones permita alcanzar un plan válido. El estado resultante, S', de la aplicación de esta secuencia de acciones se denomina estado *lookahead*. YAHSP aplica un algoritmo primero el mejor en un espacio de búsqueda formado por los estados *lookahead*.

DAE-YASH [6] proporciona las características temporales a YAHSP, donde los literales se etiquetan con una estimación del instante de tiempo más temprano donde el literal puede aparecer y donde se utilizan las restricciones de exclusión mutua (*mutex*) entre estos literales. Básicamente, DAE-YASH aplica un proceso de razonamiento sobre las restricciones mediante un algoritmo genético, y luego comunica dichas restricciones a un planificador externo ya que no dispone de un planificador autocontenido, el cual genera un plan de orden parcial.

YAHSP2 [109] es una versión simplificada de YAHSP que incorpora las siguientes modificaciones:

- Los planes relajados para calcular las acciones que se incorporan a los estados *lookahead* no se calculan mediante grafos de planificación relajados sino directamente del calculo de heurísticas de camino crítico como h^{add}.

- El valor de la heurística para cada estado ya no es la longitud del plane relajado sino el valor h^{add}.

Este enfoque minimalista hace que YAHSP2 sea un planificador muy rápido y que resuelva muchos problemas. De hecho, una versión multi-core de este planificador, YAHSP2-MT [110], fue subcampeón *ex-aequo* en la sección de satisfacibilidad temporal de la *IPC* 2011. En la *IPC* 2014, los autores de YAHSP parciparon con YAHSP3 que es una evolución de YAHSP2 donde se fijan algunos errores que impedían a YAHSP2 encontrar planes válidos para dominios con acciones de coste 0 y también con YAHSP3-MT, la evolución de YAHSP2-MT. En la sección de satifacibilidad temporal de la *IPC* 2014, YAHSP3 fue el primer planificador respecto al número de problemas resueltos y YAHSP3-MT respecto a la calidad de los planes solución.

2.3.4.3. LPG-td

LPG-td [45] es una versión extendida del planificador LPG [44], la cual puede manejar muchas características de PDDL2.2. El esquema de búsqueda básica de LPG se inspira en Walksat [106], un eficiente algoritmo de búsqueda local para resolver problemas-SAT. El espacio de búsqueda de LPG-td se compone de grafos de acción temporal, en particular subgrafos del grafo de planificación, que representan planes temporales parciales. En un paso de búsqueda, LPG-td realiza modificaciones para refinar el grafo de acción. En LPG-td, además de incluir las acciones durativas y las expresiones numéricas de PDDL2.1, se amplía la representación de estos grafos de acción temporal para manejar las nuevas características de PDDL2.2. LPG-td incluye nuevas técnicas para problemas de planificación con *timed initial literals* y predicados derivados, y algunas mejoras generales respecto a LPG como:

- Se revisa y mejora el algoritmo para el cálculo de *mutex*.
- Se desarrolla una técnica para mejorar el grado de paralelismo en planes de dominios con acciones durativas o expresiones numéricas.
- Etc.

LPG-td es un planificador no determinista, debido a su enfoque de búsqueda local estocástica, pero es capaz de encontrar una primera solución muy rápidamente.

2.3.4.4. TFD

Temporal Fast Downward (TFD) [31] es un planificador de orden parcial de encadenamiento hacia delante que combina de forma integrada los aspectos de planificación y scheduling de la generación de un plan temporal. TFD es una variante del sistema de planificación *Fast Downward* [55] que aplica una búsqueda de tipo voraz en un espacio de estados temporales (estados del problema etiquetados en un punto de tiempo) y utiliza heurísticas basadas en estados e independientes del dominio que contemplan el tratamiento de acciones durativas y variables con duración.

TFD, al igual que *Fast Downward*, utiliza el concepto de operadores preferidos (aplicación del concepto de *helpful actions* de búsqueda local a una búsqueda global primero el mejor) y emplea una evaluación heurística aplazada (en lugar de aplicar la función heurística a los nodos generados en el momento de insertarlos en la frontera se aplica en el momento de la expansión). Este último aspecto resulta muy útil en problemas fuertemente guiados por la función heurística y con un alto factor de ramificación donde el número de nodos de la frontera es muy superior al número de nodos expandidos. Con todo, TFD consigue obtener planes de gran calidad a costa de una menor cobertura de problemas resueltos. En una posterior versión de TFD [30], los autores presentan nuevos métodos para refinar el concepto de operadores preferidos e incorporan una estructura multi-cola en el proceso de búsqueda. Con la introducción de estas nuevas técnicas, la cobertura de TFD mejora notablemente preservando la calidad de los planes. TFD fue el ganador de la competición de satisficibilidad temporal de la *IPC* 2011 [62] y consiguió resolver más problemas que el resto de planificadores participantes [30].

2.3.4.5. OPTIC

OPTIC [5] es un planificador de orden parcial que utiliza un proceso de búsqueda hacia adelante y que ha demostrado obtener planes de muy buena calidad (duración del plan). En la actualidad, OPTIC se puede considerar uno de los planificadores más relevantes en el estado del arte. La eficiencia de OPTIC se basa en una rápida generación de los nodos sucesores y en la utilización de una eficiente heurística independiente del dominio.

OPTIC aplica un proceso de búsqueda hacia delante en un espacio de planes de orden parcial. OPTIC hace uso del concepto de *estado frontera* que es el estado

resultante de la simulación de la ejecución del plan del nodo del árbol de búsqueda. El estado frontera se utiliza para dos tareas:

1. Para guiar el proceso de búsqueda y seleccionar el nodo a expandir. El estado frontera de cada nodo permite aplicar heurísticas basadas en estados que son más informativas que las heurísticas basadas en planes.

2. En el proceso de expansión de un nodo del árbol de búsqueda para generar nodos sucesores. OPTIC utiliza el estado frontera para determinar el conjunto de acciones aplicables y generar así los nodos sucesores.

En OPTIC, las acciones que se insertan en un nodo son únicamente aquellas cuyas precondiciones se satisfacen en el estado frontera. Este procedimiento ignora la posibilidad de insertar una acción cuyas precondiciones no se satisfacen en el estado frontera del nodo, aun cuando estas precondiciones se pudieran cumplir en algún punto intermedio del plan. De este modo, el proceso de generación de sucesores de OPTIC es un proceso incompleto y para garantizar la completitud de la búsqueda, OPTIC recurre a un mecanismo de backtracking.

OPTIC soporta una buena parte del nivel 5 de PDDL2.1, que incluye acciones con efectos numéricos continuos (lineales) y efectos dependientes de las duraciones de las acciones. También soporta restricciones que no son de obligado cumplimiento (*soft constraints*) y preferencias sobre los objetivos del problema.

A diferencia de otros planificadores, OPTIC no trabaja con dos procesos independientes de selección de acciones temporales y ordenación de dichas acciones (*scheduling*), obteniendo planes de alta calidad con respecto a la duración. OPTIC es una versión extendida de POPF2 [17], que obtuvo el segundo lugar ex-aequo en la sección satisfacibilidad temporal de la *IPC* 2011.

2.4. Restricciones temporales

En muchas aplicaciones del mundo real, obtener un plan de duración óptima no es un requerimiento imprescindible pero en cambio es importante satisfacer determinadas restricciones sobre los objetivos. Por ejemplo, en algunos procesos industriales o de logística es necesario disponer de los productos antes de un determinado instante de tiempo.

La definición de restricciones temporales sobre los objetivos del problema implica garantizar el cumplimiento de todas las restricciones del problema para que el plan sea válido. El lenguaje PDDL2.1 se extendió para incorporar nuevas funcionalidades, entre ellas la definición de restricciones temporales, dando lugar a PDDL2.2 [60] y, posteriormente, PDDL3.0 [43].

2.4.1. PDDL2.2

La versión PDDL2.2 [60] se utilizó en la *IPC* 2004 e incorpora dos nuevas funcionalidades: predicados derivados y *timed initial literals* (TILs). Los *timed initial literals* (TIL) son un tipo de construcción que permite modelar eventos exógenos, es decir, literales que serán verdaderos o falsos en un instante de tiempo conocido por el planificador, independientemente de las acciones que el planificador elija para la construcción del plan. Los *timed initial literals* se definen en la situación inicial y son literales que estarán disponibles a partir de un instante temporal mayor de 0, a diferencia de los literales del estado inicial que están disponibles en el instante 0.

Los *timed initial literals* son muy útiles para modelar situaciones del mundo real en forma de ventana temporal (el intervalo en el que una tienda está abierta, el intervalo en el que una persona está trabajando, el intervalo en el que el tráfico es lento, el intervalo en el que un satélite está disponible para realizar una comunicación, etc). Un *timed initial literal* nunca es generado o eliminado por una acción sino que se especifica explícitamente en el fichero del problema indicando el instante de tiempo de la generación o borrado del literal. Un TIL se define como una tupla (t, l), donde t es un valor real que denota un instante de tiempo a partir del cual el literal l es válido. Además, un TIL puede presentar la forma $(t, (not\ l))$ para indicar que a partir del instante t el literal l deja de ser válido.

En PDDL2.2 la información sobre los *timed initial literals* se declaran en la sección donde se define el estado inicial en el bloque del problema. Para ilustrar un ejemplo de definición de TILs, se utilizará el problema del dominio de transporte cuya situación inicial se representa gráficamente en la Figura 2.1.

La situación inicial es la misma que la definida en la Sección 2.3.2.1 y sólo se añade la información referida a los *timed initial literals*.

Este ejemplo representa una situación en la que la conexión entre las localizaciones loc1 y loc2 estará disponible en el instante 6 y dicho camino dejará de estar

```
(:init
      ...
      (at 6 (adjacent loc1 loc2))
      (at 22 (not (adjacent loc1 loc2)))
      (at 6 (adjacent loc2 loc1))
      (at 22 (not (adjacent loc2 loc1)))
)
```

Figura 2.12: Situación inicial con *timed initial literals* para el problema de la Figura 2.1

disponible en el instante 22.

2.4.2. PDDL3.0

En la *IPC* 2006 se introdujo la versión PDDL3.0 [43] que permite al usuario expresar restricciones de obligado cumplimiento y otras que son simplemente deseables (*soft constraints* o preferencias). Estas restricciones se pueden expresar sobre la estructura de un plan, también llamadas restricciones de trayectoria de estados. La motivación del lenguaje PDDL3.0 es proporcionar nuevas funcionalidades que permitan definir problemas más cercanos al mundo real. En muchos problemas reales se requiere poder especificar restricciones sobre el estado final del plan o sobre los estados intermedios que se generan durante la ejecución del plan, que no pueden expresarse fácilmente con las versiones anteriores de PDDL.

Las restriciones de trayectoria de estados son condiciones que deben cumplirse en toda la secuencia de estados visitados durante la ejecución de un plan. Se expresan mediante operadores modales temporales sobre una formulación de primer orden que afecta a los literales del estado. Las restriciones de trayectoria de estados se pueden clasificar en los siguientes grupos:

- Condiciones sobre los estados temporales. En este grupo podemos encontrar los operadores modales básicos always, sometime, at-most-once y at end.

- Condiciones sobre los límites temporales en el cumplimiento de los objetivos o *deadlines* que se definen mediante los operadores within, sometime-before, sometime-after y always-within.

- Condiciones sobre estados futuros que se definen mediante los operadores hold-during y hold-after.

Las restricciones de trayectoria se declaran en el bloque del problema en una nueva sección llamada :constraints. Estas restricciones también se pueden declarar en el bloque de dominio afectando así a todos los problemas del dominio; por ejemplo, la especificación de condiciones legales o de seguridad sobre algún proceso.

Para ilustrar cómo se definen estos elementos se utilizará el problema del dominio de transporte cuya situación inicial se representa gráficamente en la Figura 2.1.

Los objetivos del problema son los mismos que se han definido en la Sección 2.1.2 y sólo se añade la información referida a las restricciones de trayectoria.

```
(:constraints
      (within 10 (on c3 c2))
      (sometime-before (at r1 loc1) (on c3 c2))
)
```

Figura 2.13: Restricciones de trayectoria para el problema de la Figura 2.1

La Figura 2.13 indica las rectricciones que deben satisfacerse en un plan solución. La primera restricción indica que el objetivo del problema (on c3 c2) debe conseguirse antes del instante 10; es decir, este objetivo debe conseguirse antes de que hayan transcurrido 10 unidades de tiempo desde el inicio del plan. La segunda restricción especificada con el operador modal sometime-before exige que el robot se encuentre en loc1 antes de conseguir el objetivo (on c3 c2), independientemente de que el literal (at r1 loc1) no sea objetivo del problema.

Por otro lado, las preferencias son condiciones que al usuario le gustaría ver satisfechas aunque no son de obligado cumplimiento. Es decir, las preferencias no afectan a la validez del plan pero sí mejoran la calidad del mismo o los beneficios que el plan proporciona. Se especifican en la sección :constraints con la palabra preference.

2.5. Conclusiones

En este capítulo se han presentado las principales aproximaciones de la planificación clásica así como el modelo no conservativo de acciones durativas propuesto en

el lenguaje PDDL2.1 y un recorrido que analiza la evolución de la planificación temporal, haciendo especial hincapié en las aproximaciones temporales más relevantes del estado del arte. En los siguientes capítulos se verá que la aproximación temporal utilizada en este trabajo de tesis mantiene muchas de las características de los planificadores temporales *POCL* además de utilizar grafos de planificación y heurísticas independientes del dominio como la mayoría de planificadores temporales en la actualidad. Adicionalmente, nuestro modelo temporal incorpora las restricciones de trayectoria de estado de PDDL3.0 y a partir de éstas se construye la aportación de este trabajo de tesis, el modelo de landmarks temporales.

Capítulo 3

Grafos de Landmarks STRIPS

3.1. Introducción

Los *landmarks*, literales que deben ser ciertos en algún punto de todos los planes solución de un problema, se han utilizado ampliamente en la resolución de problemas de planificación en entornos STRIPS, demostrando su gran utilidad en el diseño de funciones heurísticas para guiar el proceso de búsqueda. El diseño de técnicas basadas en *landmarks* se fundamenta en las investigaciones realizadas sobre el estudio de órdenes entre objetivos del problema, extendiendo dicho estudio a los subobjetivos del mismo.

Desde la aproximación inicial de *landmarks* presentada por Hoffmann *et al.* [61], han surgido otras aproximaciones capaces de encontrar un mayor número de *landmarks* y órdenes, las cuales se han desarrollado independientemente unas de otras y han obtenido ciertas mejoras respecto a los resultados de la aproximación inicial [61]. Estas aproximaciones proponen una visión diferente y complementaria de la extracción de *landmarks* presentada en el trabajo Hoffmann *et al.* [61], de modo que el estudio de las fortalezas y debilidades de cada aproximación, junto con una adecuada combinación de las mismas, da origen a una nueva técnica que mejora los resultados obtenidos con cada aproximación individual, y que constituye el objetivo de este capítulo [75].

La técnica de extracción de landmarks STRIPS que se presenta en este capítulo extrae un conjunto de *landmarks* individuales, un conjunto de *landmarks* disyuntivos y un conjunto de órdenes entre ellos. Los *landmarks* (individuales o disyuntivos) extraídos, junto con sus órdenes, forman un grafo que se denomina *grafo de*

landmarks. Este grafo se utilizará posteriormente en el Capítulo 4 para obtener un conjunto *landmarks* temporales y formar un nuevo grafo de *landmarks* temporales.

En la siguiente sección se presenta un ejemplo de aplicación que servirá para mostrar los conceptos de landmarks más importantes y que posteriormente se formalizarán en la Sección 3.3. En la Sección 3.4 se presentan las aproximaciones de extracción de landmarks más relevantes y destacadas así como una comparativa de los resultados obtenidos con estas aproximaciones. Por último, se explica la técnica de extracción de *landmarks* STRIPS desarrollada en este trabajo, la cual reutiliza y combina las técnicas de las aproximaciones presentadas en la Sección 3.4 y donde pueden verse las mejoras obtenidas (Sección 3.5) [75].

3.2. Ejemplo de aplicación

Para ilustrar los distintos elementos que se van a definir en este capítulo, se presenta un ejemplo de un problema de transporte del dominio *depots* donde los camiones pueden moverse entre depósitos o distribuidores y las grúas pueden cargar y descargar cajas que pueden encontrarse en un camión, en un pallet o sobre otra caja. El entorno que representa la situación inicial para este ejemplo se muestra en la Figura 3.1. Se puede observar que disponemos de un depósito (D0) donde se encuentra la grúa (H0), el pallet (P0) y el camión (T1). Adicionalmente, sobre el pallet (P0) se encuentra la caja (C1). Se dispone de un distribuidor (D1) donde se encuentra la grúa (H1), el pallet (P1), la caja (C0) y el camión (T0). Por último, se dispone de otro distribuidor (D2) donde se encuentra la grúa (H2) y el pallet (P2). Los objetivos de este problema son tener la caja C0 sobre el pallet P2 y tener la caja C1 sobre el pallet P1.

Los predicados que se utilizan para definir un estado cualquiera de este dominio se pueden ver en la Figura 2.3 del Capítulo 2.La especificación completa en PDDL del dominio del *depots* se encuentra en el Apéndice A.

3.3. Conceptos de landmarks

Un *landmark* se define como un literal que debe ser cierto en algún momento durante la ejecución de cualquier plan de solución [93].Formalmente [61]:

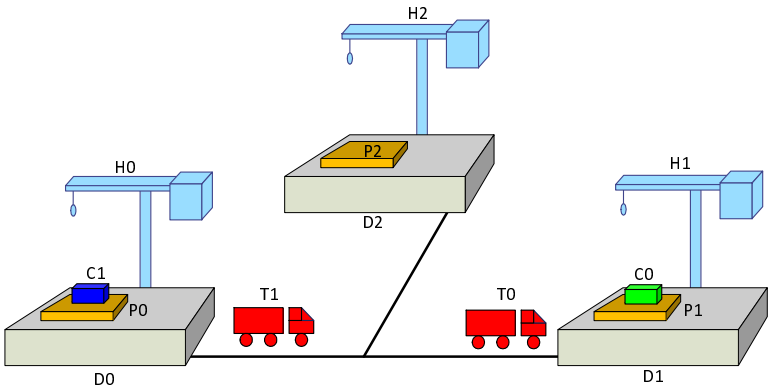

Figura 3.1: Ejemplo del dominio *depots*

Definición 3.1. *Sea un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$*. Un literal* l *es un* **landmark** *si para todos los planes* $\pi = \langle a_1, \ldots, a_n \rangle, \mathcal{G} \subseteq S_n$*, donde* $S_n = Result(\mathcal{I}, \pi)$*:* $l \in Result(\mathcal{I}, \langle a_1, \ldots, a_i \rangle)$ *para cualquier* $0 \leq i \leq n$*.* ∎

Siguiendo el ejemplo de la Sección 3.2, obtendríamos el *landmark* "la grúa H1 debe levantar la caja C1", representado por el literal (lifting H1 C1), ya que este literal deberá conseguirse en cualquier plan solución para conseguir el objetivo "tener la caja C1 sobre el pallet P1" (on C1 P1).

Los literales de la situación inicial y los objetivos son, trivialmente, *landmarks*. En general, decidir si un literal es *landmark* es PSPACE-completo. Por esta razón, se define una condición suficiente para que un literal sea *landmark* [61]:

Proposición 3.1. *Sea un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$*, un literal* l *y sea* $\mathcal{R} = (\mathcal{A}_\mathcal{R}, \mathcal{I}, \mathcal{G})$ *el problema de planificación relajado de* $\mathcal{P}$*. Se define* $\mathcal{R}(\neg l) = (\mathcal{A}_\mathcal{R}(\neg l), \mathcal{I}, \mathcal{G})$*, donde:* $\mathcal{A}_\mathcal{R}(\neg l) = \{(pre(a), add(a), \emptyset) | (pre(a), add(a), del(a)) \in \mathcal{A}, l \notin add(a)\}$*. Si* $\mathcal{R}(\neg l)$ *es irresoluble, entonces* l *es un* landmark *en* $\mathcal{P}$*.* ∎

Denominaremos $RPG(\neg l)$ al grafo de planificación relajado que se obtiene habiendo eliminado las acciones que tienen l como efecto positivo, es decir, $RPG(\neg l)$ será el grafo de planificación relajado que se obtiene sobre $\mathcal{R}(\neg l)$. Para el ejemplo de la Sección 3.2, si calculamos los *landmarks* mediante un *RPG* hasta el nivel de literal donde se consiguen los objetivos, podríamos obtener el literal (in C1 T1) que indica que "el camión T1 debe cargar la caja C1". Sin embargo, si aplicamos la Proposición 3.1 a este literal podemos comprobar que no es un *landmark* ya que, aunque eliminemos todas las acciones que consiguen (in C1 T1), se puede generar un

nuevo $RPG(\neg(\mathsf{on\ C1\ P1}))$ en el que se cumplan todos los objetivos. Esto es debido a que el objetivo (on C1 P1) se puede conseguir cargando la caja C1 en el camión T0 en lugar de en el camión T1.

3.3.1. Relaciones de orden entre *landmarks*

Para utilizar los *landmarks* durante la búsqueda, no sólo es importante extraer la mayor cantidad posible de *landmarks*, sino también poder obtener un esqueleto del plan solución. La aproximación inicial de *landmarks* descrita en [61], además de introducir un procedimiento para la extracción de *landmarks*, también define los diferentes tipos de relaciones de órdenes entre *landmarks*. Estas relaciones también han sido adoptadas por las aproximaciones posteriores de extracción y gestión de landmarks.

Definición 3.2. *Sea un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$*, y dos* landmarks l *y* l'*. Se dice que hay un relación de* ***orden de dependencia***[1] *entre* l *y* l'*, y se denota como* $l \prec_d l'$*, si* $l' \notin \mathcal{I}$*, y para todos* $\pi = \langle a_1, \ldots, a_n \rangle \in \mathcal{A}^*$*: si* $l' \in Result(\mathcal{I}, \langle a_1, \ldots, a_i \rangle)$ *entonces* $l \in Result(\mathcal{I}, \langle a_1, \ldots, a_j \rangle)$ *tal que* $j < i$*.* ∎

Existe un orden dependiente entre l y l', $l \prec_d l'$, si todos los planes añaden l antes de añadir l'. Del ejemplo de la Sección 3.2 podemos extraer un orden de dependencia entre el literal (clear C1) y el literal objetivo (on C1 P1), lo que indica que antes de poder tener la caja C1 sobre el pallet P1, la caja no debe tener ninguna otra caja encima. Si esta relación de orden sólo se define cuando $i = j - 1$ entonces se convierte en una relación de orden necesario. Una relación de orden *necesario* se define entre un par de *landmarks* l y l', si, en cualquier secuencia de acciones que hacen que l' sea cierto en algún estado, l debe ser cierto en el estado inmediatamente anterior [61]. Una relación de orden *necesario* se denota como $l \prec_n l'$. Formalmente:

Definición 3.3. *Sea un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$*, y dos* landmarks l *y* l'*. Se dice que hay una relación de* ***orden necesario*** *entre* l *y* l'*, y se denota como* $l \prec_n l'$*, si* $l' \notin \mathcal{I}$*, y para todos* $\pi = \langle a_1, \ldots, a_n \rangle \in \mathcal{A}^*$*: si* $l' \in Result(\mathcal{I}, \langle a_1, \ldots, a_i \rangle)$ *entonces* $l \in Result(\mathcal{I}, \langle a_1, \ldots, a_{i-1} \rangle)$*.* ∎

Hay que tener en cuenta que si $l' \in Result(\mathcal{I}, \langle a_1, \ldots, a_i \rangle)$ tal que $i \geq 1$ entonces $l' \notin \mathcal{I}$. Como el propósito de una relación de orden necesario $l \prec_n l'$ es determinar

[1] *Esta relación de orden también se llama* orden natural *en el trabajo de Porteous* et al. *[93].*

que l debe ser cierto en el estado inmediatamente anterior al estado donde l' es cierto, no tiene sentido permitir una relación de orden necesario entre literales de la situación inicial.

Decidir si existe una relación de orden necesario entre dos literales es también un problema PSPACE-completo. Además, las condiciones para establecer una relación de orden necesario son tan restrictivas que muy pocos *landmarks* podrían ordenarse de acuerdo a esta definición. Por este motivo se define otra relación de orden, los *órdenes necesarios voraces*, una versión menos estricta que los órdenes necesarios [61]. Los autores de este trabajo postulan que no es necesario que l sea cierto inmediatamente antes de l' en todas las secuencias de acción, sino sólo en las secuencias de acción donde l' *se consigue por primera vez* desde la situación inicial.

Definición 3.4. *Sea un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$*, y dos* landmarks l *y* l'*. Se dice que hay una relación de* ***orden necesario voraz*** *entre* l *y* l'*, y se denota como* $l \prec_{gn} l'$*, si* $l' \notin \mathcal{I}$*, y para todos* $\pi = \langle a_1, \ldots, a_n \rangle \in \mathcal{A}^*$*: si* $l' \in Result(\mathcal{I}, \langle a_1, \ldots, a_i \rangle)$ *y* $l' \notin Result(\mathcal{I}, \langle a_1, \ldots, a_j \rangle)$ *para* $1 \leq j < i$*, entonces* $l \in Result(\mathcal{I}, \langle a_1, \ldots, a_{i-1} \rangle)$*.* ∎

Una relación de orden necesario voraz indica que es más relevante la información que se extrae cuando un literal se consigue por primera vez directamente desde el estado inicial, frente a cuando se consigue desde cualquier otro estado del problema. Cuando se consideran todos los estados del problema, algo intrínseco en la definición de las relaciones de orden necesario (no voraz), se puede perder información útil.

De acuerdo con la Definición 3.4 y siguiendo con estado el inicial representado en la Figura 3.1, suponiendo que sólo disponemos del camión T1, podemos extraer un **orden necesario voraz** entre los literales (lifting H0 C1) y (in C1 T1) ya que, desde el estado inicial, para poder tener la caja C1 en el camión T1 - (in C1 T1) - es necesario que la grúa H0 haya levantado la caja C1 - (lifting H0 C1) - en el estado inmediatamente anterior. Nótese que una relación de orden necesario (no voraz) entre (lifting H0 C1) y (in C1 T1) estaría indicando que en cualquier estado del problema en el que la caja C1 esté dentro del camión T1 tiene que cumplirse (lifting H0 C1) en el estado inmediatamente anterior. Como se puede observar por la descripción del propio problema, esto no tiene que ser necesariamente así ya que existen otras grúas que se podrían utilizar para recoger la caja antes de introducirla en el camión. En cambio, en el estado inicial del problema, la única grúa que puede

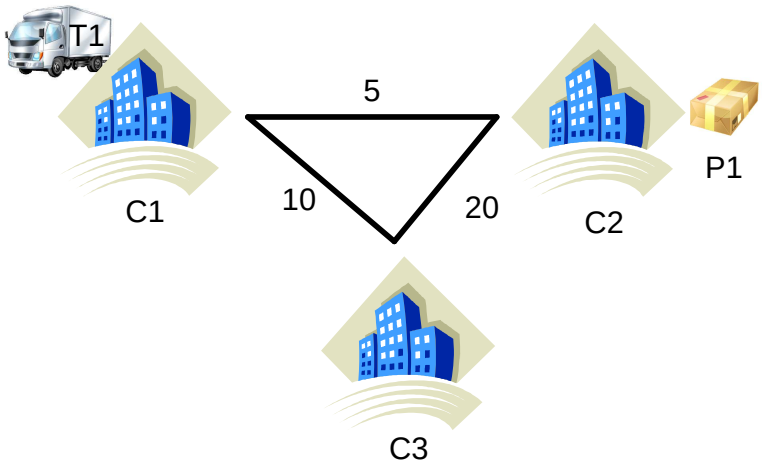

Figura 3.2: Ejemplo de órdenes razonables

recoger C1 en el estado inmediatamente anterior a introducirla en el camión T1 es la grúa H0.

Al igual que ocurre con las relaciones de orden necesario, en una relación de orden necesario voraz la secuencia de acciones para conseguir l debe contener al menos un paso, por lo que $l' \notin \mathcal{I}$.

En esta tesis, nos referiremos a los órdenes necesarios voraces como órdenes necesarios y se denotarán como $l \prec_n l'$.

Por otro lado, existe también otro tipo de relaciones de orden llamados *órdenes razonables* [61] que no se considerarán en este trabajo ya que nuestro principal objetivo es transformar un grafo de *landmarks* STRIPS en un grafo de *landmarks* temporal y los órdenes razonables establecen un tipo de relación que puede no ser obligatorio desde el punto de vista temporal. Las relaciones de orden razonable se utilizan para minimizar el número de acciones mientras que en un contexto temporal el objetivo es minimizar el tiempo total del plan. Por ejemplo, de acuerdo con la Figura 3.2, imaginemos que el objetivo es transportar el paquete P1 de la ciudad C2 a la ciudad C3 utilizando el camión T1. El camión T1 debe ir a la ciudad C2 para cargar y a la ciudad C3 para descargar. Para hacer esta ruta, T1 parte de C1 hacia C2 y después de cargar puede ir directamente a C3 o bien volviendo por C1. En esta situación parece más *razonable* ir directamente a C3 por lo que se introduciría un *orden razonable* tratando de minimizar el número de acciones. Supongamos ahora

que los números que aparecen en los trazos de la Figura 3.2 representan el tiempo que le cuesta al camión T1 desplazarse entre las ciudades. En este caso, resulta más conveniente que, una vez cargado el paquete P1, el camión se desplace a la ciudad C3 a través de C1, es decir, ya no sería razonable ir directamente a C3.

3.3.2. Grafo de landmaks

El conjunto de *landmarks* que se extraen de un problema junto con las relaciones de orden entre dichos *landmarks* definen un grafo que se denomina **grafo de *landmarks***, donde:

- cada *landmark* se representa como un nodo del grafo y
- cada relación de orden se representa como un arco entre dos nodos.

Definición 3.5. *Sea un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$*. Se define* ***grafo de* landmarks *(LG)*** *como* $LG = (N, E)$ *donde* N *es el conjunto de* landmarks *de* $\mathcal{P}$ *y* E *es el conjunto de relaciones de orden entre estos* landmarks*, es decir,* $l_i \prec_n l_j$ *o* $l_i \prec_d l_j$*.* ∎

Un LG define el esqueleto de cualquier plan solución y se puede utilizar para distintos propósitos como veremos a lo largo de este trabajo.

3.4. Métodos para la generación del grafo de *landmarks*

En la aproximación inicial de *landmarks* los autores proponen un algoritmo para la extracción de *landmarks* y sus órdenes a partir del grafo de planificación relajado (RPG) de un problema de planificación [93, 61]. Denominaremos LM a esta aproximación en este trabajo de tesis.

Mientras que la aproximación LM establece las bases para la extracción de *landmarks* de un problema de planificación, existen otros trabajos que realizan nuevas aportaciones a este análisis, como por ejemplo una identificación más exhaustiva de los conjuntos de *landmarks disyuntivos* [92]. El método para calcular *landmarks disyuntivos* presentado por Porteous y Cresswell [92] (denominaremos DL a esta aproximación en este trabajo de tesis) es una aproximación diferente y de carácter más general que la utilizada en LM.

El trabajo de Richter *et al.* [95] adopta las aproximaciones LM y DL para extraer *landmarks* y órdenes e introduce un conjunto adicional de *landmarks* extraídos a partir de grafos de transición del dominio (DTG). Este nuevo enfoque es capaz de encontrar más *landmarks* y órdenes que la aproximación LM y proporciona una mayor garantía de correctitud de los órdenes entre *landmarks* (denominaremos DTG a esta aproximación en este trabajo de tesis). Por otro lado, la técnica desarrollada por Zhu y Givan [118], que utiliza un grafo de planificación (PG) en lugar de un RPG para extraer *landmarks*, distingue dos tipos de *landmarks*: *landmarks* proposición y *landmarks* acción. Esta aproximación (que denominaremos Pro en el presente trabajo) realiza una eficaz propagación de la información de los *landmarks* sobre el PG y, según afirman los autores, es capaz de calcular un número de *landmarks* significativamente mayor que la aproximación LM.

Partiendo de estas aproximaciones se mostrará cómo se pueden integrar las diferentes técnicas y diseñar así una nueva aproximación que permite extraer un mayor número de *landmarks* que cada técnica por separado. Esta aproximación (que denominaremos FULL en este trabajo de tesis) no sólo consiste en la suma de las diferentes aproximaciones mencionadas anteriormente sino en un cuidadoso análisis de las propiedades de cada técnica para determinar cuándo y cómo introducir cada una de ellas.

3.4.1. La aproximación LM

Calcular el conjunto de *landmarks* de un problema de planificación se puede hacer de forma sencilla mediante la evaluación de la Proposición 3.1 para cada literal del problema. Sin embargo, este proceso puede resultar muy costoso cuando el problema contiene un gran número de literales. Por esta razón, en el trabajo de Hoffmann *et al.* [61] se propone un método que consta de dos pasos: extraer un conjunto de *landmarks* candidatos y, posteriormente, realizar una verificación de los mismos. Antes de detallar el funcionamiento de este método, se introduce la definición de los *first archievers* de un *landmark*. Utilizaremos la función $fl(l)$ para obtener el primer nivel del RPG donde se consigue el literal l. Al conjunto de literales comprendidos en un nivel i del RPG lo llamaremos $RPGL_i$. Del mismo modo, al conjunto de acciones comprendidas en un nivel de acción i del RPG lo denominaremos $RPGA_i$.

Definición 3.6. *Sea l un literal que aparece por primera vez en $RPGL_i (fl(l) = i)$. Se denomina* ***first achievers*** *de l al conjunto $RPGA_i$ formado por las acciones que tienen l como efecto positivo y cuyas precondiciones pertenecen a $RPGL_{i-1}$. Usaremos la función $fa_{LM}(l)$ para referirnos a los first achievers del landmark l: $fa_{LM}(l) = \{a \in RPGA_i : l \in add(a) \wedge pre(a) \subseteq RPGL_{i-1}\}$.* ∎

El **primer paso** de la aproximación LM consiste en calcular un conjunto de *landmarks* candidatos LC. Para ello se calcula un RPG hasta que se consiguen todos los objetivos del problema. Si se construye el grafo de planificación relajado de un problema hasta el nivel en el que no se pueden añadir nuevas acciones ni nuevos literales y en el último nivel del RPG no están presentes todos los objetivos, entonces el problema de planificación es irresoluble [7]. El Algoritmo 3.1 muestra el procedimiento de extracción de landmarks candidatos de LM. La línea 5 del algoritmo calcula el RPG del problema hasta que se alcanza un nivel donde se encuentran todos los objetivos del mismo (utilizamos el índice g para denotar el primer nivel del RPG donde aparecen todos los objetivos del problema). A continuación se inicializa el vector $Goals(i)$ con los objetivos ($g \in G$) que se alcanzan por primera vez en el nivel i (líneas 6 al 8). A partir de este punto se recorren todos los niveles del RPG en orden descendiente y para cada literal g en $Goals(level)$ se obtienen los literales resultantes de la intersección de las precondiciones de las acciones en $fa_{LM}(g)$ (línea 2 del Algoritmo 3.2), añadiendo dichos literales al conjunto LC como puede verse en el línea 4 del Algoritmo 3.2. En el línea 5 de este mismo algoritmo cada nuevo *landmark* candidato se añade al vector $Goals(i)$ en el primer nivel i donde se ha alcanzado dicho *landmark*.

Sin embargo, este proceso tiene un inconveniente. Supongamos un *landmark* candidato l que puede alcanzarse mediante un conjunto de acciones $\{a_1, \ldots, a_n\}$. Si la intersección de las precondiciones de estas acciones está vacía, el proceso descrito finalizaría. Por esta razón, se considera un nuevo tipo de *landmark*, denominado *landmark disyuntivo*, que está formado por un conjunto de literales de los cuales uno de ellos debe ser cierto en algún momento en todo plan de solución.

Definición 3.7. *Dado un conjunto de literales $l_1 = (pred_1\ a_{11}\ ...\ a_{1m_1})$, ..., y $l_n = (pred_n\ a_{n1}\ ...\ a_{nm_p})$ se dice que son del* ***mismo tipo*** *y se denota como $sameType(l_1, ..., l_n)$ si:*

- $pred_i = pred_j \forall i, j : 1 \leq i, j \leq n$

Algoritmo 3.1 Aproximación LM

Input: $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$
$N = \mathcal{G} \cup \mathcal{I}$;
$E = \emptyset$
{**Extraer *landmarks* candidatos**}
Calcular RPG **hasta alcanzar** $RPGL_g$
for all $g \in \mathcal{G}$ **do**
  $Goals(fl(g)) = Goals(fl(g)) \cup g$
end for
$LC = \emptyset$
$DsjGoals = \emptyset$
for all $level$ **de** RPG **(en orden descendiente) do**
  for all $g \in Goals(level)$ **do**
    $intersection(fa_{LM}(g), g, LC, Goals, DsjGoals, level, E)$
  end for
  for all $s \in DsjGoals(level)$ **do**
    $N = N \cup \{s\}$
    $FA_s = \bigcup_{\forall l \in s} fa_{LM}(l)$
    $intersection(FA_s, s, LC, Goals, DsjGoals, level, E)$
  end for
end for
{Verificar *landmarks* candidatos}
for all $l \in LC$ **do**
  Calcular $RPG(\neg l)$ **hasta** $RPGL_f$
  if $\mathcal{G} \nsubseteq RPGL_f$ **then**
    $N = N \cup \{l\}$
  end if
end for
return $LG = (N, E)$

- $m_i = m_j \forall i, j \in [1, p]$
- $\forall a_{1i}, a_{2i}, \ldots, a_{ni}$ $1 \leq i \leq m_1$ *son objetos cuyo tipo es el mismo.* ∎

Por ejemplo, en el problema de la Sección 3.2, los literales (at T0 D0), (at T1 D1 y (at T0 D2) son del mismo tipo:

- el predicado es el mismo en todos los literales: at
- la aridad de los literales es la misma

Algoritmo 3.2 Función *intersection*

1: **Input:** $actions$, g, LC, $Goals$, $DsjGoals$, $level$, E
2: $intersec = \bigcap_{\forall a \in actions} pre(a)$
3: **for all** $l \in intersec$ **do**
4: $\quad LC = LC \cup \{l\}$
5: $\quad Goals(fl(l)) = Goals(fl(l)) \cup l$
6: $\quad E = E \cup \{l \prec_n g\}$
7: **end for**
8: $union = \bigcup_{\forall a \in actions} pre(a) - intersec$
9: **for all** $sameType(l_1, ..., l_n) \in union$ **do**
10: $\quad DsjGoals(level - 1) = DsjGoals(level - 1) \cup \{sameType(l_1, ..., l_n)\}$
11: $\quad E = E \cup \{sameType(l_1, ..., l_n) \prec_n g\}$
12: **end for**

- los objetos T0 y T1 son del mismo tipo truck y los objetos D0, D1 y D2 son del tipo distributor

En cambio, los literales (at T0 D0) y (in C0 P1) no son del mismo tipo ya que los predicados no son iguales y los objetos T0 y C0 son de tipo truck y crate, respectivamente. Los distintos tipos definidos para el dominio *depots* del problema de la Sección 3.2 pueden verse en la Figura 2.2 y A.3 es un apéndice que muestra el problema de aplicación de la sección 3.2, donde puede verse el tipo de cada uno de los objetos definidos en este problema.

Definición 3.8. *Dado un problema de planificación* $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ *y un conjunto de literales* $ls = \{l_1, \ldots, l_n\}$ *donde todos los literales son del mismo tipo (sameType* $(l_1, \ldots, l_n)$*) diremos que este conjunto de literales es un* **landmark *disyuntivo*** *si* $\forall \pi = \langle a_1, \ldots, a_n \rangle \in \mathcal{A}^*, \mathcal{G} \subseteq S_n$*, donde* $S_n = Result(\mathcal{I}, \pi)$*:* $\exists l_i : l_i \in ls \wedge l_i \in Result(\mathcal{I}, \langle a_1, \ldots, a_j \rangle)$ *para cualquier* $1 \leq j \leq n$*.* ∎

Del ejemplo de la Sección 3.2 se obtiene un conjunto de *landmarks disyuntivos* formado por los literales {(in C1 T0), (in C1 T1)} ya que se dispone de dos camiones y a priori se desconoce el camión en el que se cargará la caja C1.

Al proceso de extracción de *landmarks* candidatos del Algoritmo 3.1 se incorpora el calculo de los *landmarks disyuntivos*. Así, igual que se dispone del vector $Goals(i)$ para almacenar los *landmarks*, tenemos el vector $DsjGoals(i)$ que almacena los *landmarks disyuntivos* que se han conseguido en el nivel i. En la función *intersection* (Algoritmo 3.2) se obtiene la unión de las precondiciones de las acciones

que pertenecen a $fa_{LM}(g)$ y se eliminan los literales que son *landmarks* candidatos (línea 8). Cada conjunto de literales que son del mismo tipo en *union* formará un *landmark disyuntivo* que se añadirá al vector $DsjGoals$ en el nivel anterior al del literal g que se está analizando; esto es, en el nivel $level - 1$. En el Algoritmo 3.1 (líneas 15 a 19) se aplica el proceso de extracción de *landmarks* candidatos a los *landmarks disyuntivos*, de forma que si se ha creado un *landmark disyuntivo* $ls = \{l_1, \ldots, l_n\}$ para conseguir el *landmark* candidato l y todas las acciones que consiguen alguno de los literales $\{l_1, \ldots, l_n\}$ comparten alguna precondición l', se puede deducir que l' es un *landmark* candidato que tiene que ser cierto dos pasos anterior a l; es decir, l' será anterior a ls y ls será anterior a l.

Después de la extracción de los *landmarks* candidatos y *landmarks* disyuntivos, el **segundo paso** del Algoritmo 3.1 es verificar los literales en LC para descartar los que no son *landmarks* (líneas del 22 al 27 del Algoritmo 3.1). Esta evaluación se realiza de acuerdo a la Proposición 3.1 y se calcula $RPG(\neg l)$ hasta el nivel en el que no se pueden añadir más acciones y ni más literales. Este nivel se denomina punto fijo y el último nivel de literal del grafo será $RPGL_f(\neg l)$, donde f indica el nivel del punto fijo. Si todos los objetivos se encuentran en el nivel $RPGL_f(\neg l)$, entonces l no es un *landmark*.

En la aproximación LM la extracción de los órdenes necesarios se realiza al mismo tiempo que la extracción de *landmarks* candidatos (línea 6 y 11 del Algoritmo 3.2). Cuando un literal l es un *landmark* candidato porque pertenece a la intersección de las precondiciones de las acciones que consiguen otro *landmark* l', se añade un orden necesario entre l y l' ($l \prec_n l'$). Cuando existe un *landmark disyuntivo* ls con la unión de los literales que son del mismo tipo de las acciones que consiguen otro *landmark* l', se añade un orden necesario entre ls y l' ($ls \prec_n l'$).

Para el ejemplo de la Sección 3.2 la aproximación LM devuelve el grafo de landmarks de la Figura 3.3, donde se han omitido los literales de la situación inicial que no tienen ningún enlace en el grafo. En esta figura puede verse que para el objetivo (on C0 P2) se necesita (lifting H2 C0) ya que es la única posibilidad de conseguir (on C0 P2). En cambio para el *landmark* (lifting H2 C0) se tiene el *landmark disyuntivo* {(in C0 T0), (in C0 T1)} ya que se desconoce el camión que transportará la caja C0 hasta el distribuidor D2.

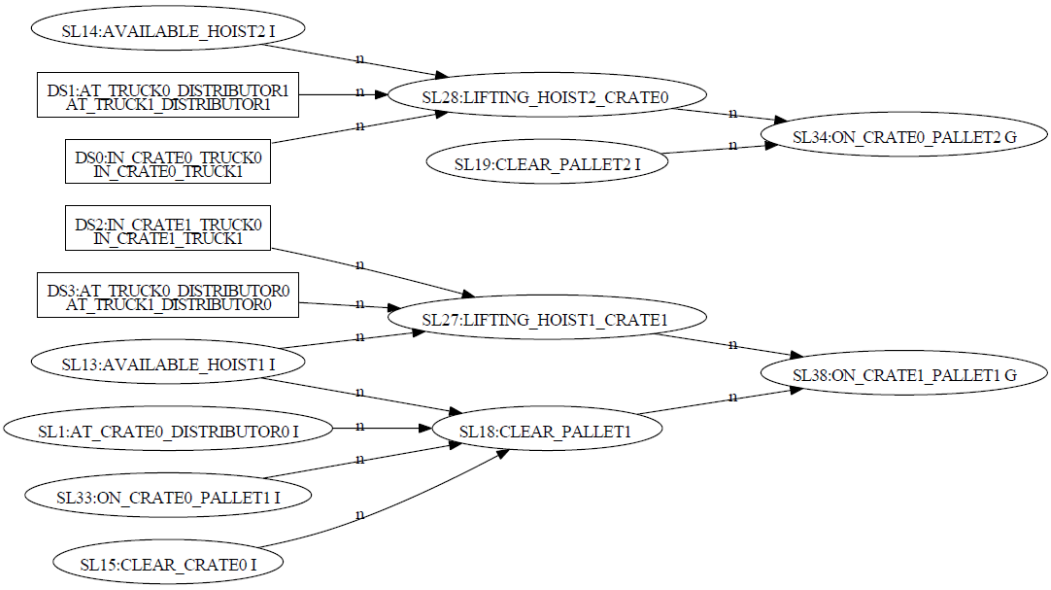

Figura 3.3: Grafo de *landmarks* del dominio *depots* con la aproximación *LM*

3.4.2. La aproximación DL

El número de *landmarks* que se puede encontrar en un problema no sólo depende del algoritmo de extracción sino también del tipo de dominio de planificación del problema. En muchos problemas de planificación no es posible identificar *landmarks* cuando hay una elección arbitraria de los objetos que participan en las acciones. La aproximación DL [92] amplía el análisis de *landmarks* de la aproximación LM haciendo uso de recursos abstractos. En DL, cada *landmark* se trata como un conjunto disyuntivo, que puede ser un conjunto formado por un único *landmark* o un conjunto de literales que representa un *landmark* con $r$ecursos abstractos.

El proceso del cálculo de *landmarks* y órdenes de la aproximación DL es ligeramente distinto al que se realiza en la aproximación LM. Por una parte, dado un *landmark* l, se utiliza el concepto de *landmarks possibly before* a l para establecer órdenes dependientes entre *landmarks* y, por otra parte, utiliza una definición distinta del concepto de *first archievers*. Antes de detallar el funcionamiento de esta aproximación se introduce la definición de *possibly before* y de *first archievers* de esta aproximación.

Algoritmo 3.3 Aproximación DL

1: **Input:** $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$
2: $N = \mathcal{G} \cup \mathcal{I}$
3: $E = \emptyset$
4: {**Extracción de *landmarks* y conjuntos disyuntivos**}
5: $QL = \mathcal{G}$
6: **for all** $l \in QL$ **do**
7: $intersec = \bigcap_{\forall a \in fa_{DL}(l)} pre(a)$
8: $dsj = findDisjuntiveSets(l, intersec)$
9: $QL = QL \cup intersec \cup dsj$
10: $N = N \cup intersec \cup dsj$
11: **for all** $l' \in intersec \cup dsj$ **do**
12: $E = E \cup \{l' \prec_n l\}$
13: **end for**
14: **end for**
15: **{Extracción de nuevos ordenes}**
16: **for all** $l_1, l_2 \in N$ **do**
17: **if** $l_2 \notin pb(l_1)$ **then**
18: $E = E \cup \{l_1 \prec_d l_2\}$
19: **end if**
20: **end for**
21: **return** $LG = (N, E)$

Definición 3.9. *Un literal l se dice que es* ***possibly before*** *a un literal l' si l pertenece al nivel literal $RPGL_f(\neg l')$. Usaremos la función $pb(l')$ para referirnos a los* landmarks *possibly before del landmark l': $pb(l') = RPGL_f(\neg l')$.* ∎

Definición 3.10. *Se denomina* ***first achievers*** *de l al conjunto formado por todas las acciones del problema cuyas precondiciones se han alcanzado en $pb(l)$. Usaremos la función $fa_{DL}(l)$ para referirnos a los* first achievers *del* landmark *l: $fa_{DL}(l) = \{a \in A : l \in add(a) \wedge pre(a) \subseteq pb(l)\}$.* ∎

El Algoritmo 3.3 muestra el proceso de cálculo de la aproximación DL. En términos generales, este procedimiento puede resumirse de la siguiente forma: sea l un *landmark* tal que $fa_{DL}(l) = \{a_1, a_2, \ldots, a_n\}$. La intersección de las precondiciones de las acciones en $fa_{DL}(l)$ son *landmarks* (línea 7 del Algoritmo 3.3). El siguiente paso consiste en calcular los *landmarks disyuntivos* (Algoritmo 3.4) que crea un conjunto disyuntivo por cada predicado y lo almacena en DS. Para crear estos conjuntos disyuntivos se selecciona una acción cualquiera de $fa_{DL}(l)$, por ejemplo a_0 (línea 3 del Algoritmo 3.4), y para cada precondición de a_0 que no es *landmark* se

obtiene su predicado ($predicate(p)$), se crea el conjunto disyuntivo $ds_{predicate(p)}$ que se añade a DS y se inicializa $ds_{predicate(p)}$ con dichas precondiciones (líneas del 5 al 8). A continuación, se repite el mismo proceso para el resto de las acciones de $fa_{DL}(l)$ (líneas del 9 al 17 del Algoritmo 3.4). Finalmente, se devuelve DS con los *landmarks disyuntivos* que se hayan conseguido.

De acuerdo con el Algoritmo 3.3, para todos los *landmarks* l' que se han obtenido como resultado de las operaciones de intersección y cálculo de *landmarks disjuntivos* de un *landmark* l, l' se ordena necesariamente antes de l ($l' \prec_n l$) (líneas del 11 al 13). A continuación (líneas del 16 al 20 del Algoritmo 3.3), para cada par de *landmarks* l_1, l_2 si l_2 no pertenece a los *landmarks possibly before* de l_1 se establece el enlace dependiente $l_1 \prec_d l_2$ de acuerdo con la Definición 3.9.

Algoritmo 3.4 Función *findDisjuntiveSets*

```
1:  Input: l, lms_l
2:  DS = ∅
3:  a_0 = seleccionar acción de fa_DL(l)
4:  precs = pre(a_0) − lms_l
5:  for all p ∈ precs do
6:     ds_predicate(p) = {p}
7:     DS = DS ∪ ds_predicate(p)
8:  end for
9:  for all a ∈ fa_DL(l) − a0 do
10:    precs = pre(a) − lms_l
11:    for all p ∈ precs do
12:       ds_predicate(p) = ds_predicate(p) ∪ {p}
13:       if ds_predicate(p) ∉ DS then
14:          return nulo
15:       end if
16:    end for
17: end for
18: return DS
```

El cálculo de *landmarks* en la aproximación DL es esencialmente el mismo que en la aproximación LM, pero hay dos diferencias importantes entre las dos aproximaciones:

- DL construye el RPG hasta el punto fijo ($RPGL_f$), a diferencia de LM donde se construye el RPG hasta el primer nivel en el que todos los objetivos del problema ($\mathcal{G}$) se consiguen ($RPGL_g$). Esto garantiza que los *landmarks* extraídos

en DL son válidos sin necesidad de verificación, ya que se consideran todas las posibilidades de conseguir un literal y, por tanto, se garantiza la correctitud de los *landmarks* y órdenes encontrados.

- Como el RPG se calcula hasta el punto fijo, calcular $fa_{DL}(l)$ para cada *landmark* l requiere construir el RPG excluyendo todas las acciones que añaden l ($RPGL_f(\neg l)$), lo que implica que el cálculo de los *first achievers* de la aproximación DL es distinta de la proporcionada en la Definición 3.6.

Los *landmarks disyuntivos* se utilizan como un paso intermedio para extraer más *landmarks* aunque también se pueden utilizar para guiar el proceso de búsqueda. En esencia, $RPGL_f(\neg l)$ tiene en cuenta todos los literales que pueden ser *possibly before* a un *landmark* dado l, mientras que $RPGL_g$ representa sólo un subconjunto de estos literales.

3.4.3. La aproximación DTG

Otra aproximación para la extracción y uso de los *landmarks* en problemas de planificación es el enfoque presentado por Richter *et al.* [95]. Lo esencial del método desarrollado en este trabajo se basa en los enfoques LM y DL pero introduce una extensión adicional que, según los autores, encuentra generalmente más *landmarks* y órdenes.

Concretamente, esta aproximación, denominada DTG, adapta las aproximaciones LM y DL al formalismo SAS^+ [3]. En principio, DTG utiliza el mismo algoritmo que LM pero opta por un enfoque más general para conseguir los *landmarks disyuntivos* de DL. Por otra parte, la aproximación DTG incorpora un paso adicional que extrae más *landmarks* a partir de la información codificada en los grafos de transición de dominio (DTG). El formalismo SAS^+ genera los DTG, unas estructuras compactas desde las cuales se puede inferir información relevante. Un DTG de una variable estado es una representación de las formas en que la variable puede cambiar su valor. Estas estructuras codifican las dependencias entre los valores que una variable puede tomar, siendo esta información muy útil para extraer nuevos *landmarks*. Así, mientras LM sólo hace uso de dependencias causales para extraer *landmarks*, el enfoque DTG también explora dependencias entre los valores de las variables.

En particular, el enfoque DTG construye un DTG para cada variable estado v

que aparece tanto en los literales de la inicial situación $\mathcal{I}$ como en los objetivos $\mathcal{G}$. En un DTG, el nodo inicial es un *landmark* $l_0 =< v,\ d_0 >$, donde d_0 es el valor que v toma en $\mathcal{I}$, y el nodo final es un *landmark* $l_g =< v,\ d_g >$, donde d_g es el valor que v toma en $\mathcal{G}$. Si hay un nodo cuyo valor es d_i y se produce en cada camino desde d_0 hasta d_g, entonces $l_i =< v,\ d_i >$ es un *landmark* (que puede ser ordenado delante de l_g). Al calcular los grafos de transición de dominio, todas las asignaciones a v que no están en $RPGL_f(\neg l_g)$ se eliminan, ya que sólo podrían ocurrir después de l_g.

Como veremos en la siguiente sección, el conjunto adicional de *landmarks* que se obtienen con los $DTGs$ está subsumido en el conjunto de *landmarks* obtenidos con nuestra aproximación FULL.

3.4.4. La aproximación Pro

La aproximación Pro [118] presenta importantes diferencias con respecto a las aproximaciones anteriores. En primer lugar, la extracción de *landmarks* se realiza en un grafo de planificación (PG) [7] en lugar de en un RPG. En segundo lugar, se utiliza un proceso de encadenamiento hacia adelante para propagar la información desde el estado inicial hasta el nivel en el que el PG alcanza un punto fijo. En tercer lugar, se extraen *landmarks* literal y *landmarks* acción. La aproximación Pro utiliza una representación compacta de los elementos de un literal y de una acción, denominada *etiqueta*, y que se utiliza para realizar la propagación de la información en los niveles del PG. El método de propagación utilizado elimina la necesidad de la verificación de los *landmarks* obtenidos y funciona como sigue:

1. En el nivel de nodos acción A_0, la etiqueta de cada nodo acción representa a la propia acción.

2. Un nodo literal l del PG se etiqueta con la intersección de las etiquetas de sus nodos acción predecesores (es decir, las acciones que añaden l).

3. A partir del nivel A_0, un nodo acción a del PG se etiqueta con la unión de las etiquetas de su nodos literal predecesores (es decir, las precondiciones de a).

4. Cuando se completa la propagación, cada valor en la etiqueta de un nodo literal que sea objetivo del nivel final es un *landmark*.

El algoritmo de propagación de la aproximación Pro es más costoso que las aproximaciones que necesitan verificar *landmarks* pero es sustancialmente menos costoso cuando el número de *landmarks* a verificar es muy alto. En cambio, el cálculo de órdenes entre *landmarks* de la aproximación Pro es mucho más complejo que el de las aproximaciones LM, DL y DTG.

3.4.5. Comparativa entre estas aproximaciones

En esta sección se presenta un estudio sobre 12 dominios de planificación de las competiciones internacionales de planificación [63] que analiza el número de *landmarks*, *landmarks disyuntivos* y relaciones de orden obtenidas con las aproximaciones presentadas en la sección anterior. Los dominios utilizados son: gripper de IPC1, logistics y blocks de IPC2; depots, driverlog, zenotravel, rovers y freecell de IPC3; pipesworld, storage y openstacks[2] de IPC5 y elevator de IPC6.

Como se ha comentado, no sólo es importante extraer la mayor cantidad posible de *landmarks* sino también ordenarlos parcialmente a fin de obtener un esqueleto del plan solución. Por esta razón, en la comparativa que se ha realizado también se tiene en cuenta el número de órdenes entre los *landmarks* obtenidos.

Para realizar el análisis comparativo, hemos utilizado nuestra propia implementación de las aproximaciones LM, DL y Pro. La aproximación DTG la hemos probado ejecutando el planificador LAMA [96] que está basado en esta aproximación. En la Figura 3.4 puede verse un esquema que muestra las principales características de cada una de las cuatro aproximaciones.

La Tabla 3.1 muestra los resultados obtenidos para las distintas aproximaciones. La columna "*landmarks* individuales" muestra el número de *landmarks* obtenidos para todos los problemas de cada dominio. La columna "*landmarks* disyuntivos" muestra el número de *landmarks disyuntivos* obtenidos para todos los problemas de cada dominio (la aproximación Pro no se muestra porque esta aproximación no calcula este tipo de *landmarks*). La columna "relaciones de orden" muestra el número de órdenes obtenidos para todos los problemas de cada dominio; el resultado se muestra con el formato `x/y` que denota *órdenes necesarios*/*órdenes dependientes*. La tabla no muestra el número de órdenes obtenidos por la aproximación DTG ya

[2]No se pudieron obtener resultados para el dominio OpenStacks con la aproximación DTG debido a algunos problemas con la codificación del dominio.

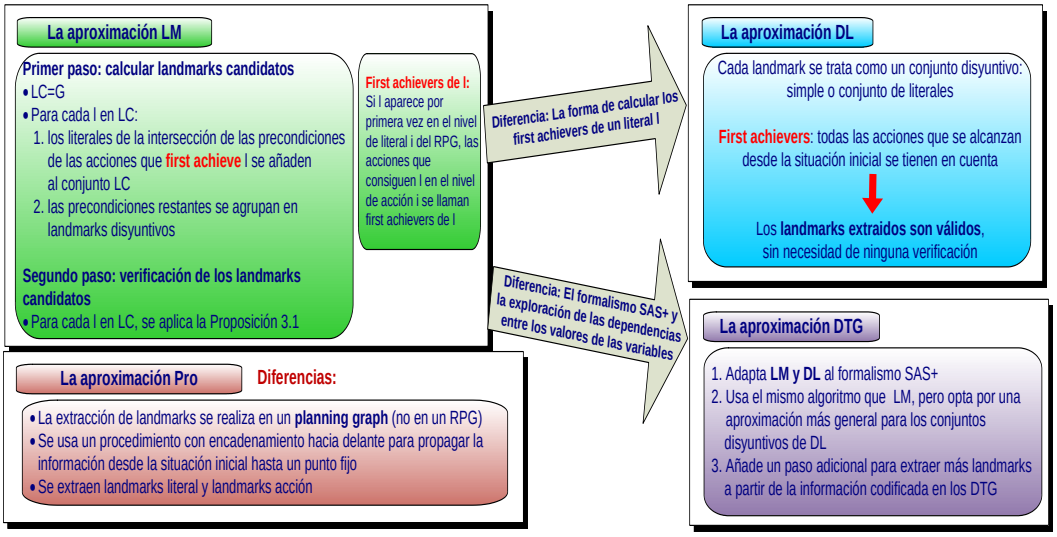

Figura 3.4: Resumen aproximaciones

que el planificador *LAMA* devuelve valores en los que se consideran los órdenes redundantes, los cuales se eliminan en las aproximaciones LM y DL y, por tanto, no son comparables. Un orden redundante es aquel que permite alcanzar un *landmark* l desde un *landmark* l' por dos caminos distintos. El cálculo de los órdenes entre *landmarks* de la aproximación Pro se omite por ser mucho más complejo. En la tabla también puede verse que el valor de los órdenes dependientes de la aproximación LM es siempre 0 ya que esta aproximación no calcula órdenes de dependencia.

Respecto a los resultados obtenidos, como muestra la tabla 3.1, ninguna de las aproximaciones utilizadas ha conseguido extraer *landmarks* individuales de los dominios driverlog y zenotravel ya que estos dominios disponen de varios recursos para alcanzar un mismo objetivo. En los dominios logistics y freecell, las aproximaciones Pro y DTG superan, respectivamente, a las otras aproximaciones, mientras que para el resto de dominios es la aproximación LM la que siempre obtiene un número igual o mayor de *landmarks* individuales que el resto de aproximaciones, número que aumenta significativamente en los dominios depots, pipesworld y elevator.

Comparando LM y DL, se puede observar que LM consigue obtener un mayor número de *landmarks* individuales. LM selecciona un subconjunto de *landmarks* candidatos y, posteriormente, aplica un proceso de verificación sobre todos los literales

Dominio	Landmarks individuales				Landmarks Disyuntivos		
	LM	DL	DTG	Prop	LM	DL	DTG
Gripper	20	20	20	-	**480**	460	460
Logistics	536	536	630	**665**	88	**311**	81
Blocks	523	523	523	523	0	0	0
Depots	**341**	224	283	140	277	**365**	157
Driverlog	0	0	0	-	146	**329**	120
ZenoTravel	2	2	1	-	123	**283**	80
Rovers	**76**	75	75	**76**	191	**277**	159
Freecell	864	596	**1032**	510	**65**	0	0
PipesWorld	**149**	123	118	-	336	**362**	31
Storage	**27**	25	25	-	112	**275**	123
OpenStacks	1024	1024	-	-	20	**40**	-
Elevator	**264**	123	217	-	315	**749**	242

Tabla 3.1: Comparación de *landmarks* entre las distintas aproximaciones.

que pueden ser *landmarks*; por ejemplo, en el dominio depots, LM obtiene 341 *landmarks* frente a los 224 de la aproximación DL. DL carece de algunos *landmarks* individuales ya que al considerar todas las posibles formas de conseguir un literal, sólo selecciona los literales confirmados como *landmarks*. Por esta misma razón, la aproximación DL obtiene un mayor número *landmarks disyuntivos* que la aproximación LM en la mayoría de dominios como puede verse en la Tabla 3.1. Por otro lado, algunos *landmarks* individuales candidatos que LM descarta en la verificación es probable que sean *landmarks disyuntivos* en DL. Esto puede ocurrir, por ejemplo, en el dominio logistics ya que el número de *landmarks* individuales es el mismo para ambas aproximaciones y el número de *landmarks disyuntivos* es significativamente mayor en la aproximación DL.

Intuitivamente el mayor número de *landmarks* individuales se concentra en la parte más cercana al nivel inicial del *RPG* debido a que existen menos alternativas para conseguir un determinado literal. A medida que el proceso de construcción del *RPG* continúa, las posibilidades de aplicar un mismo operador con diferentes recursos aumenta y por tanto existen más alternativas para conseguir un mismo literal. Por este motivo, la aproximación LM extrae una mayor cantidad de *landmarks* individuales que la aproximación DL, tal y como puede observarse en la Tabla 3.1, ya

Dominio	Relaciones de orden	
	LM	DL
Gripper	**2320/0**	940/0
Logistics	1331/0	**1606/426**
Blocks	2131/0	**1956/563**
Depots	**2010/0**	1376/368
Driverlog	540/0	**539/88**
ZenoTravel	367/0	**455/119**
Rovers	**998/0**	917/0
Freecell	**1650/0**	676/0
PipesWorld	**1367/0**	835/469
Storage	504/0	**565/2219**
OpenStacks	**11884/0**	**11884/85**
Elevator	**1473/0**	1384/14

Tabla 3.2: Comparación de relaciones de orden entre las distintas aproximaciones.

que el RPG_g tiene menos niveles que el RPG_f.

Por otra parte, a raíz de nuestros experimentos podemos afirmar que es difícil encontrar *landmarks* individuales a partir de conjuntos disyuntivos que contienen muchos elementos, lo que significa que DL tiene más dificultades para extraer *landmarks* individuales, ya que trabaja principalmente con estos conjuntos.

El enfoque DTG es capaz de obtener más *landmarks* individuales que LM en el dominio freecell gracias a la información proporcionada por los *DTGs*. Se obtienen 1032 *landmarks* frente a 864 de la aproximación LM. Sorprendentemente, la aproximación DTG obtiene menos *landmarks disyuntivos* que DL a pesar de utilizar el mismo método de extracción. Este menor número de *landmarks disyuntivos* de DTG se debe a que algunos de los *landmarks disyuntivos* calculados por DL se transforman en *landmarks* individuales en la aproximación DTG gracias a la información porporcionada por los *DTGs*, como por ejemplo en el dominio depots o en el dominio logistics.

En cuanto al enfoque Pro, sólo mostramos los *landmarks* individuales para los dominios depots, freecell, logistics, rovers y blocks debido al alto coste de construir el *PG*. A pesar de que los autores de Pro afirman, en el trabajo presentado en el *ICAPS Doctoral Consortium* del año 2003 [118], que el número de *landmarks* extraídos es

mayor que en la aproximación LM, en nuestra comparativa esto sólo sucede en el dominio logistics porque, para el resto de dominios, Pro sólo pudo resolver 14 problemas (sobre 20) en el dominio depots y 12 (de un total de 20) en el dominio freecell. Aún cuando el número total de *landmarks* obtenido por Pro es menor porque ha resuelto menos problemas, vale la pena señalar que algunos *landmarks* encontrados por la aproximación Pro en los dominios depots y freecell (8 *landmarks* para el dominio depots y 113 para el dominio freecell, respectivamente) no se han encontrado con la aproximación LM.

Por último, como puede observarse en la Tabla 3.2, no hay una tendencia clara respecto a los órdenes obtenidos por las aproximaciones LM y DL. En general, el número de órdenes depende sobre todo del número total de *landmarks* (individuales + disyuntivos) y, en consecuencia, la aproximación que más *landmarks* (sumando ambos valores) extrae es también la aproximación que obtiene más órdenes. Por ejemplo, en el dominio gripper la aproximación LM obtiene 500 *landmarks* (20 + 480) frente a 480 *landmarks* de la aproximación DL y es también la aproximación LM la que obtiene un mayor número de órdenes necesarios. En cambio, en el dominio logistics la aproximación LM obtiene 624 *landmarks* , la aproximación DL obtiene 847 *landmarks* y, en este caso, es la aproximación DL la que obtiene un mayor número de órdenes necesarios. En cualquier caso, es importante remarcar que LM casi siempre obtiene más órdenes necesarios, que son los órdenes más útiles para construir el esqueleto del plan de solución.

3.5. Aproximación FULL

En esta sección se presenta la aproximación FULL como resultado de la combinación de los algoritmos de extracción de *landmarks* presentados en la Sección 3.4. Básicamente, la aproximación FULL se basa en la aplicación secuencial de la técnica LM, una propagación de literales (Pro) y, finalmente, la aplicación de la técnica DL para conseguir el conjunto de literales disyuntivos. De acuerdo con la Tabla 3.1, el método LM es el que consigue mayor número de *landmarks* individuales. Por este motivo, la aproximación FULL utiliza LM como base para la extracción inicial del conjunto de *landmarks* y órdenes. También puede observarse en la Tabla 3.1 que los mejores resultados para los *landmarks disyuntivos* los proporciona la aproximación

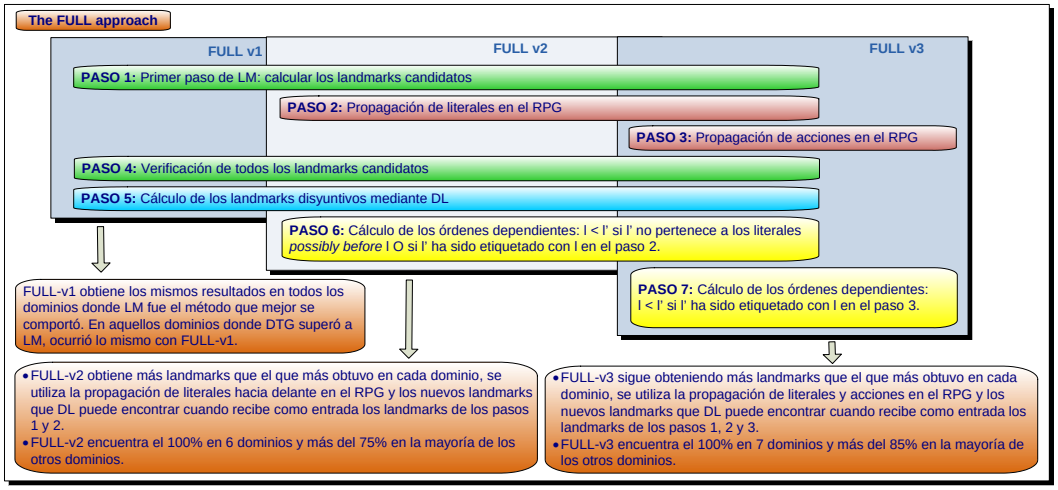

Figura 3.5: Aproximación FULL

DL, por lo que se utiliza esta aproximación para completar el conjunto inial de *landmarks*. Con esta configuración inicial, se desarrollaron tres versiones diferentes de la aproximación FULL: FULL-v1, FULL-v2 y FULL-v3. La Figura 3.5 muestra cada una de estas versiones. A continuación se comentan las principales características de cada versión.

La versión FULL-v1 simplemente consiste en unir las técnicas de LM y DL. En primer lugar se calcula un conjunto de *landmarks* candidatos y se calculan los órdenes necesarios mediante la técnica LM (líneas del 12 al 14 del Algoritmo 3.1 de LM). Aunque LM también extrae un conjunto de *landmarks disyuntivos* (líneas del 15 al 19 del Algoritmo 3.1), este conjunto sólo se utiliza para extraer más *landmarks* candidatos. A continuación se verifican los *landmarks* candidatos (líneas del 22 al 27 del Algoritmo 3.1), los cuales son utilizados posteriormente por el método DL para extraer *landmarks disyuntivos* ya que este método utiliza un cálculo más completo y robusto de los *first archievers*, proporcionando *landmarks disyuntivos* de mejor calidad (líneas del 6 al 14 del Algoritmo 3.3 de DL). Adicionalmente, durante el proceso de verificación se actualizan los *first archievers* utilizando la función $fa_{DL}()$ y se calcula $pb()$ para cada *landmark* de acuerdo a la aproximación DL. Durante el proceso de cálculo de *landmarks disyuntivos* mediante el método DL se pueden inferir nuevos *landmarks* que no necesitarían ser verificados. La parte izquierda de

la Figura 3.5 esquematiza este proceso.

FULL-v2 es una versión mejorada de FULL-v1 en la que se aplica una propagación de literales similar a la presentada en el trabajo de Zhu y Givan [118] pero esta propagación se realiza en un RPG en lugar de en un PG. Para la propagación de la información se utiliza el mismo concepto de *etiquetas* de la aproximación Pro. Particularmente, para realizar la propagación de literales se define el término *literal dependency label* de la siguiente forma:

Definición 3.11. *Los **literal dependency labels** de un literal l_i es el conjunto de literales que son necesarios para conseguir el literal l_i y se denota como $lit\_labels_l(l_i)$:*

$$lit\_labels_l(l_i) = \{l_i\} \cup \left(\bigcap_{\forall a \in fa_{LM}(l_i)} lit\_labels_a\,(a) \right)$$

donde

$$lit\_labels_a(a) = \left(\bigcup_{\forall c \in Cond(a)} lit\_labels_l\,(c) \right)$$

■

El cálculo de los *literal dependency labels* de un literal l_i es una función recursiva que finaliza cuando $l_i \in \mathcal{I}$ ya que $fa_{LM}(l_i) = \emptyset$ lo que implica que $\left(\bigcap_{\forall a \in fa_{LM}(l_i)} lit\_labels_a\,(a) \right) = \emptyset$. La función $lit\_labels_a(a)$ representa el conjunto de literales que son necesarios para aplicar la acción a. Los pasos de la versión FULL-v2 son los siguientes (ver parte central de la Figura 3.5):

- Paso 1: Se extrae un primer conjunto de *landmarks* candidatos a través de la primera etapa de LM. También se calculan los órdenes necesarios.
- Paso 2: Se añaden más *landmarks* candidatos a través de la propagación de literales en un RPG: l se añade a *landmarks* candidatos si $l \in \bigcup_{\forall g \in \mathcal{G}} lit\_labels_l\,(g)$.
- Paso 4: Se verifican todos los *landmarks* candidatos y se calculan las funciones $fa_{DL}()$ y $pb()$ para cada *landmark*.
- Paso 5: Se calculan los *landmarks* disyuntivos a través de DL tomando como entrada los *landmarks* verificados.

- Paso 6: Se calculan los órdenes de dependencia: $l \prec_d l'$ (1) si l' no pertenece a los literales *possibly before* de l; (2) si $l \in lit\_labels_l(l')$.

Dado que la versión FULL-v2 realiza la propagación sobre un RPG, este proceso es menos costoso que el que aplica la aproximación Pro, pero obliga a realizar una verificación de los *landmarks* obtenidos. Por dicho motivo, los *landmarks* que se extraen de la propagación de literales se añaden a los *landmarks* candidatos de LM y, a continuación, se verifican todos ellos. Por otro lado, los órdenes necesarios se añaden en el primer paso de FULL-v2 mientras que los órdenes dependientes se pueden añadir en dos etapas:

1. Tras aplicar el proceso de propagación, se establece un orden dependiente entre l y l' ($l \prec_d l'$) si $l \in lit\_labels_l(l')$.

2. Una vez calculados los *landmarks disyuntivos*, se establece un orden dependiente entre l y l' ($l \prec_d l'$) si l' no pertenece a los literales *possibly before* de l.

Finalmente, FULL-v3 es una versión mejorada de FULL-v2 en la que se aplica, adicionalmente, una propagación de acciones similar a la presentada en el trabajo de Zhu y Givan [118] pero realizando esta propagación en un RPG. Para realizar la propagación de acciones se define *action dependency label* de la siguiente forma:

Definición 3.12. *Las* ***action dependency labels*** *del literal l_i es el conjunto de acciones necesarias para conseguir el literal l_i y se denota como $act\_labels_l(l_i)$:*

$$act\_labels_l(l_i) = \left(\bigcap_{\forall a_i \in fa_{LM}(l_i)} act\_labels_a(a_i) \right)$$

donde

$$act\_labels_a(a_i) = \{a_i\} \cup \left(\bigcup_{\forall c \in Cond(a)} act\_labels_l(c) \right)$$

■

El cálculo de las *action dependency labels* de un literal l_i es una función recursiva que termina cuando $l_i \in \mathcal{I}$ ya que $fa_{LM}(l_i) = \emptyset$. La función $act\_labels_a(a)$ representa el conjunto de acciones que son necesarias para aplicar la acción a. Los pasos de la versión FULL-v3 son los siguientes (ver parte derecha de la Figura 3.5):

- Pasos 1 y 2 de la versión FULL-v2.
- Paso 3: Se añaden más *landmarks* candidatos a través de la propagación de acciones en un RPG: l se añade a los *landmarks* candidatos si $l \in Eff(a)$: $a \in \bigcup_{\forall g \in \mathcal{G}} act\_labels_l(g)$.
- Pasos 4, 5 y 6 de la versión FULL-v2.
- Paso 7: Se calculan los órdenes de dependencia, si $l' \in Eff(a)$: $a \in act\_labels_l(l)$, se añade el orden $l \prec_d l'$.

La versión FULL-v3 mejora la versión FULL-v2 al incluir la propagación de las acciones. Aunque este proceso no tiene un alto coste computacional, sigue siendo necesario verificar los *landmarks* obtenidos al haber realizado la propagación de las acciones sobre un RPG.

A continuación se presenta una comparativa entre las aproximación de la Sección 3.4 (LM, DL, DTG y Pro) que obtuvieron los mejores resultados en cada dominio frente a las diferentes versiones de FULL. En la Tabla 3.3 se puede ver la comparativa respecto al número de *landmarks* individuales, donde las columnas etiquetadas como "*% wrt T*" indican el porcentaje de *landmarks* encontrados respecto al número total de *landmarks* que se pueden encontrar en un dominio (el número total de landmarks se obtuvo aplicando la *Proposición 1* a cada literal de cada problema de un dominio). En la Tabla 3.4 se puede ver la comparativa respecto al número de *landmarks* disyuntivos

De acuerdo a los resultados de la Tabla 3.3, la versión FULL-v1 (F-v1) obtiene los mismos resultados en todos los dominios donde LM ha sido la mejor aproximación. En aquellos dominios en los que DTG supera a LM, DTG también supera a la versión FULL-v1, como ocurre en el dominio freecell.

En la Tabla 3.3 se puede observar que la versión FULL-v2 (F-v2) extrae más *landmarks* que la mejor aproximación para cada uno de los dominios. Este aumento en el número de *landmarks* respecto a FULL-v1 se da, en parte, por la propagación de literales en el RPG, lo que nos permite obtener un conjunto adicional de *landmarks* respecto a los obtenidos por la aproximación LM. Gracias a este conjunto adicional de *landmarks* que se obtiene por la propagación de literales, el paso 5 de la versión FULL-v2 (Figura 3.5) también encuentra más *landmarks* individuales y

	Landmarks Individuales					
Dominio	Mejor aprox.	% wrt T	F-v1	F-v2	F-v3	% wrt T
Gripper	20	100 %	20	20	20	100 %
Logistics	665 (Prop)	100 %	536	**665**	**665**	100 %
Blocks	523	91.9 %	523	523	**569**	100 %
Depots	341 (LM)	72.7 %	341	353	**443**	94.6 %
Driverlog	0	100 %	0	0	0	100 %
ZenoTravel	2	100 %	2	2	2	100 %
Rovers	76 (LM)	85.3 %	76	76	**88**	98.9 %
Freecell	1032 (DTG)	87.9 %	864	1143	**1153**	98.2 %
PipesWorld	149 (LM)	80.1 %	149	149	**158**	84.9 %
Storage	27 (LM)	17.7 %	27	29	**150**	98.7 %
OpenStacks	1024	98 %	1024	**1044**	**1044**	100 %
Elevator	264 (LM)	94.6 %	264	**279**	**279**	100 %

Tabla 3.3: Comparación de los *landmarks* individuales de la mejor aproximación de la Sección 3.4 con las versiones de la aproximación FULL.

	Landmarks Disyuntivos			
Dominio	Mejor aprox.	F-v1	F-v2	F-v3
Gripper	**480**	460	460	460
Logistics	311	311	311	311
Blocks	0	0	0	0
Depots	365	365	365	**412**
Driverlog	329	329	329	329
ZenoTravel	283	283	283	283
Rovers	277	277	277	277
Freecell	**65**	0	0	5
PipesWorld	362	411	411	**412**
Storage	275	275	275	**343**
OpenStacks	40	40	**60**	**60**
Elevator	749	842	**849**	**849**

Tabla 3.4: Comparación de los *landmarks* disyuntivos de la mejor aproximación de la Sección 3.4 con las versiones de la aproximación FULL.

disyuntivos. Por lo tanto, el aumento en el número de *landmarks* de FULL-v2 no sólo es debido a la propagación de literales, sino también a los nuevos *landmarks* que DL es capaz de encontrar cuando recibe como entrada los *landmarks* de los pasos 1 y 2. Esto se puede observar en los dominios OpenStacks y Elevator de la Tabla 3.4 donde donde FULL-v2 extrae un mayor número de *landmarks* disyuntivos que FULL-v1 a pesar de utilizar el mismo método de cálculo de *landmarks* disyuntivos.

Por último, la Tabla 3.3 muestra que la versión FULL-v3 (F-v3) es capaz de extraer más *landmarks* que la versión FULL-v2 para todos los dominios. FULL-v3 obtiene todos los *landmarks* posibles para 7 dominios y encuentra más del 85 %

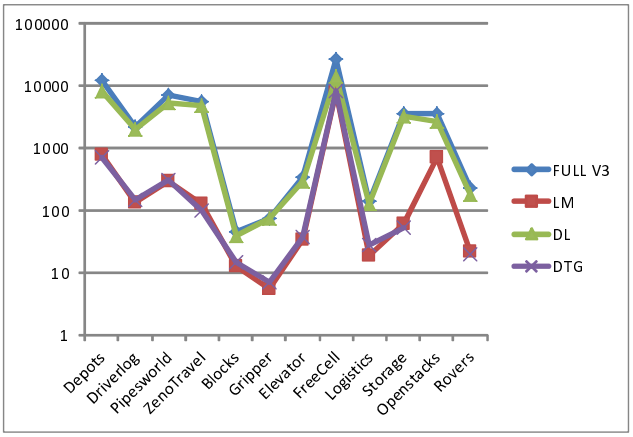

Figura 3.6: Tiempo de computo de FULL-v3 vs LM vs DL y DTG.

(casi el 98 %) de los *landmarks* posibles en el resto de dominios. Este aumento en el número de *landmarks* se da por la propagación de acciones en el *RPG* y también por los nuevos *landmarks* individuales y disyuntivos que DL es capaz de encontrar cuando recibe como entrada los *landmarks* de los pasos 1, 2 y 3 (Figura 3.5). Esto se puede observar en la Tabla 3.4 donde FULL-v3 extrae un mayor número de *landmarks disyuntivos* que FULL-v2 en los dominios Depots, PipesWord y Storage.

En cuanto a las **relaciones de orden** (los datos no se muestran en las tablas), en general, las distintas versiones de la aproximación FULL obtienen un mayor número de órdenes en comparación con LM y DL, ya que extrae más *landmarks* que estos dos métodos. Respecto al **rendimiento** (ver Figura 3.6), el coste computacional de DTG y LM son similares, mientras que la aproximación DL es poco más lenta. Por otra parte, la propagación de literales y acciones en FULL-v3 es muy poco costosa, añadiendo unos pocos milisegundos al coste de DL.

3.6. Conclusiones

En este capítulo se han presentado los resultados del análisis de las distintas aproximaciones que existen para la extracción de *landmarks*. También se ha realizado un estudio de las fortalezas y debilidades de cada una de ellas y se ha presentado una comparativa del número de *landmarks*, *landmarks disyuntivos* y órdenes que

obtiene cada una de las aproximaciones. A partir de los resultados obtenidos, se diseñó la aproximación FULL como una combinación de las distintas aproximaciones existentes, la cual proporciona mejores resultados respecto al número de landmarks y órdenes extraídos. En el siguiente capítulo se utilizará el grafo de *landmarks* que se obtiene a partir de la aproximación FULL-v3 para extenderlo al contexto de problemas de planificación temporal con restricciones.

Capítulo 4

Modelo de Landmarks Temporales

4.1. Introducción

La extensión del concepto de *landmark* a un entorno temporal conlleva la inclusión del tiempo en la especificación del problema y en la definición del *landmark*. Mientras que los *landmarks* STRIPS hacen referencia a la estructura lógica de las relaciones entre las acciones del plan, uno de los problemas más relevantes en los dominios temporales es situar las acciones en el tiempo teniendo en cuenta las interacciones de una acción con otra y las consecuencias en la duración total del plan. De este modo, en un problema de planificación temporal se pueden encontrar nuevas relaciones producidas por *causas temporales*, esto es, relaciones producidas por una consideración explícita del tiempo. Dichas relaciones extienden las restricciones de orden entre landmarks y se representan como restricciones temporales.

En un problema de planificación temporal, el tiempo forma parte de la formulación del problema y de la resolución del mismo. Dado el conjunto de planes temporales para un determinado problema, todos ellos compartirán el mismo conjunto de *landmarks* STRIPS. Adicionalmente, si se considera el subconjunto de planes temporales cuya duración no supera un determinado umbral entonces podremos encontrar un conjunto adicional de *landmarks* compartido por dicho subconjunto de planes.

Consideremos el problema del dominio logistics en un entorno temporal como el mostrado en la Figura 4.1. Los enlaces entre las ciudades están marcados con la duración del vuelo y asumimos que las operaciones de subir y bajar del avión

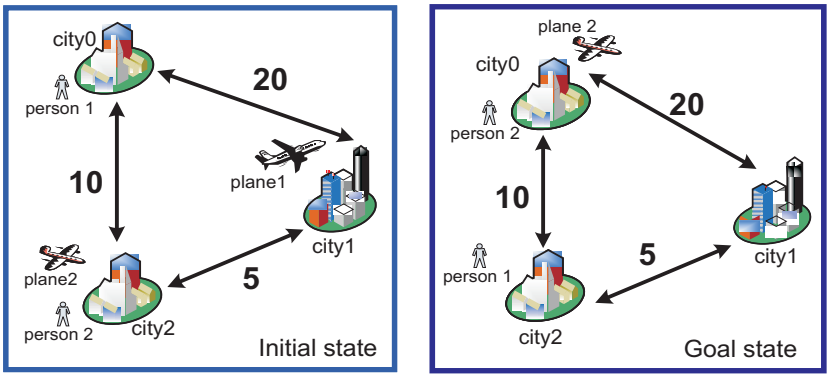

Figura 4.1: Un problema del dominio *logistics* temporal

cuestan dos unidades de tiempo. No se requiere un destino final para el plane1 en el estado objetivo.

Desde una perspectiva STRIPS, no se obtiene ningún *landmark* para este problema, exceptuando los literales de la situación inicial y objetivo. La razón es que hay varios recursos diferentes (dos aviones) para conseguir los objetivos y no se puede determinar cuál de ellos se utilizará. Sin embargo, desde una perspectiva temporal, es posible extraer alguna información relevante. Particularmente, existen diferentes planes solución para este problema, cada uno con una duración diferente. La duración del plan óptimo es de 29 unidades de tiempo; también se puede encontrar planes de duración 33, 34, 38, 50 e incluso planes de mayor duración. Si consideramos el conjunto completo de planes temporales que son solución no podemos obtener ningún *landmarks* (este escenario sería equivalente a un entorno STRIPS). Sin embargo, si consideramos los planes temporales hasta una duración determinada, podemos deducir algunas conclusiones interesantes. Consideremos los siguientes cuatro planes temporales:

- Plan1: 29 unidades de tiempo; esta es la solución óptima. Este plan usa ambos aviones para transportar a los pasajeros: plane1 para person1 y plane2 para person2, y la ruta de plane1 para llegar a city0 pasa por city2.

- Plan2: 31 unidades de tiempo; este plan sólo utiliza plane1 para transportar a los dos pasajeros mientras, simultáneamente, plane2 se dirige a su destino.

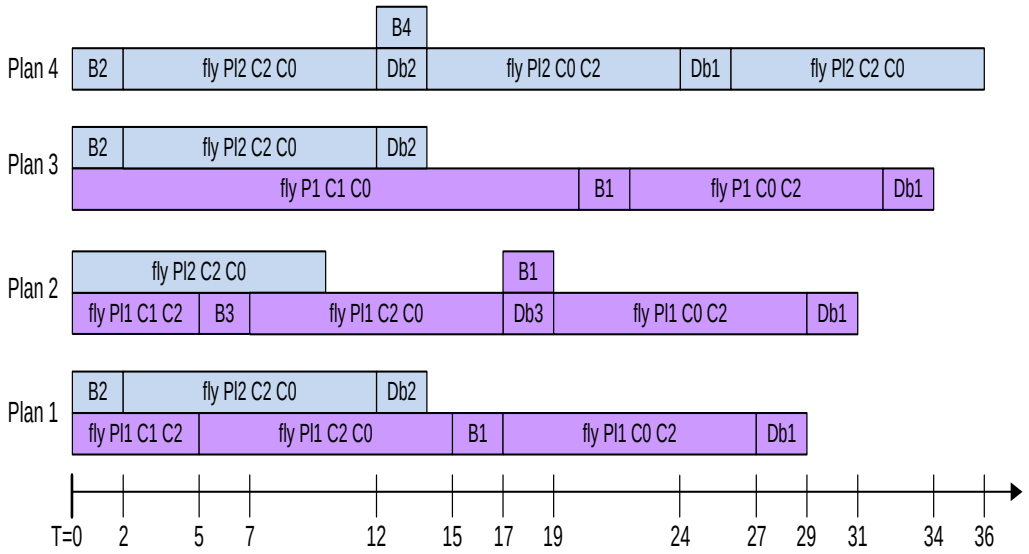

Figura 4.2: Varios planes para el problema de la Figura 4.1

- Plan3: 34 unidades de tiempo; este plan utiliza los dos aviones para transportar a los pasajeros: plane1 para person1 y plane2 para person2, y plane1 alcanza city0 mediante la conexión directa city1-city0, que es un trayecto más largo que ir a través de city2.

- Plan4: 36 unidades de tiempo; este plan sólo utiliza plane2 para transportar a los dos pasajeros.

La Figura 4.2 muestra los cuatro planes donde Pl1 y Pl2 representan el plane1 y plane2, respectivamente y C0, C1 y C2 representan las ciudades city0, city1 y city2, respectivamente. La acción B1 representa la acción (Board plane1 person1 city0), B2 representa la acción (Board plane2 person2 city2), B3 representa la acción (Board plane1 person2 city2) y B4 representa la acción (Board plane2 person1 city0). Mientras que la acción Db1 representa la acción (Debark plane1 person1 city2), Db2 representa la acción (Debark plane2 person2 city0), Db3 representa la acción (Debark plane1 person2 city0) y Db4 representa la acción (Debark plane2 person1 city2). Haciendo un análisis de los cuatro planes podemos obtener las siguientes conclusiones:

- Si tomamos como referencia todos los planes Π cuya $dur(\Pi) \leq 36$ (es decir, los cuatro planes), no podemos extraer ninguna información común (*landmarks*).

- Considerando los planes Π tal que $dur(\Pi) \leq 34$ entonces podemos deducir que person1 debe ser transportada con plane1; particularmente, el literal (in person1 plane1) debe ocurrir siempre antes del instante 22.

- Considerando los planes Π tal que $dur(\Pi) \leq 31$ podemos concluir que person1 debe ser transportada con plane1 y plane1 debe alcanzar city0 a través city2; esto significa que el literal (at plane1 city0) debe ocurrir siempre antes del instante 17.

A partir de este análisis, se puede inferir un conjunto común de literales a los planes solución dentro de un margen temporal determinado. Esto significa que es posible extraer información útil cuando se considera el conjunto de planes delimitados por una duración máxima. Adicionalmente, esta característica también es relevante cuando el usuario está interesado en obtener planes que no superen una determinada duración.

Con la introducción de los *timed initial literals* en PDDL2.2 [60] y de las restricciones de trayectoria de estados en PDDL3.0 [43] se incorpora la posibilidad de establecer un límite de tiempo máximo para un objetivo. En este caso, la duración máxima del plan vendrá determinada en función de los límites de tiempo de los objetivos del problema.

En las secciones siguientes se introducen varias definiciones para planes temporales con restricciones, se define el concepto de landmark en un entorno temporal y se detalla el proceso de construcción del grafo de landmarks temporales.

4.2. Problema de Planificación Temporal con Restricciones

Como se ha comentado en la Sección 2.3.2, asumimos la semántica del modelo temporal definido en PDDL2.1 [37] ya que resulta apropiado para la representación de problemas de planificación con referencias explícitas de tiempo. También incorporamos los *timed initial literals* (*TILs*) de PDDL2.2 [60] y las restricciones de trayectoria de estados de PDDL3.0 [43].

A continuación presentaremos los elementos básicos de nuestro modelo temporal.

Definición 4.1. *Un* ***problema de planificación temporal relajado*** $\mathcal{P}_\mathcal{R}$ *es una tripleta* $(\mathcal{A}_\mathcal{R}, \mathcal{I}, \mathcal{G})$ *donde* $\mathcal{A}_\mathcal{R}$ *es un conjunto de acciones durativas en las que se eliminan los efectos que se borran :*

$\mathcal{A}_\mathcal{R} = \{(Cond(a), dur(a), SAdd(a) \bigcup EAdd(a), \emptyset) |$

$(Cond(a), dur(a), SAdd(a) \bigcup EAdd(a), SDel(a) \bigcup EDel(a)) \in A\}.$ ∎

Definición 4.2. *Sea* $\mathcal{P}_\mathcal{R} = (\mathcal{A}_\mathcal{R}, \mathcal{I}, \mathcal{G})$ *un problema de planificación temporal relajado. El* ***grafo de planificación temporal relajado (TRPG[1])*** *sobre* $P_\mathcal{R}$ *es un grafo dirigido que contiene niveles de literal y niveles de acción:*

(i) $L_0 \leftarrow \mathcal{I}$ *y* $A_0 \leftarrow \{a | \forall p \in SCond(a) \cup Inv(a) : p \in L_0\}$*: nivel de acción que contiene los nodos acción que son aplicables en el estado inicial*

(ii) *En general, el nivel de literal* L_i *es el conjunto de nodos literal que se añaden al grafo después de* i *unidades de tiempo y el nivel de acción* A_i *es el conjunto de nodos acción que son aplicables después de* i *unidades de tiempo. Formalmente:*

- $L_{i+\varepsilon} \leftarrow L_i$
- $A_i \leftarrow A_{i-1} \cup \{a | \forall p \in SCond(a) \cup Inv(a) : p \in L_i\}$
- $\forall a \in A_i$, *asumiendo que* $\forall p \in ECond(a) : p \in L_{i+dur(a)}$
  - $L_{i+\varepsilon} \leftarrow L_{i+\varepsilon} \cup SAdd(a)$
  - $L_{i+dur(a)+\varepsilon} \leftarrow L_{i+dur(a)+\varepsilon} \cup EAdd(a)$ ∎

Al conjunto de literales comprendidos en un nivel i del $TRPG$ lo llamaremos $TRPGL_i$. Del mismo modo, al conjunto de acciones comprendidas en un nivel de acción i del $TRPG$ lo denominaremos $TRPGA_i$. Se denomina $TRPGL_g$ al primer nivel de literal del $TRPG$ donde se consiguen todos los objetivos de $\mathcal{G}$ ($\mathcal{G} \subseteq TRPGL_g$), y

[1] *Del inglés Temporal Relaxed Planning Graph*

$TRPGL_f$ al nivel de literal del punto fijo, es decir, el nivel a partir del cual no se pueden añadir más acciones ni más literales. Obviamente, se cumple que $\mathcal{G} \subseteq TRPGL_f$ (si el problema es resoluble).

Definición 4.3. *Un* ***problema de planificación temporal con restricciones*** $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ *es una tupla donde* $\mathcal{A}$ *es el conjunto de acciones durativas que pueden aplicarse en el dominio,* $\mathcal{I}$ *y* $\mathcal{G}$ *son los conjuntos de literales que representan la situación inicial y objetivo, respectivamente y* $\mathcal{D}$ *es un conjunto de restricciones de la forma* (t, l)*, indicando que el literal* l *debe conseguirse antes de* t *unidades de tiempo.* ∎

$\mathcal{D}$ es el conjunto de todas las restricciones que se especifican en un problema de planificación $\mathcal{P}$ e internamente se representan de la forma (t, l) donde l es un literal de $\mathcal{P}$ y t es el límite de tiempo (o *deadline*) donde l debe conseguirse. Como estamos interesados en aquellos problemas donde se puede establecer un *deadline* $T_{\mathcal{P}}$ para el problema $\mathcal{P}$, es decir, para todos los planes que resuelven $\mathcal{P}$, asumimos que se establece un límite de tiempo t para todos los objetivos del problema, es decir, $\forall g \in \mathcal{G}, \exists (t, g) \in \mathcal{D}$. De este modo:

$$T_{\mathcal{P}} = \max_{(t,l) \in \mathcal{D}} (t) \tag{4.2.1}$$

En PDDL, existen dos formas para expresar una restricción temporal sobre un literal l:

- Explícitamente, utilizando los operadores modales de PDDL3.0, como por ejemplo within [43]: $\langle (S_0, 0), (S_1, t_1), \ldots, (S_n, t_n) \rangle \models (\text{within } t\ l)\ iff\ \exists i : 0 \leq i \leq n \cdot S_i \models g \wedge t_i \leq t$. Por lo tanto, dado un *deadline* representado por (within t l), la restricción (t, l) se añade a $\mathcal{D}$.

  De acuerdo con esta definición, si un objetivo se alcanza más de una vez en el plan, es suficiente que una instancia cumpla la restricción within.

  En la Figura 4.2, la restricción "la persona person2 debe encontrarse en la ciudad city0 antes de 15 u.t." se especificaría como: (within 15 (at P2 C0)).

- Implícitamente, mediante los *TILs* de PDDL2.2, que es una forma de expresar ventanas temporales. La especificación de los *TILs* puede verse en la Sección 2.4.1. Dado un *TIL* cuyo comienzo es (t, l) y cuya finalización es $(t', (not\ l))$ y una acción a tal que $l \in Cond(a)$, los literales $Eff(a)$ estarán supeditados al intervalo $[t, t']$ donde l está activo, y esta información nos permitirá calcular el punto de finalización de la ejecución de la acción a, pudiendo así establecer restricciones temporales sobre los literales de $Eff(a)$.

  Por ejemplo, supongamos un *TIL* que indica que el aeropuerto de la ciudad city0 no está disponible a partir del instante 20. Este dato nos permite calcular la restricción "la persona person2 debe estar en la ciudad city0 antes de 22 u.t." ya que los aviones tendrán que aterrizar en el aeropuerto de city0 antes del instante 20 y sólo a partir de ese instante la persona person2 puede bajarse del avión, que tiene un coste de 2 u.t.

La definición del operador within de PDDL3.0 no implica ningún tipo de persistencia por lo que es suficiente que una instancia de un objetivo cumpla la restricción aunque se dé el caso que la última instancia del objetivo no la cumpla. Por ejemplo, si sobre la Figura 4.2 expresamos las restricciones (within 35 (at P2 C0)) y (within 11 (at Pl2 C0)), el avión plane2 tendría que volar a la ciudad city0 sin cargar ningún pasajero para poder cumplir la restricción (within 11 (at Pl2 C0)) pero después podría volver a la ciudad city2 para cargar a la persona person2 y llevarla a la ciudad city0, en cuyo caso una instancia del objetivo (at Pl2 C0) se cumpliría en el instante 10 y otra instancia del objetivo (at Pl2 C0) se cumpliría en el instante 32. Para poder expresar la persistencia de un literal sería necesaria la anidación de varios operadores modales (within <deadline> (always <goal>)), lo cual no está permitido en la sintaxis estándar de PDDL3.0[2].

Definición 4.4. *Un plan temporal* Π *es un* ***plan solución para un problema de planificación temporal con restricciones*** $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ *si:*

1. $\mathcal{G} \subseteq S_g$, *donde* $\mathcal{I} \rightarrow_{\Pi} S_g$
2. $\forall (t, l) \in \mathcal{D} : \exists t' \leq t : l \in S_{t'}$, *donde* $\mathcal{I} \rightarrow_{\Pi_{t'}} S_{t'}$ ∎

[2]Comunicación personal de Derek Long.

Un plan solución Π alcanza un estado S_g donde se satisfacen los objetivos de $\mathcal{G}$, y los literales l de cada restricción $(t, l) \in \mathcal{D}$ se satisfacen en un estado S'_t tal que $t' \leq t$.

4.2.1. Interferencias entre literales del modelo temporal

En esta sección vamos a estudiar las relaciones *mutex* que surgen entre los literales del modelo temporal. En este punto, nuestro objetivo es identificar sólo *mutex permanentes*, es decir, interferencias que suceden siempre independientemente de la ocurrencia temporal de los literales que participan en el *mutex*. Para ello, seguiremos el mismo análisis que se muestra en en trabajo de Gerevini *et al.* [44] para literales STRIPS y adaptaremos este cálculo de *mutex* a literales del modelo temporal PDDL2.1.

Definición 4.5. *Dos literales l_1 y l_2 son **mutex permanentes** en un contexto temporal si ambos no pueden ser válidos simultáneamente en ningún instante de tiempo.* ∎

Vamos a mostrar las situaciones donde se cumple la Definición 4.5 en nuestro modelo temporal. Según este trabajo, un conjunto de relaciones *mutex permanentes* se puede calcular mediante un proceso de dos etapas: (1) identificar relaciones *mutex potenciales* y (2) verificar dichas relaciones *mutex potenciales* [44].

El concepto de **mutex potencial** entre dos literales l_1 y l_2 se define dentro del contexto de una única acción a y se utiliza para indicar que dos literales l_1 y l_2 que aparecen en $Cond(a)$ o $Eff(a)$ pueden llegar a ser *mutex permanentes* si se verifica el potencial *mutex* entre l_1 y l_2.

Definición 4.6. *Dos literales l_1 y l_2 forman un **mutex potencial** si:*

- *Caso base: dada una acción a, l_1 es un efecto positivo de a y l_2 es un efecto negativo.*
- *Caso recurrente: dada una acción a, l_1 es un efecto positivo de a y l_2 es (potencialmente)* mutex *con la precondición l_3 de a.* ∎

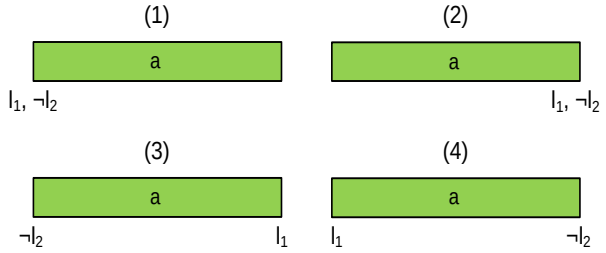

Figura 4.3: Situaciones de *mutex* potenciales y *no-mutex* en el modelo temporal

El caso base se produce en nuestro modelo temporal cuando el efecto positivo y negativo ocurren en el mismo instante de tiempo, es decir $(l_1 \in SAdd(a) \wedge l_2 \in SDel(a)) \vee (l_1 \in EAdd(a) \wedge l_2 \in EDel(a))$ (véase la Figura 4.3(1) y 4.3 (2)), o cuando $(l_1 \in EAdd(a) \wedge l_2 \in SDel(a))$ (véase la Figura 4.3 (3)). En estos tres casos l_1 y l_2 formarán un mutex potencial y no serán válidos simultáneamente durante la ejecución de la acción a.

No obstante, el caso base no se cumple cuando $(l_1 \in SAdd(a) \wedge l_2 \in EDel(a))$ (véase la Figura 4.3(4)) ya que, en este caso, l_1 y l_2 podrían ser válidos simultáneamente a lo largo del intervalo de la acción (habitualmente, si un literal aparece como efecto negativo en una acción, dicho literal será requerido como precondición de la acción). En consecuencia, no hay ningún impedimento en que ambos literales sean válidos al mismo tiempo durante la ejecución de la acción (excepto al final de la misma).

Por otro lado, las cuatro situaciones que satisfacen el caso recurrente se muestran en la Figura 4.4, las cuales combinan $p \in SCond(a)$ o $p \in Inv(a)$ con $l_1 \in SAdd(a)$ o $l_1 \in EAdd(a)$. Por lo tanto, la segunda condición de *mutex potencial* que se muestra en el trabajo de Gerevini *et al.* [44] tiene una aplicación directa en nuestro modelo temporal.

La segunda etapa para determinar una relación de *mutex permanentes* consiste en verificar los *mutex potenciales*. Un *mutex potencial* entre l_1 y l_2 deja de ser una relación de exclusión mutua si:

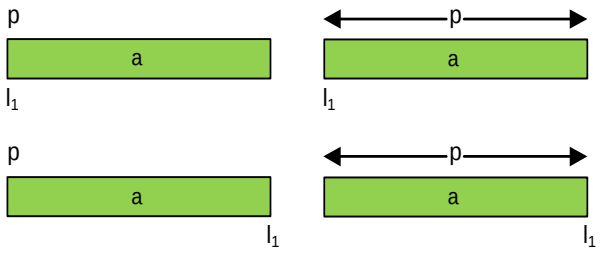

Figura 4.4: *Mutex* potential cuando l_2 es *mutex* potencial con la precondición p y l_1 es un efecto positivo

- Condición 1: existe una acción a que contiene a l_1 y l_2 entre sus efectos positivos. Esta condición se satisface en el modelo temporal cuando los dos literales l_1 y l_2 se añaden en el mismo punto de tiempo, es decir $(l_1 \in SAdd(a) \wedge l_2 \in SAdd(a)) \vee (l_1 \in EAdd(a) \wedge l_2 \in EAdd(a))$. Sin embargo, cuando l_1 y l_2 se añaden en puntos de tiempo diferentes, `at start` y `at end`, respectivamente, (o viceversa), no podemos garantizar que esta condición sea suficiente para que el *mutex potencial* desaparezca ya que otra acción podría borrar el efecto añadido `at start` antes de añadir el efecto `at end`.

- Condición 2: existe una acción a donde l_2 es un efecto positivo de la acción a, l_1 no es un efecto negativo de la acción a ni tampoco es *mutex potencial* con ninguna precondición de a. Esta condición se satisface en el modelo temporal ya que es independiente del momento en que l_2 se produce. La condición 2 establece que el literal l_1 puede ser válido en un estado en el que a es aplicable por lo que l_1 y l_2 persisten en el estado donde l_2 se genera.

Todo *mutex potencial* que no deje de serlo tras realizar la verificación anterior, se convierte en un *mutex permenente*.

4.3. Modelo de Landmarks Temporales

En un problema de planificación temporal con restricciones se puede extraer información del mismo para calificar los literales con distintos intervalos temporales. Dado un literal l que se extrae de un problema $\mathcal{P}$ se puede obtener la siguiente

información: el intervalo que acota el instante en el que l comenzará a ser cierto o válido, el intervalo en el que l es *necesario* y el intervalo en el que l será finalmente válido en el plan de solución. Los literales que deben ser ciertos en todos los planes solución de duración menor o igual a $T_{\mathcal{P}}$ se denominan *landmarks temporales*. En las siguientes secciones se caracterizan los *landmarks temporales* y las relaciones de orden y temporales entre dichos *landmarks*, lo que constituye el llamado grafo de *landmarks temporales*.

La información contenida en un grafo de *landmarks temporales* representa los planes solución para un problema de planificación temporal con restricciones $\mathcal{P}$ para el que se define un *deadline* $T_{\mathcal{P}}$. Los nodos de este grafo denotan *landmarks temporales*, los cuales se deducen a partir de la información de $\mathcal{P}$. Sobre los *landmarks temporales* debe tenerse en cuenta que:

1. los *landmarks temporales* se definen en el grafo mediante un conjunto de intervalos que se actualizan a medida que se infiere nueva información sobre las restricciones que deben satisfacer los planes solución de $\mathcal{P}$.

2. un *landmark temporal* del grafo se corresponde finalmente con un literal l de un plan solución Π para el problema $\mathcal{P}$, de modo que la ocurrencia temporal o intervalo de existencia de l en Π vendrá soportado por los intervalos que definen el *landmark temporal* en el grafo.

4.3.1. Definición de landmark temporal

Definición 4.7. *Dado un problema de planificación temporal con restricciones* $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ *donde se define un* deadline $T_{\mathcal{P}}$*, un literal* l *es un* ***landmark temporal*** *si* l *es cierto en todos los planes solución de duración menor o igual que el deadline* $T_{\mathcal{P}}$*.* ∎

Usaremos las funciones $start(l)$ y $end(l)$ para referirnos a los puntos de tiempo donde comienza y termina, respectivamente, una ocurrencia temporal o intervalo de existencia de un *landmark* l en un plan solución, es decir, el intervalo donde l es cierto en el plan solución. De este modo, l es un *landmark temporal* si para todos los planes solución Π del problema $\mathcal{P}$ cuya $dur_{\Pi} \leq T_{\mathcal{P}}$, se cumple $start(l) \leq end(l) \leq T_{\mathcal{P}}$.

4.3.1.1. Intervalos de un landmark temporal

A partir de un *deadline* o límite temporal $T_{\mathcal{P}}$ para los planes solución de un problema $\mathcal{P}$ y un *landmark temporal* l, se definen una serie de intervalos asociados a l en el grafo de *landmarks*:

- **intervalo de generación** ($[min_g(l), max_g(l)]$), donde $min_g(l)$ es el instante de tiempo más temprano donde l puede ocurrir en el plan y $max_g(l)$ representa el último instante de tiempo donde l debe ocurrir para obtener un plan solución de duración menor o igual que $T_{\mathcal{P}}$. El intervalo de generación representa el periodo de tiempo donde debe comenzar la ocurrencia temporal de l en los planes solución de duración menor o igual que $T_{\mathcal{P}}$. Se cumple que $min_g(l) \leq start(l) \leq max_g(l) \leq end(l)$. El valor de $min_g(l)$ viene determinado por el primer nivel de literal del $TRPG$ donde aparece l. Inicialmente, $max_g(l) = T_{\mathcal{P}}$.

- **intervalo de validez** ($[min_v(l), max_v(l)]$) este intervalo establece los límites inferior y superior dentro de los cuales se debe encontrar el intervalo de existencia de l en los planes solución de duración menor o igual a $T_{\mathcal{P}}$. Por tanto, se cumple que $min_v(l) \leq start(l) \leq end(l) \leq max_v(l)$. Inicialmente, $min_v(l) = min_g(l)$ y $max_v(l) = T_{\mathcal{P}}$.

- **intervalo de necesidad** ($[min_n(l), max_n(l)]$) representa el conjunto de puntos de tiempo donde l se requiere como condición de una acción que satisface otros *landmarks*. Obviamente, los puntos de tiempo donde l se necesita deben estar contenidos dentro del intervalo de existencia de l en el plan, por tanto se cumple $start(l) \leq min_n(l) \leq max_n(l) \leq end(l)$. Inicialmente, $min_n(l) = min_g(l)$ y $max_n(l) = T_{\mathcal{P}}$.

La Figura 4.5 muestra una representación gráfica de los tres intervalos de un *landmark temporal* en el grafo de *landmarks temporales*. Evidentemente, el intervalo de necesidad tiene que estar contenido en el intervalo de validez ya que $min_v(l) \leq start(l) \leq end(l) \leq max_v(l)$ y $start(l) \leq min_n(l) \leq max_n(l) \leq end(l)$, por tanto $min_v(l) \leq min_n(l) \leq max_n(l) \leq max_v(l)$. En la parte superior de la figura se puede observar la correspondencia entre los intervalos del *landmark* en el grafo y el intervalo de la ocurrencia temporal de dicho *landmark* en el plan.

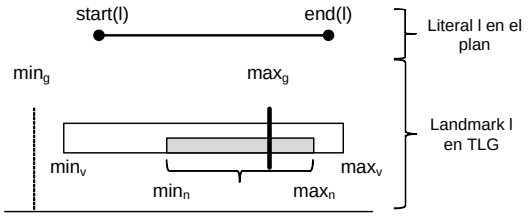

Figura 4.5: Representación de un landmark temporal

Supongamos que estamos interesados en obtener los *landmarks temporales* y sus intervalos para el problema del dominio logistics de la Figura 4.1 con un *deadline* $T_{\mathcal{P}} = 31$. Este proceso se realiza a partir de la información del problema, como se verá en la Sección 4.4, pero en este ejemplo se realizará a partir de la información contenida en los planes para ver la correspondencia entre la información veraz representada por el intervalo de existencia de un literal en el plan y la información deducida, la cual está reflejada en los intervalos del *landmark temporal* correspondiente en el grafo. En la Figura 4.6 se muestran los planes Plan1 y Plan2 que resuelven este problema para el *deadline* $T_{\mathcal{P}} = 31$. En Plan1, plane1 alcanza city0 en el instante 15 y el intervalo de existencia del literal (at plane1 city0) en Plan1 es [15, 17], es decir $start$(at plane1 city0) $= 15$ y end(at plane1 city0) $= 17$. El instante $start$(at plane1 city0) se produce cuando termina la acción (fly plane1 C2 C0) ya que (at plane1 city0) es un efecto $EAdd$ de esta acción; por otro lado, el instante end(at plane1 city0) se genera cuando comienza la acción (fly plane1 C0 C2) ya que (at plane1 city0) es un efecto $SDel$ de esta acción. En Plan2, plane1 alcanza city0 en el instante 17 y, por tanto, el intervalo de existencia del literal (at plane1 city0) en Plan2 es [17, 19]. Observando los planes que resuelven el problema para un *deadline* $T_{\mathcal{P}} = 31$ se puede deducir que es obligatorio obtener plane1 en city0 en el instante $t \leq 17$. Por lo tanto, el instante max_g(at plane1 city0) es 17 ya que es lo más tarde que (at plane1 city0) puede comenzar para obtener un plan Π de duración $dur_{\Pi} \leq 31$. Por otro lado, el instante min_v(at plane1 city0) $= 15$ y max_v(at plane1 city0) $= 19$ para un plan Π de duración $dur_{\Pi} \leq 31$ ya que lo más pronto que el literal (at plane1 city0) puede existir en un plan (Plan1 en este caso) es 15 y lo más tarde que el literal (at plane1 city0) puede existir en un plan (en este caso Plan2) es 19.

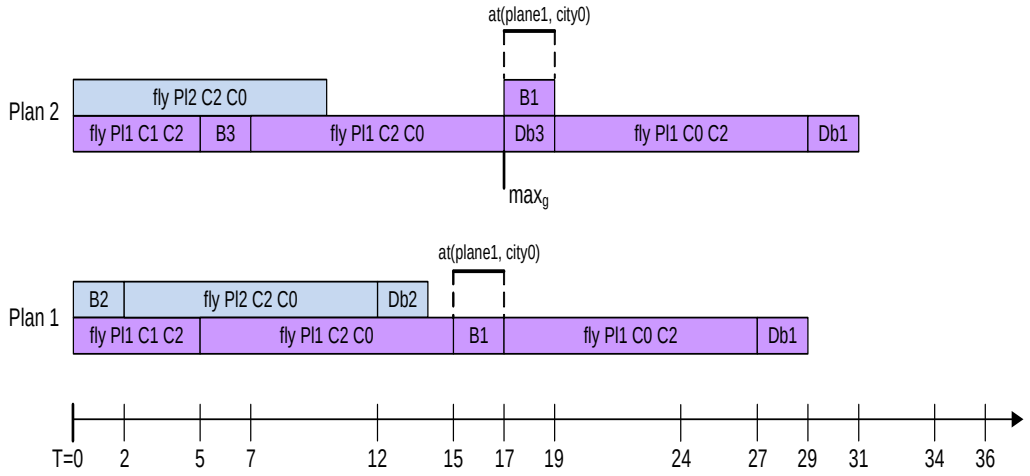

Figura 4.6: Planes temporales el problema del dominio *logística*.

4.3.1.2. Consistencia de un landmark temporal

La información reflejada en los intervalos de un *landmark temporal* del grafo deben cumplir una serie de relaciones para que el *landmark* sea consistente:

- $min_v(l) \leq max_v(l) \leq T_{\mathcal{P}}$
- $min_g(l) \leq max_g(l)$
- $min_n(l) \leq max_n(l)$
- $min_g(l) \leq min_v(l)$
- $max_g(l) \in [min_v(l), max_v(l)]$
- $[min_n(l), max_n(l)] \subseteq [min_v(l), max_v(l)]$

La relación $min_g(l) \leq min_v(l)$ se cumple siempre ya que $min_g(l)$ es el instante de tiempo más temprano donde l puede empezar en el plan, y viene determinado por el instante de tiempo del primer nivel de literal del $TRPG$ donde aparece l. Aunque $min_v(l)$ se inicializa a $min_g(l)$, el intervalo de validez de l se puede restringir porque $min_v(l)$ estará condicionado por los intervalos de los *landmarks* anteriores a l. Algo similar ocurre con los instantes $max_g(l)$ y $max_v(l)$; en este caso se restringen los

Deadline	En $\mathcal{D}$
(within t l)	(t, l)
(always-within t l_i l_j)	$(max_g(l_i) + t, l_j)$
(sometime-before l_i l_j)	$(max_g(l_i), l_j)$
(sometime-after l_i l_j)	$(max_g(l_j), l_i)$

Tabla 4.1: Representación de *deadlines* en $\mathcal{D}$

límites derechos del intervalo de generación y validez, respectivamente, debido a las relaciones existentes con los *landmarks* posteriores. La actualización o propagación de la información temporal de estos intervalos se verá en la Sección 4.4.2.

Cuando alguna de estas relaciones entre los límites de los intervalos de un *landmarks* no se cumplen, significa que es posible añadir más información (como ocurre cuando las relaciones del intervalo de necesidad no están comprendidas en el intervalo de validez) o que el problema no tiene solución. En la Sección 4.4.3 se verá con detalle el análisis de las inconsistencias que se producen en los *landmarks temporales*.

4.3.2. Representación de *deadlines* en un modelo de landmarks temporales

En nuestro modelo se pueden manejar todas las restricciones temporales de PDDL3.0 que representan un *deadline* mediante los operadores modales within, always-within, sometime-before y sometime-after. En esta sección se presenta la traducción interna de las restricciones expresadas mediante los operadores de PDDL3.0 al formato (t, l) del conjunto de restricciones $\mathcal{D}$ de un problema $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$.

La Tabla 4.1 muestra la representación interna de los *deadlines* que se pueden expresar con los diferentes operadores modales. El operador within tiene una traducción directa al formato (t, l). La restricción (always-within t l_i l_j) denota que l_j debe conseguirse antes de haber transcurrido t unidades de tiempo a partir de $max_g(l_i)$. Es importante destacar que esta restricción sólo se incorpora a $\mathcal{D}$ si l_i es un *landmark* ya que la restricción indica que se debe conseguir l_j antes de t unidades de tiempo a partir del instante en el que se ha conseguido l_i, lo cual implica que l_j sólo debe conseguirse si l_i está presente en el plan. Por último, las restricciones (sometime-before l_i l_j) y (sometime-after l_i l_j) se definen en términos de $max_g(l_i)$ y

$max_g(l_j)$, respectivamente, para incorporarlas a $\mathcal{D}$. Igualmente, estas restricciones sólo se incorporan si l_i es un *landmark* ya que la semántica de las mismas sólo exige conseguir l_j cuando l_i está en el plan.

4.3.3. Relaciones de orden entre landmarks temporales

En esta sección se definen el conjunto de relaciones de orden entre *landmarks temporales* como una extensión de las relaciones de orden de los *landmarks* STRIPS presentadas en la Sección 3.3.1.

Definición 4.8. *Existe una* ***relación de dependencia*** *entre dos landmarks temporales l_i y l_j ($l_i \prec_d l_j$) si para todo plan temporal Π que consigue l_j en el instante t' ($l_j \in S_{t'}$) desde el estado $\mathcal{I}$ existe al menos un estado S_t anterior a $S_{t'}$ que contiene l_i. Formalmente,* $\forall \Pi \subseteq \mathcal{A}^* : \mathcal{I} \rightarrow_\Pi S_{t'} \wedge l_j \in S_{t'} \Rightarrow \exists S_t / t \in [0, t' - \varepsilon] \wedge l_i \in S_t$. ∎

Básicamente, la idea intuitiva de una relación de dependencia es que para conseguir el *landmark* l_j en el instante t', se debe conseguir l_i en un instante $t < t'$.

Definición 4.9. *Dados dos landmarks temporales l_i y l_j tales que $l_i \in S_t$, $l_j \in S_{t'}$ y $l_i \prec_d l_j$, se dice que existe una* ***relación de necesidad*** *entre l_i y l_j ($l_i \prec_n l_j$) si para todo plan Π que consigue l_j en el instante t', Π genera un estado S_t desde el cual se puede alcanzar S'_t con una única acción a tal que $l_i \in Cond(a) \wedge l_j \in AddEff(a)$. Formalmente:* $\forall \Pi \subseteq \mathcal{A}^* : \mathcal{I} \rightarrow_\Pi S_{t'} \wedge l_j \in S_{t'} \Rightarrow \exists S_t / t \in [0, t' - \varepsilon] \wedge l_i \in S_t \wedge \exists a / Cond(a) \subseteq S_t \wedge l_i \in Cond(a) \wedge l_j \in AddEff(a)$. ∎

Pueden existir diferentes secuencias de acciones para conseguir l_j en el instante t' ($l_j \in S_{t'}$) a partir del estado S_t ($l_i \in S_t$) y no todas ellas ser de longitud 1. La Definición 4.9 establece que para que exista una relación de necesidad entre l_i y l_j ($l_i \prec_n l_j$) debe existir una secuencia que contenga una única acción entre S_t y S'_t. En caso que la secuencia tuviera más de una acción entonces podríamos encontrar un estado $S_{t''}$ entre S_t y $S_{t'}$ donde se cumpliría que $l' \in S_{t''}$, $l_i \prec_n l'$ y $l' \prec_n l_j$.

Deadline	Orden
(within t l)	-
(always-within t l_i l_j)	$l_i \preceq_d l_j$
(sometime-before l_i l_j)	$l_j \preceq_d l_i$
(sometime-after l_i l_j)	$l_i \preceq_d l_j$

Tabla 4.2: Restricciones de orden de las restricciones PDDL3.0

Obsérvese que $l_i \prec_n l_j$ es un caso particular de $l_i \prec_d l_j$. La idea intuitiva de una relación de necesidad es que para conseguir l_j en un estado $S_{t'}$, es obligatorio conseguir l_i en un estado S_t inmediatamente anterior al estado $S_{t'}$.

Es importante destacar que la relación $l_i \prec_n l_j$ implica que debe cumplirse $min_v(l_i) < min_v(l_j)$; esto no impide que la relación entre los límites superiores de sus intervalos de validez satisfagan la relación $max_v(l_i) > max_v(l_j)$.

Por otro lado, las restricciones temporales que se expresan mediante los operadores always-within, sometime-before y sometime-after conllevan las relaciones de orden que se muestran en la Tabla 4.2.

4.3.4. Grafo de Landmarks Temporales

A partir de los *landmarks* extraídos para un problema se construye el grafo de *landmarks* que representa un esqueleto de los planes solución que satisfacen las restricciones $\mathcal{D}$ del problema. Como ya se ha comentado, en un problema $\mathcal{P}$ podemos distinguir dos tipos de *landmarks*: *landmarks* STRIPS, aquellos que se obtienen sin tener en cuenta las restricciones temporales en $\mathcal{D}$ y *landmarks temporales*, aquellos que se obtienen para satisfacer las restricciones de $\mathcal{D}$. Los *landmarks temporales* y STRIPS se combinan en una estructura denominada **Grafo de Landmarks Temporales**[3].

Definición 4.10. *Dado un problema de planificación con restricciones* $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$, *un* ***grafo de landmarks temporales (TLG)*** *se define como* $TLG =$

[3]Aunque este grafo contiene también *landmarks* STRIPS se denomina grafo de *landmarks temporales* porque los *landmarks* STRIPS se caracterizan asimismo con los intervalos comentados en la Sección 4.3.1.1. De este modo, se considera que el grafo está compuesto únicamente por *landmarks temporales*.

(N_T, E_T)[4] *donde* N_T *es el conjunto de landmarks de* $\mathcal{P}$ *y* E_T *es el conjunto de órdenes establecidos entre estos landmarks, es decir,* $l_i \prec_n l_j$ *o* $l_i \prec_d l_j$*, indicando que* l_i *debe ocurrir antes que* l_j *en el plan solución.* ∎

Los *landmarks* de este grafo vendrán definidos por los *intervalos temporales* comentados en la Sección 4.3.1.1.

4.4. Construcción del Grafo de Landmarks Temporales

En esta sección mostramos en detalle el proceso de construcción del grafo de *landmarks temporales* donde se representa la información temporal del problema de planificación. La información del grafo puede ser muy útil a la hora de determinar si el problema es irresoluble o no. En el caso de que el problema tenga solución, la información contenida en el grafo servirá de ayuda en el proceso de búsqueda de un plan solución.

El algoritmo 4.1 muestra los principales pasos que se llevan a cabo en el proceso de construcción del TLG:

- La operación indicada en la línea 1 del Algoritmo 4.1 se encarga de **extraer los *landmarks* STRIPS y los *landmarks temporales*** junto con sus órdenes. También se caracteriza cada *landmark* con sus intervalos temporales. En la Sección 4.4.1 se muestran los detalles de este paso.

- En la línea 3 se actualizan los distintos intervalos temporales de los landmarks en función de las relaciones de orden y de exclusión mutua existentes entre ellos. Los detalles de este **proceso de propagación de la información temporal** se verán en la Sección 4.4.2.

- En la línea 4 del Algoritmo 4.1 se **analizan las inconsistencias temporales** en un *landmark* a partir de las relaciones entre los límites de sus intervalos. La Sección 4.4.3 muestra los detalles de este análisis.

[4]Se utiliza la nomenclatura N_T y E_T para distinguir los nodos y enlaces del grafo de *landmarks temporales* (TLG) de los nodos y enlaces del grafo de *landmarks* STRIPS (LG).

Algoritmo 4.1 Algoritmo para calcular el Grafo de Landmarks Temporales

1: Extracción de landmarks (STRIPS y temporales) y sus órdenes
2: **while** Exista información para actualizar **do**
3: Propagar la información temporal
4: Análisis de las inconsistencia temporales
5: **if** Se encuentra una inconsistencia irresoluble **then**
6: **return** Problema sin solución
7: **end if**
8: **end while**
9: **return** TLG

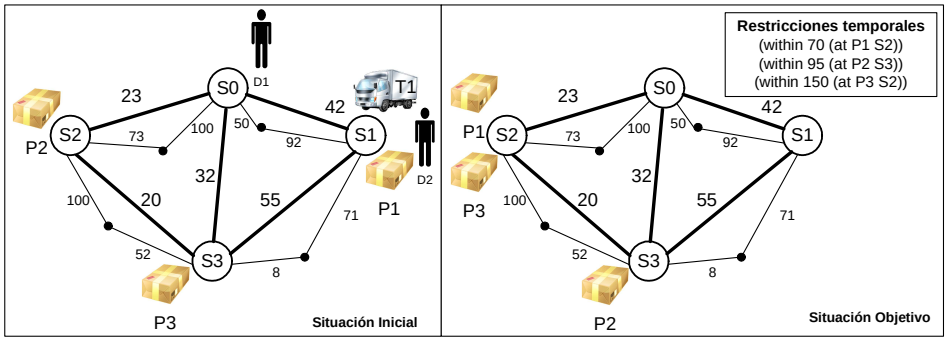

Figura 4.7: Situación inicial y objetivos para el ejemplo utilizado para la construcción del TLG.

- De acuerdo al análisis de inconsistencias del paso anterior, el algoritmo puede devolver que el problema es irresoluble (líneas 5 y 6 del algoritmo).

- En caso de que el problema tenga solución, se devuelve el *TLG* construido (línea 9 del algoritmo).

Para ilustrar la construcción del *TLG* se utilizará un problema del dominio driverlog [73]. La parte izquierda de la Figura 4.7 muestra la situación inicial y en la parte derecha de la Figura 4.7 se muestran los objetivos y las restricciones temporales sobre dichos objetivos. El objetivo de este dominio es repartir paquetes que se transportan utilizando camiones. Los camiones se conducen a través de una red de carreteras (líneas en negrita Figura 4.7). Existe también una serie de conductores que son necesarios para conducir los camiones y que se pueden desplazar a través de caminos (líneas normales de la Figura 4.7).

En las siguientes secciones se explican detalladamente los diferentes pasos del

proceso de construcción del TLG.

4.4.1. Paso 1: Extracción de landmarks y órdenes

En esta sección se analiza la incorporación de los dos tipos de *landmarks* que se pueden extraer de un problema $\mathcal{P}$ en el TLG: *landmarks* STRIPS, literales que se cumplen para cualquier plan solución sin tener en cuenta las restricciones temporales especificadas en $\mathcal{D}$; y *landmarks temporales*, literales que deben cumplirse en un plan solución que satisfaga las restricciones en $\mathcal{D}$. Sea $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ un problema de planificación temporal y sea $\mathcal{P}' = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ un problema de planificación temporal con restricciones tal que el conjunto de acciones $\mathcal{A}$, la situación inicial $\mathcal{I}$ y los objetivos $\mathcal{G}$ son los mismos para los dos problemas. El conjunto de *landmarks* STRIPS para el problema de planificación $\mathcal{P}$ es, por definición, la intersección de todos los literales conseguidos en la ejecución de todos los planes solución del problema $\mathcal{P}$ [61]. Intuitivamente, para el problema $\mathcal{P}'$ tendremos el mismo o un número menor de planes solución que para el problema $\mathcal{P}$ debido a la existencia de las restricciones $\mathcal{D}$, lo cual permite encontrar nuevos *landmarks* para $\mathcal{P}'$.

4.4.1.1. Paso 1.1: extracción de landmarks STRIPS y órdenes

En el capítulo anterior se presentó la aproximación FULL (ver Sección 3.5), resultado de la combinación de distintos métodos de extracción de *landmarks* STRIPS que permite explotar los puntos fuertes de cada técnica a la vez que mitigar los puntos más débiles. Para la obtención del conjunto de *landmarks STRIPS* de nuestro modelo temporal se ha utilizado la técnica FULL ya que es la técnica que consigue extraer un mayor número de *landmarks*, *landmarks disyuntivos* y relaciones de orden que el resto de aproximaciones de la Sección 3.4, tal y como muestran los resultados de la Sección 3.5.

La aproximación FULL devuelve un grafo de landmarks (LG), $LG = (N, E)$ donde N está formado por *landmarks* individuales y disyuntivos y $E = \{(l_i \prec_{\{\mathtt{n},\mathtt{d}\}} l_j)$[5]: $l_i, l_j \in N\}$. Denominaremos SL al conjunto de *landmarks* individuales de N y DL al conjunto de *landmarks* disyuntivos de N. La información del grafo LG se utiliza para inicializar $TLG = (N_T, E_T)$ de la siguiente forma:

[5]Se utiliza esta nomenclatura para representar a $l_i \prec_n l_j$ o $l_i \prec_d l_j$.

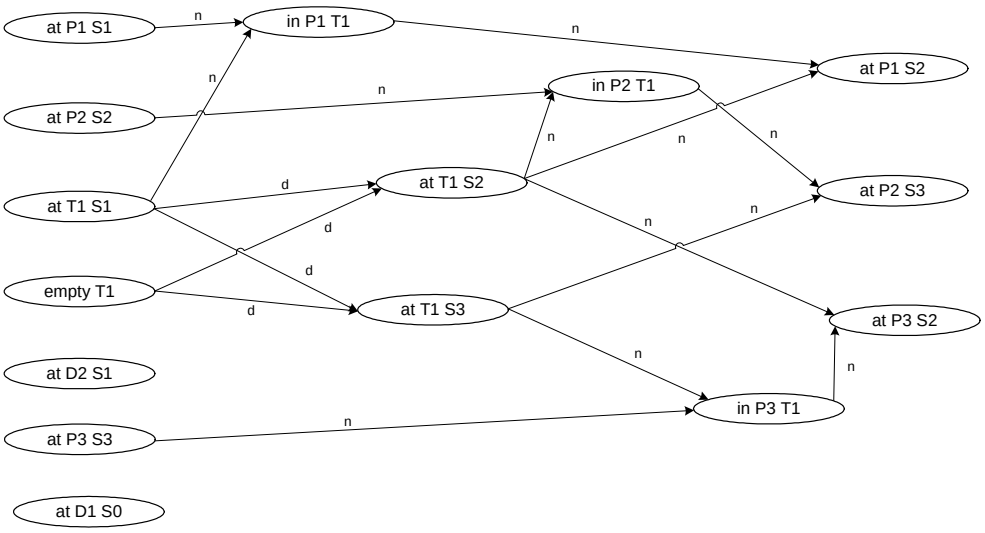

Figura 4.8: *Landmarks* STRIPS del TLG correspondiente al ejemplo 4.7.

- $N_T = SL$ de LG y
- $E_T = \begin{cases} (l_i \prec_{\{\mathtt{n},\mathtt{d}\}} l_j) : l_i, l_j \in SL \cup \\ (l_i \prec_{\mathtt{d}} l_j) : \exists l_k \in DL : (l_i \prec_{\{\mathtt{n},\mathtt{d}\}} l_k) \wedge (l_k \prec_{\{\mathtt{n},\mathtt{d}\}} l_j) \end{cases}$

La Figura 4.8 muestra el grafo de *landmarks temporales* (TLG) que se obtiene en este paso con los *landmarks* STRIPS y sus órdenes donde los arcos se etiquetan con el tipo de enlace que representan.

4.4.1.2. Paso 1.2: extracción de landmarks temporales y órdenes

Para la extracción de los *landmarks temporales* necesitamos:

1. extender la definición de fa_{LM} (ver Definición 3.6) al contexto temporal. El conjunto resultante se denominará *temporal first achievers*.

2. redefinir el concepto de *literal dependency labels* (Definición 3.11) utilizando para ello el $TRPG$ en lugar del RPG; esta nueva definición dará lugar a los *temporal literal dependency labels*. No es necesario redefinir el concepto de *action dependency labels* al contexto temporal porque no se consideran en la extracción de *landmarks temporales*.

Definición 4.11. *Se denomina* ***temporal first achievers*** *de un literal l al conjunto $TRPGA_t$ formado por las acciones del nivel t que tienen l como efecto positivo. Usaremos la función $fa^t(l)$ para referirnos a los temporal first achievers del landmark l en el instante t: $fa^t(l) = \{a \in TRPGA_t : l \in SAdd(a) \cup EAdd(a)\}$* ∎

Definición 4.12. *Los* ***temporal literal dependency labels*** *de un literal l en el instante t es el conjunto de los literales necesario para conseguir l en el instante t y se denota como $lit\_labels_l^t(l)$:*

$$lit\_labels_l^t(l) = \{l\} \cup \left(\bigcap_{\forall a \in fa^t(l)} lit\_labels_a^t(a) \right)$$

donde

$$lit\_labels_a^t(a) = \left(\bigcup_{\forall c \in Cond(a)} lit\_labels_l^{t-dur(a)}(c) \right)$$

∎

Al igual que ocurre en la Definición 3.11 el cálculo de los *temporal literal dependency labels* de un literal l es una función recursiva que finaliza cuando $l \in \mathcal{I}$ ya que $fa^t(l) = \emptyset$ cuando se alcanza fa^0. La función $lit\_labels_a^t(a)$ representa el conjunto de literales que son necesarios para aplicar la acción a.

El Algoritmo 4.2 muestra el proceso para extraer un conjunto de *landmarks temporales* (TL) y sus órdenes y añadirlos al TLG. La entrada de este algoritmo es el TLG calculado en la Sección 4.4.1.1 y las restricciones $\mathcal{D}$ definidas en el problema. La primera parte del Algoritmo 4.2 (líneas 4 a la 7) consiste en inicializar los intervalos temporales de los landmarks en N_T, los cuales son los *landmarks* STRIPS que provienen de la aproximación FULL.

Dado un conjunto de restricciones $\mathcal{D}$, donde $(t, l) \in \mathcal{D}$, existen dos métodos complementarios que se aplican de forma secuencial para calcular *landmarks temporales*.

- El **primer método** consiste en extraer los *landmarks temporales* directamente de las restricciones de $\mathcal{D}$ (línea 9 del Algoritmo 4.2), añadiendo dichos *landmarks temporales* al conjunto TL y actualizando el límite superior del

Algoritmo 4.2 Algoritmo para la extracción de landmarks temporales y órdenes

1: **Input:** $TLG = (N_T, E_T)$ y $\mathcal{D}$
2: $TL = \emptyset$
3: {Inicialización de los intervalos de los landmarks}
4: **for** $l \in N_T$ **do**
5: $\quad Set\ min_g(l)$ al primer nivel del TRPG donde l aparece
6: $\quad Set\ min_v(l) = min_n(l) = min_g(l)$
7: $\quad Set\ max_g(l) = max_v(l) = max_n(l) = T_{\mathcal{P}}$
8: **end for**
9: {Extracción de los landmarks temporales mediante las restricciones}
10: **for** $(t, l) \in \mathcal{D}$ **do**
11: $\quad TL = TL \cup \{l\}$
12: $\quad Set\ max_g(l) \leq t$
13: **end for**
14: {Extracción de los landmarks temporales mediante los temporal literal dependency labels}
15: **for** $l \in N_T \cup TL$ **do**
16: $\quad TL = TL \cup lit\_labels_l^{max_g(l)}(l)$
17: **end for**
18: {Calculo de los órdenes para los nuevos landmarks temporales}
19: **for** $l \in TL$ **do**
20: $\quad t = max_g(l)$
21: $\quad E_T = E_T \cup \{(l_i \prec_{\mathtt{n}} l) : l_i \in \bigcap_{\forall a \in fa^t(l)} Cond(a)\}$
22: $\quad E_T = E_T \cup \{(l_i \prec_{\mathtt{d}} l) : l_i \in labels^t(l)\}$
23: **end for**
24: **return** $TLG = (N_T, E_T)$, donde $N_T = N_T \cup TL$

intervalo de generación de cada *landmark*. Formalmente, $\forall (t, l) \in \mathcal{D} : TL = TL \cup \{l\},\ max_g(l) = t$. Por ejemplo, uno de los objetivos de la Figura 4.7 es (within 70 (at P1 S2)) donde l =(at P1 S2) es un *landmark* con $max_g(l)$ =70. Las líneas 10 a la 13 del Algoritmo 4.2 extraen los *landmarks temporales* de $\mathcal{D}$ y los añaden al conjunto TL.

- El **segundo método** se utiliza como base el conjunto de acciones que permiten conseguir un literal en el instante de tiempo definido por su *deadline*. Para ello se utilizan los *temporal literal dependency labels* de los literales (línea 14 Algoritmo 4.2). Dado un *landmark* $l \in N_T \cup TL$, $lit\_labels_l^t(l)$ en $t = max_g(l)$ es el conjunto de literales que se deben conseguir para obtener un

plan que alcance l, como muy tarde, en el instante $max_g(l)$. Formalmente, $\forall l \in N_T \cup TL : TL = TL \cup lit\_labels_l^{max_g(l)}(l)$. En el ejemplo de la Figura 4.7), el conjunto $lit\_labels_l^t$(at P1 S2) en $t = 70$ contiene los literales (driving D2 T1) y (at T1 S0) ya que son condiciones de la acción (Drive D2 T1 S0 S2) que es la única que consigue (at P1 S2) en $t = 70$. Así, estos literales se convertirán en *landmarks temporales* (nodos resaltados en negrita en la Figura 4.9). Este paso se repite mientras se encuentren nuevos *landmarks temporales* (líneas de la 15 a la 17 en el Algoritmo 4.2).

Después de extraer el conjunto de *landmarks temporales*, el siguiente paso consiste en **establecer los órdenes** entre ellos (líneas 18 a la 23 del Algoritmo 4.2). Existe un orden necesario entre l_i y l ($l_i \prec_n l$) si $l_i \in \bigcap_{\forall a \in fa^t(l)} Cond(a)$ para el instante $t = max_g(l)$. Por otra parte, existe un orden dependiente entre l_i y l ($l_i \prec_d l$) si $l_i \in lit\_labels_l^t(l)$ para el instante $t = max_g(l)$.

Por ejemplo, en el TLG de la Figura 4.9, hay dos órdenes necesarios ((driving D2 T1) $\prec_n$ (at T1 S0)) y ((at T1 S1) $\prec_n$ (at T1 S0)) porque la única forma de conseguir (at T1 S0) en $max_g = 45$ es usando la acción `(Drive T1 S1 S0 D2)` cuyas condiciones se satisfacen con los dos literales (driving D2 T1) y (at T1 S1). Por otro lado, hay un orden dependiente ((at T1 S2) $\prec_d$ (at T1 S3)) porque (at T1 S2) pertenece a los *temporal literal dependency labels* de (at T1 S3) en $t = 93$.

La definición de órdenes entre *landmarks* puede provocar la aparición de ciclos, tanto explícitos como implícitos, durante la construcción del TLG. La aparición de un ciclo indica que al menos un *landmarks* l del ciclo debe conseguirse más de una vez. Se pueden dar los siguientes casos:

- Si $l \in \mathcal{I}$ indica que l se eliminará del plan en algún momento y luego es necesario conseguirlo de nuevo para satisfacer los objetivos.

- Si $l \in \mathcal{G}$ indica que l será necesario para conseguir otro objetivo y que, al haber sido borrado en el plan, será necesario volver a conseguirlo dado que $l \in \mathcal{G}$.

- Si l es un *landmark* cualquiera indica que l es necesario para conseguir un objetivo, más tarde se elimina y será entonces necesario volver a conseguirlo para satisfacer otro objetivo.

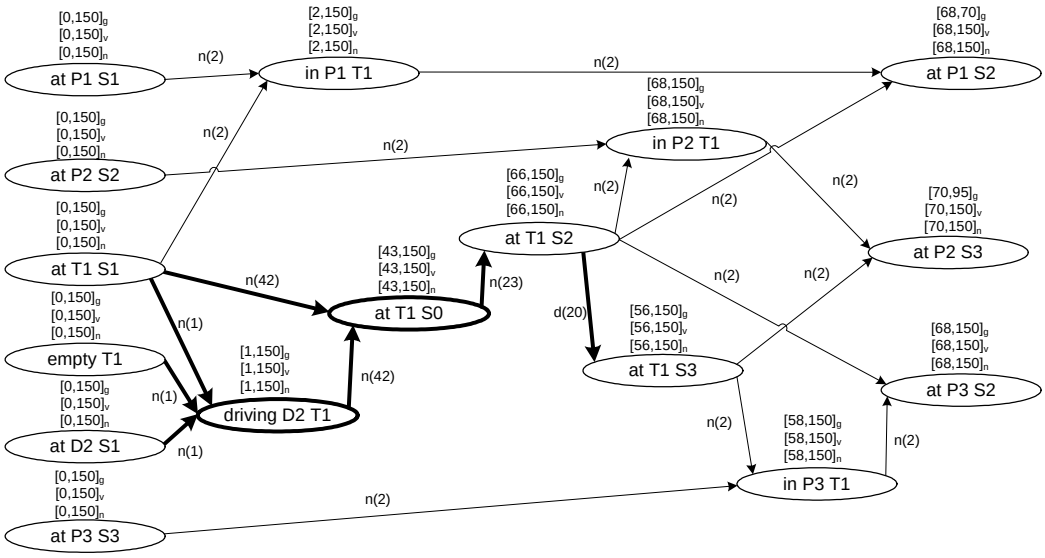

Figura 4.9: TLG para el ejemplo de la Figura 4.7

Un ciclo explícito, es decir, aquel que aparece explícitamente en el TLG, se produce por la forma en la que se extraen los enlaces durante la construcción del TLG. Aunque la aparición de ciclos explícitos no es frecuente, estos se resuelven creando una nueva instancia, l', del *landmark* l. El nuevo *landmark* l' tendrá los mismos intervalos que l pero sus relaciones de orden serán distintas ya que se elimina el enlace que provoca el ciclo con l y se añade un nuevo enlace con l' [61]. Por otro lado, un ciclo implícito es aquel que no se detecta a priori en el TLG. Este tipo de ciclos aparece por necesidades competitivas entre varios *landmarks* . En nuestra aproximación se detectan como una inconsistencia cuando se calculan los intervalos de necesidad en la Sección 4.4.2.3. En la Sección 4.4.3 se detalla como se resuelve una inconsistencia derivada a partir de un ciclo implícito.

Al final del Paso 4.4.1.2 se dispone del TLG, el cual se compone del conjunto de *landmarks temporales* y STRIPS, unidos por un conjunto de órdenes entre ellos y donde los intervalos temporales de todos los landmarks están inicializados. La Figura 4.9 muestra el TLG que se obtiene para el ejemplo de la Figura 4.7 donde los arcos entre dos *landmarks* se etiquetan con el tipo de enlace entre dichos *landmarks* (n si se trata de $l_i \prec_n l_j$, o d si se trata de $l_i \prec_d l_j$), y entre paréntesis se muestra un valor numérico que representa una distancia temporal entre esos *landmarks* (en la Sección 4.4.2.1 se muestra con detalle como se calculan estas distancias).

4.4.2. Paso 2: Propagación de la información temporal

Una relación de orden entre dos *landmarks* l_i y l_j ($l_i \prec_d l_j$ o $l_i \prec_n l_j$) determina que l_i pertenece a un estado S_t a partir del cual la aplicación de una acción (o una secuencia de acciones) da como resultado un nuevo estado $S_{t'}$ que contiene l_j. En otras palabras, una relación de orden implícitamente establece una relación temporal entre los dos *landmarks* tal que el estado S_t siempre debe ser anterior al estado $S_{t'}$. La existencia de relaciones de orden entre *landmarks* también permite establecer relaciones temporales entre los límites de los intervalos de los *landmarks*.

Como puede verse en el grafo de la Figura 4.9, donde l_i es el literal (in P1 T1) y l_j (at P1 S2), existe una relación de necesidad entre l_i y l_j ya que la acción (Unload P1 T1 S2) es la única que consigue el literal l_j desde un estado donde l_i es cierto. Teniendo en cuenta que el último instante en el que se puede generar l_j es 70 ($max_g(l_j) = 70$), y suponiendo que la duración de la acción (Unload P1 T1 S2) son 2 unidades de tiempo, el *landmark* l_i debe generarse en el instante 68 como muy tarde, es decir $max_g(l_i) \leq 68$. Por tanto, cuando existe una relación de orden entre dos literales, se pueden derivar ciertas relaciones entre los límites de sus intervalos temporales, las cuales deben satisfacerse siempre en un plan correcto.

Una vez construido el TLG, los intervalos de los *landmarks temporales* se actualizan y propagan a través de las relaciones de dependencia y necesidad existentes entre los *landmarks*. Este proceso se realiza en los siguientes pasos:

- En el primer paso se determina las **distancias que existen entre los límites de los intervalos** como resultado de las relaciones de orden entre dos literales. En la Sección 4.4.2.1 se explican los tipos de distancias y cómo se realizan los cálculos entre los límites de los intervalos.

- El siguiente paso consiste en realizar un análisis de las relaciones de orden y exclusión mutua entre literales lo que permite restringir los intervalos de los *landmarks* del TLG. Este proceso de **actualización de los intervalos temporales de los *landmarks*** se muestra en la Sección 4.4.2.2.

- En el último paso se **calculan los intervalos de necesidad** a partir del resto de intervalos que ya se han restringido en el paso anterior. En la Sección 4.4.2.3 se explican los detalles de este proceso.

El Algoritmo 4.3 muestra el proceso de actualización y propagación de los intervalos de generación, validez y necesidad de los *landmarks* del TLG a través de las relaciones de orden y exclusión mutua. En este algoritmo la propagación de las relaciones entre los intervalos se realiza mediante la pila *events* donde se añaden *eventos* que son pares $(l, point)$ y donde *point* es el límite del intervalo que se ha modificado del *landmark* l. La pila *events* se inicializa con los eventos (l, max_g) para los *landmarks* de los objetivos del problema y con los eventos (l, min_v) para los *landmarks* que pertenecen a la situación inicial (líneas 2 y 3 del Algoritmo 4.3). Mientras existan *eventos* se realizará el análisis de las relaciones de orden y exclusión mutua (líneas de la 6 a la 10 del Algoritmo 4.3) mediante la función *relationAnalysis* (Algoritmo 4.4). A continuación se realizara el cálculo del intervalo de necesidad (líneas 11 y 12 del Algoritmo 4.3) mediante la función *necessityComputation* (Algoritmo 4.4). Finalmente se realiza el análisis de las inconsistencias que se hayan detectado y se almacenan en la lista *inconsistentList* (líneas de la 13 a la 16 del Algoritmo 4.3) mediante la función *inconsistencyAnalysis* (Algoritmo 4.6). Cuando se detecta una inconsistencia entre los intervalos de un *landmark* que no se puede resolver, esto indica que el problema es irresoluble (líneas 15 y 16 del Algoritmo 4.3). Por otro lado, si la inconsistencia se puede resolver se realizan las acciones necesarias para su resolución (el análisis de estas inconsistencias se ve en la Sección 4.4.3). El Algoritmo 4.3 repite todas estas operaciones mientras se añadan nuevos *landmarks* (líneas 5 y 18 del Algoritmo 4.3). Finalmente, si no se ha detectado ninguna inconsistencia irresoluble, se devuelve el TLG (línea 19).

4.4.2.1. Distancias entre los límites de los intervalos temporales

Para actualizar los intervalos de los *landmarks* se emplea una estimación de la distancia[6] entre los límites de los intervalos de los *landmarks*. Suponiendo que existe una relación de orden entre l_i y l_j ($l_i \prec_d l_j$ o $l_i \prec_n l_j$), la forma general de la relación entre los límites de los intervalos es:

$$limite_{l_i} + distancia \leq limite_{l_j} \tag{4.4.1}$$

[6]Siendo estrictos matemáticamente deberíamos hablar de separación ya que una distancia no puede ser negativa y en este caso existe esa posibilidad como puede verse en la Tabla 4.3.

Algoritmo 4.3 Algoritmo para actualizar y propagar la información temporal en el TLG y analizar las inconsistencias entre los intervalos de un *landmark*

1: **Input:** $TLG = (N_T, E_T = \{(l_i \prec_{\{\mathtt{n},\mathtt{d}\}} l_j) : l_i, l_j \in N_T\})$
2: $Push(events, (l, max_g))$, $\forall l \in \mathcal{G}$
3: $Push(events, (l, min_v))$, $\forall l \in \mathcal{I}$
4: $inconsistentList = \emptyset$
5: **repeat**
6: {Análisis de relaciones de orden y de exclusión mutua}
7: **while** $events \neq \emptyset$ **do**
8: $event = Pop(events)$, donde $event = (l, point)$
9: $relationAnalysis(TLG, events, event, inconsistentList)$
10: **end while**
11: {Cálculo del intervalo de necesidad}
12: $necessityComputation(TLG, inconsistentList)$
13: {Análisis de inconsistencias}
14: $result = inconsistencyAnalysis(TLG, events, inconsistentList)$
15: **if** $result = FALSE$ **then**
16: **return** No hay solución
17: **end if**
18: **until** no se añaden nuevos *landmarks*
19: **return** TLG

que denota la necesidad de l_i para generar l_j. En PDDL2.1, l_i puede ser necesario at start, overall o at end y, por tanto, vamos a distinguir dos tipos de distancias[7], las cuales se calculan en función de los instantes temporales de las condiciones y efectos de las acciones que necesitan l_i y generan l_j, respectivamente, así como en función del número de acciones o secuencias de acciones que generan l_j a partir de l_i.

Teniendo en cuenta el instante temporal en el que se producen las condiciones y efectos de cada acción, se definen dos tipos de distancias entre una condición $l_i \in Cond(a)$ y un efecto $l_j \in Eff(a)$ de una acción durativa a:

1. $dis^a_E(l_i, l_j)$ es la distancia entre el punto de tiempo cuando l_j se añade y el primer punto de tiempo en el que se necesita l_i; y

2. $dis^a_L(l_i, l_j)$ es la distancia entre el punto de tiempo cuando l_j se añade y el último punto de tiempo donde se necesita l_i.

La Tabla 4.3 muestra los valores de $dis^a_E(l_i, l_j)$ y $dis^a_L(l_i, l_j)$ para todas las posibles combinaciones de condiciones y efectos.

[7]El caso at end se puede considerar un caso particular del caso overall.

Condición l_i	Efecto l_j	$dis^a_E(l_i, l_j)$	$dis^a_L(l_i, l_j)$
$SCond(a)$	$SAdd(a)$	ε	ε
$SCond(a)$	$EAdd(a)$	$dur(a)$	$dur(a)$
$Inv(a)$	$SAdd(a)$	ε	$-dur(a)$
$Inv(a)$	$EAdd(a)$	$dur(a)$	ε
$ECond(a)$	$SAdd(a)$	$-dur(a)$	$-dur(a)$
$ECond(a)$	$EAdd(a)$	ε	ε

Tabla 4.3: Definición de $dis^a_E(l_i, l_j)$ y $dis^a_L(l_i, l_j)$ para secuencias de una única acción a y donde a sólo puede tomar un valor

Por ejemplo, sea a la acción (Unload P1 T1 S2) cuya duración son 2 u.t., sea l_i el literal (in P1 T1) y sea l_j el literal (at P1 S2). El literal l_i es una condición at start y el literal l_j es un efecto at end, respectivamente, de la acción (Unload P1 T1 S2). Esto significa que el primer instante de tiempo donde l_i se necesita es al comienzo de la acción y l_j se añade al final de la misma. Por tanto, $dis^a_E(l_i, l_j)$ coincidirá con la duración de a, tal y como se muestra en la Tabla 4.3. Por otro lado, para calcular el valor de $dis^a_L(l_i, l_j)$ se tiene en cuenta que el último punto donde l_i se necesita es al comienzo de la acción[8] y, por tanto, $dis^a_L(l_i, l_j)$ coincide asimismo con la duración de a. Otra situación distinta se produce cuando se consideran las condiciones overall de la acción, como, por ejemplo, el literal (at T1 S2) que es una condición overall de la acción (Unload P1 T1 S2). En este caso, el primer punto de tiempo donde l_i se necesita es al comienzo de la acción, dando lugar a $dis^a_E(l_i, l_j) = dur(a)$, pero el último punto donde l_i se necesita es al final de la acción l_i por lo que $dis^a_L(l_i, l_j) = \varepsilon$.

La Tabla 4.3 muestra las distancias entre los límites de los intervalos de dos *landmarks* cuando sólo es necesario una secuencia compuesta de una única acción a para conseguir l_j a partir de l_i y a sólo puede tomar un valor, como es el caso del ejemplo anterior donde a sólo puede ser la acción (Unload P1 T1 S2). Sin embargo, en otras situaciones es posible encontrar distintas acciones a para la secuencia mono-acción que genera l_j a partir de l_i. En este caso se define dis_E y dis_L como dos casos generales de dis^a_E y dis^a_L, respectivamente; es decir, se tiene en cuenta todas las acciones a tal que $l_i \in Cond(a) \wedge l_j \in SAdd(a) \cup EAdd(a)$. El cálculo

[8]Siendo estrictos el último punto donde l_i se necesita sería al comienzo de la acción más ε pero se puede omitir porque ε es un valor tan pequeño que no influye en los cálculos

de $dis_{\{E,L\}}(l_i, l_j)$[9] que se muestra en la Ecuación 4.4.2 se utiliza para obtener la distancia entre dos landmarks l_i y l_j cuando existe una relación de orden necesario entre ellos ($l_i \prec_n l_j$).

$$\begin{gathered} dis_{\{E,L\}}(l_i, l_j) = \min_{\forall a \in \mathcal{A}} (dis^a_{\{E,L\}}(l_i, l_j)) \\ donde\ l_i \in Cond(a) \wedge l_j \in SAdd(a) \cup EAdd(a). \end{gathered} \tag{4.4.2}$$

Por otro lado, también puede darse el caso que la secuencia de acciones para conseguir l_j a partir de l_i esté compuesta de más de una acción. Esto es, existe una secuencia $A = \{a_1, a_2, ..., a_n\}$ para conseguir un efecto l_j desde una condición l_i tal que:

$$\begin{aligned} l_i \in Cond(a_1) \wedge l_1 \in Eff(a_1) \wedge l_1 \in Cond(a_2) \wedge l_2 \in Eff(a_2) \wedge ... \\ \wedge l_{n-1} \in Cond(a_n) \wedge l_j \in Eff(a_n) \end{aligned} \tag{4.4.3}$$

Se trata de una secuencia de acciones donde l_i es una condición de la primera acción de la secuencia, a_1; la acción a_i de la secuencia genera como efecto un literal que es condición de la siguiente acción a_{i+1}; finalmente, la última acción de la secuencia, a_n, genera el efecto l_j. Para este caso se define DIS_E y DIS_L como una generalización de dis_E y dis_L, respectivamente, que determinan la distancia entre li y l_j cuando existe una relación de orden dependiente entre los dos *landmarks* ($l_i \prec_d l_j$). A continuación se define el conjunto de secuencia de acciones entre dos literales l_i y l_j para poder calcular DIS_E y DIS_L.

Definición 4.13. *Se denomina* ***conjunto de secuencia de acciones*** *entre dos literales l_i y l_j, denotado como $A(l_i, l_j)$ a todas las secuencias de acciones del $TRPG$ $\{a_1, a_2, ..., a_n\}$ que permiten conseguir l_j desde l_i, es decir todas las secuencias que satisfacen la Ecuación 4.4.3.* ∎

Las distancias DIS_E y DIS_L se calculan de la siguiente forma:

$$DIS_{\{E,L\}}(l_i, l_j) = \begin{cases} 0 & : \ si\ l_i = l_j \\ \min\limits_{\substack{A_i \in A(l_i,l_j): \\ A_i = \{a_1, a_2, ..., a_n\}}} (DIS_{\{E,L\}}(l_i, l) + dis_{\{E,L\}}(l, l_j)) & : \ en\ otro\ caso \end{cases} \tag{4.4.4}$$

donde $l \in Cond(a_n)$.

[9] Se utiliza esta notación para representar dis_E o dis_L.

Por ejemplo, en la Figura 4.8 existe un orden dependiente entre (at T1 S1) y (at T1 S2). Si calculamos la secuencia de acciones entre estos dos *landmarks* tenemos que es necesario primero subir a un conductor al camión T1 y luego mover T1 entre S1 y S2. La duración de una acción de tipo Board (subir a un conductor a un camión) es de 1 u.t., mientras que para mover el camión entre S1 y S2 tenemos varias posibilidades. Por ejemplo, se puede mover T1 a través de S0 cuya duración sería 65 (42 de S1 a S0 y 23 de S0 a S2); o a través de S3 cuya duración sería 75 (55 de S1 a S3 y 20 de S3 a S2). Ante esta situación el valor DIS_E entre (at T1 S1) y (at T1 S2) sería 66 (1 por la acción de subir y 65 por el camino más rápido de la acción Drive para llegar de S1 a S2).

Por último hay que comentar que el cálculo de $DIS_{\{E,L\}}(l_i, l_j)$ coincide con el cálculo de $dis_{\{E,L\}}(l_i, l_j)$ cuando existe una relación $(l_i \prec_n l_j)$ ya que en este caso la secuencia de acciones $A(l_i, l_j)$estará formada por una única acción y, en este caso, $DIS_{\{E,L\}}(l_i, l_j) = DIS_{\{E,L\}}(l_i, l_i) + dis_{\{E,L\}}(l_i, l_j)$ de acuerdo con la Ecuación 4.4.4 lo que implica que $DIS_{\{E,L\}}(l_i, l_j) = 0 + dis_{\{E,L\}}(l_i, l_j)$. Hay que recordar que, de acuerdo con la Definición 4.9, cuando existe una relación de necesidad hay una única acción a que produce l_j en el estado $S_{t'}$ cuando existe l_i en el estado S_t tal que $l_i \in Cond(a) \wedge l_j \in Eff(a)$.

4.4.2.2. Actualización de los intervalos temporales

Como se ha comentado anteriormente, una relación de orden de la forma $l_i \prec_{\{d,n\}} l_j$ entre dos *landmarks* l_i y l_j establece implícitamente algunas relaciones entre los límites de sus intervalos. En esta sección se analizan las relaciones que se pueden establecer entre los límites de los intervalos de los *landmarks* y cómo utilizar estas relaciones para ajustar los valores de los límites de dichos intervalos. Los límites inferiores de los intervalos se restringen desplazando el valor del límite hacia delante en el tiempo y los límites superiores se restringen desplazando su valor hacia atrás en el tiempo, es decir, la actualización de un límite inferior del intervalo avanza en el tiempo y la actualización del límite superior de un intervalo retrocede en el tiempo.

Actualización del límite inferior del intervalo de validez (min_v). Para el caso del límite inferior del intervalo de validez se establece la siguiente relación:

$$min_v(l_j) = \max\left(min_v(l_j), min_v(l_i) + DIS_E(l_i, l_j)\right) \tag{4.4.5}$$

Esta relación indica que el límite $min_v(l_j)$ del intervalo de validez de l_j se actualiza

Algoritmo 4.4 Función *relationAnalysis*

```
1: Input: TLG = (N_T, E_T), events, event = (l, point), inconsistentList
2: if point == min_v then
3:    for e = (l ≺_{n,d} l_i) ∈ E_T do
4:       Set min_v(l_i) =max(min_v(l_i), min_v(l) + DIS_E(l, l_i))
5:       Push(events, (l_i, min_v))
6:    end for
7: end if
8: if point == max_g then
9:    for e = (l_i ≺_{n,d} l) ∈ E_T do
10:      Set max_g(l_i) =min(max_g(l_i), max_g(l) − DIS_E(l_i, l))
11:      if ∃t : l_i, l son mutex en t then
12:         Set max_v(l_i) =min(max_v(l_i), min_v(l), max_g(l) − DIS_L(l_i, l)
13:         Set max_g(l_i) =min(max_v(l_i), max_g(l_i))
14:         Push(events, (l_i, max_v))
15:      end if
16:      Push(events, (l_i, max_g))
17:   end for
18: end if
19: if point == max_v then
20:   for e = (l ≺_{n} l_i) ∈ E_T do
21:      if min_v(l_i) < max_v(l) + DIS_L(l, l_i) then
22:         Set max_g(l_i) =min(max_g(l_i), max_v(l) + DIS_L(l, l_i))
23:      else
24:         Add(inconsistentList, min_v(l_i) > max_v(l) + DIS_L(l, l_i))
25:      end if
26:   end for
27: end if
28: return TLG, inconsistentList
```

teniendo en cuenta cada *landmark* l_i con el que l_j tiene una relación $l_i \prec_{\{d,n\}} l_j$. El objetivo de esta relación es determinar el desplazamiento en el tiempo de $min_v(l_j)$, para lo cual se utiliza el valor de $min_v(l_i)$ y la distancia $DIS_E(li, lj)$. La actualización de la relación entre intervalos de validez se inicia con los *landmarks* del estado inicial

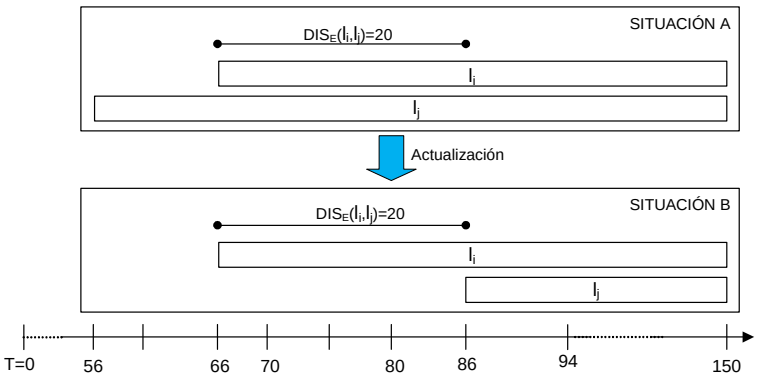

Figura 4.10: Propagación del min_v para el ejemplo 4.7

(línea 3 del Algoritmo 4.3) y se propaga hacia delante en el tiempo, afectando a todos aquellos *landmarks* que mantengan una relación de orden con uno de los *landmarks* modificados (líneas de la 2 a la 7 del Algoritmo 4.4). Por ejemplo, sea l_i el *landmark* (at T1 S2) y sea l_j el *landmark* (at T1 S3) de la Figura 4.9; se puede observar que entre l_i y l_j existe una relación de orden dependiente $l_i \prec_d l_j$ y que $DIS_E(l_i, l_j) = 20$. La Figura 4.10 Situación A muestra los intervalos iniciales de validez de l_i y l_j donde los límites de los intervalos no satisfacen la Relación 4.4.5 ya que $min_v(l_j)$ es anterior a $min_v(l_i) + DIS_E(l_i, l_j)$. Una vez actualizado el valor de $min_v(l_j)$ obtenemos el resultado que se muestra la Figura 4.10 Situación B donde $min_v(l_j) = 86$. Esta actualización se propaga posteriormente al *landmark* (at P2 S3), actualizando min_v(at P2 S3)=86. Este nuevo valor de min_v(at P2 S3) es más preciso que su valor inicial (min_v(at P2 S3)=min_g(at P2 S3)= 70), donde el valor inicial 70 proviene del primer nivel del $TRPG$ donde se consigue (at P2 S3).

Actualización del límite superior del intervalo de generación (max_g). Para el límite superior del intervalo de generación se establece la siguiente relación:

$$max_g(l_i) = \min\left(max_g(l_i), max_g(l_j) - DIS_E(l_i, l_j)\right) \tag{4.4.6}$$

Esta relación indica que el límite $max_g(l_i)$ del intervalo de generación se actualiza teniendo en cuenta cada *landmark* l_j con el que l_i tiene una relación $l_i \prec_{\{d,n\}} l_j$. El objetivo de esta relación es determinar el desplazamiento hacia atrás en el tiempo de $max_g(l_i)$, para lo cual se utiliza el valor de $max_g(l_j)$ y la distancia $DIS_E(li, lj)$. La actualización de la relación entre intervalos de generación de dos *landmarks* se

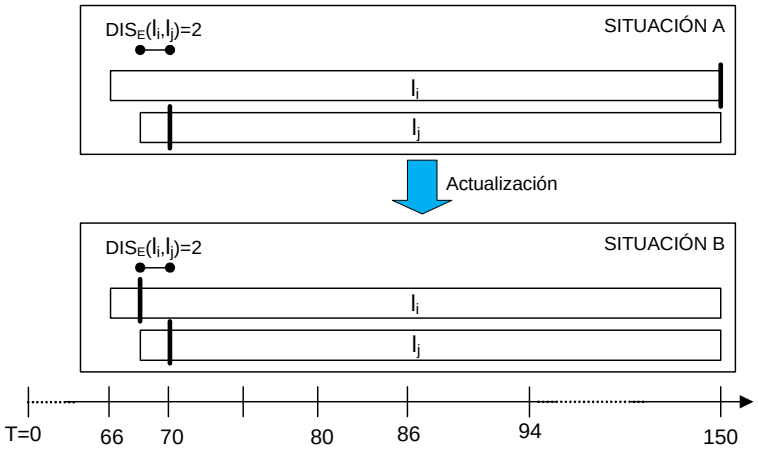

Figura 4.11: Propagación del max_g para el ejemplo 4.7

inicia con los *landmarks* de los objetivos del problema (línea 2 del Algoritmo 4.3) y se propaga hacia atrás en el tiempo, afectando a todos aquellos *landmarks* que mantengan una relación de orden con uno de los *landmarks* modificados (líneas de la 8 a la 10 y la 16 del Algoritmo 4.4). Por ejemplo, sea l_i el *landmark* (at T1 S2) y sea l_j el *landmark* (at P1 S2) de la Figura 4.9; se puede observar que entre l_i y l_j existe una relación de orden necesario $l_i \prec_n l_j$ y que $DIS_E(li, lj) = 2$. La Figura 4.11 Situación A muestra los intervalos de validez de l_i y l_j así como $max_g(l_i)$ y $max_g(l_j)$ (línea vertical más gruesa). La información de estos intervalos no es consistente con la Relación 4.4.6 ya que $max_g(l_i)$ es posterior a $max_g(l_j) - DIS_E(l_i, l_j)$. Teniendo en cuenta que $max_g(l_j) = 70$ se ha establecido por una restricción de $\mathcal{D}$, al propagar esta información en el TLG se actualiza el valor de $max_g(l_i)$ ya que el último punto de tiempo en el que se se puede conseguir l_i para que $max_g(l_j) = 70$ es el instante 68. La Figura 4.10 Situación B muestra el resultado de esta actualización.

Actualización por las relaciones de exclusión mutua. Cuando existe un *mutex permanente* entre l_i y l_j ($mutex(l_i, l_j)$) las relaciones que se pueden establecer son las siguientes:

$$\begin{aligned} max_v(l_i) =& \min(max_v(l_i), min_v(l_j), max_g(l_j) - DIS_L(l_i, l_j)) \\ max_g(l_i) =& \min(max_v(l_i), max_g(l_i)) \end{aligned} \tag{4.4.7}$$

Estas relaciones permiten actualizar el valor del límite superior del intervalo de validez de l_i aunque este caso es distinto de los anteriores. Para restringir $max_v(l_i)$ no

es suficiente con la existencia de una relación de orden $l_i \prec_{\{d,n\}} l_j$ ya que esta relación no implica que se pueda restringir el valor de $max_v(l_i)$ en función de $max_v(l_j)$. Por ejemplo, sea l_i =(at T1 S2) y l_j =(in P2 T1) de la Figura 4.9; existe una relación de orden necesario entre estos dos *landmarks*, la cual indica que para tener el paquete P2 dentro del camión T1 es necesario que el camión se encuentre en la posición S2 pero esta relación no restringe max_v(at T1 S2) por lo que se desconoce el instante en el que el camión T1 dejará de estar en la posición S2. Para poder restringir el valor del límite superior del intervalo de validez de l_i es necesario que se cumpla la relación $mutex(l_i, l_j)$ que implica que l_i y l_j no pueden coexistir al mismo tiempo y, por tanto, l_j puede restringir a l_i o viceversa. La existencia de $mutex(l_i, l_j)$ es una condición necesaria pero no suficiente. Para poder aplicar la Relación 4.4.7 es necesario que exista adicionalmente una relación de orden para saber si l_j restringe a l_i o l_i restringe a l_j ya que ambas situaciones pueden ser posibles a priori. Por tanto, es necesario que exista una relación $mutex(l_i, l_j)$ y una relación de orden $l_i \prec_{\{d,n\}} l_j$ para poder aplicar la Relación 4.4.7. Para actualizar el valor de $max_v(l_i)$ se aplica el principio de *menor compromiso* tomando como referencia el último instante donde l_j debe ocurrir ($max_g(l_j)$) puesto que a partir de este punto se puede asegurar que l_j ya se ha conseguido. También se tiene en cuenta el valor $min_v(l_j)$ para garantizar que l_i y l_j no coexisten simultáneamente. En la línea 12 del Algoritmo 4.4 se muestra la actualización de $max_v(l_i)$ a partir de los valores de los intervalos de l_j. Cuando se actualiza el valor $max_v(l_i)$ también se puede actualizar el valor $max_g(l_i)$ para satisfacer la condición $max_g(l_i) \leq max_v(l_i)$ (línea 13 del Algoritmo 4.4). Esta información se propaga al resto del grafo de modo que una modificación en $max_v(l_i)$ afecta a $max_g(l)$ si $l_i \prec_{\{n,d\}} l$ ya que l no se podrá conseguir en un instante de tiempo posterior a $max_v(l_i)$ más la distancia entre ellos (líneas de la 19 a la 27 del Algoritmo 4.4). Por ejemplo, teniendo en cuenta que l_i =(at T1 S1) y l_j =(at T1 S0) son *mutex* y que hay una relación de orden necesario entre ellos, actualizamos max_v(at T1 S1)=3, según la línea 12 del Algoritmo 4.4. Adicionalmente, como existe una relación de orden (at T1 S1)$\prec_n$(in P1 T1), se actualiza max_g(in P1 T1) a 3 ya que $dis_L^a = \varepsilon$ cuando a =`Load(P1 S1 T1)`, de acuerdo con la línea 22 del Algoritmo 4.4.

Otra situación que puede ocurrir es la siguiente. Sea l_i =(at T1 S2) y l_j =(at T1 S3) son *mutex* y existe un orden entre ellos. Cuando max_v(at T1 S2) se actualiza

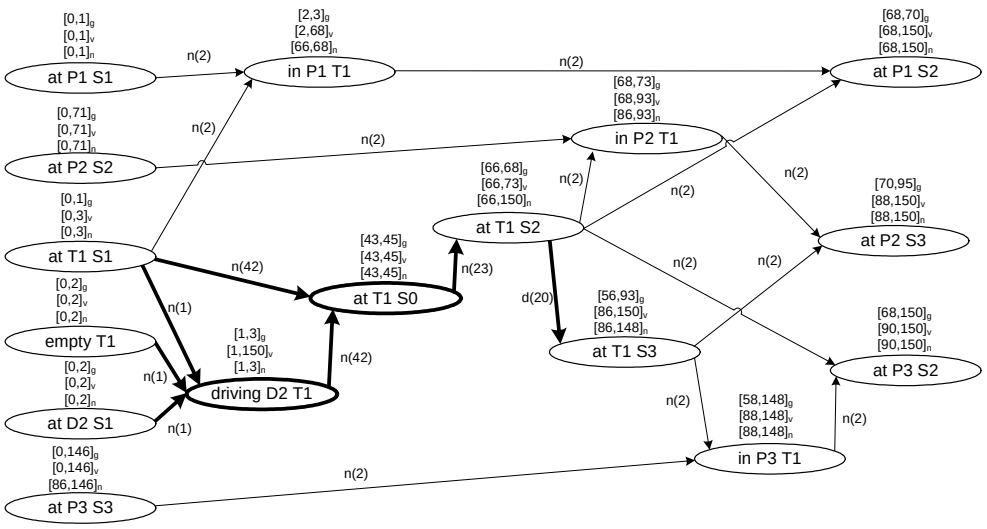

Figura 4.12: TLG con los intervalos de los landmarks actualizados para el ejemplo 4.7

al valor 73, indicado así que (at T1 S2) no será válido de 73 en adelante, esto provoca una inconsistencia ya que (at T1 S2) se necesita para conseguir (at P3 S2) en el intervalo [90, 50]. En la línea 21 del Algoritmo 4.4 se detecta la inconsistencia min_v(at P3 S2)> max_v(at T1 S2)+ε[10] que se añade a la lista de inconsistencias (*inconsistentList*) en la línea 24 (en la Sección 4.4.3 se describe cómo se resuelve esta inconsistencia).

4.4.2.3. Cálculo de los intervalos de necesidad

El intervalo de necesidad de un *landmark* l_i se calcula teniendo en cuenta todos los órdenes necesarios $l_i \prec_n l_j$ tal que l_i es una condición para generar l_j. El Algoritmo 4.5 describe el cálculo de los intervalos de necesidad, que se realiza siguiendo las siguientes fórmulas:

$$min_n(l_i) = \min_{\forall l_j : \exists l_i \prec_n l_j} (min_v(l_j) - DIS_E(l_i, l_j)) \quad (4.4.8)$$

$$max_n(l_i) = \max_{\forall l_j : \exists l_i \prec_n l_j} (max_g(l_j) - DIS_L(l_i, l_j)) \quad (4.4.9)$$

El comienzo del intervalo necesidad de l_i se actualiza con el primer punto de

[10] DIS_L((at T1 S2),(at P3 S2))=ε ya que la acción que se utiliza es (Unload P3 T1 S2) puesto que existe una relación (at T1 S2)$\prec_n$(at P3 S2)

Algoritmo 4.5 Función *necessityComputation*

1: **Input:** $TLG = (N_T, E_T)$, *inconsistentList*
2: **for** $l_i \in N_T$ **do**
3: $min_n(l_i) = \min_{\forall l_j : \exists l_i \prec_n l_j} (min_v(l_j) - DIS_E(l_i, l_j))$
4: $max_n(l_i) = \min(max_v(l_i), \max_{\forall l_j : \exists l_i \prec_n l_j} (max_g(l_j) - DIS_L(l_i, l_j)))$
5: **if** $[min_n(l), max_n(l)] \not\subseteq [min_v(l), max_v(l)]$ **then**
6: $Add(inconsistentList, [min_n(l), max_n(l)] \not\subseteq [min_v(l), max_v(l)])$
7: **end if**
8: **end for**
9: **return** TLG, *inconsistentList*

tiempo en el que l_i se necesita para garantizar la validez temporal ($min_v(l_j) - DIS_E(l_i, l_j)$) de cada *landmark* l_j. El valor de $max_n(l_i)$ se establece en el último punto de tiempo en el que l_i es necesario para asegurar la generación temporal ($max_g(l_j) - DIS_L(l_i, l_j)$) de cada *landmark* l_j. En las líneas 3 y 4 del Algoritmo 4.5 se realiza el cálculo del intervalo de necesidad para todos los *landmarks* del TLG. Al realizar este cálculo se puede detectar una inconsistencia si el intervalo de validez del *landmark* l_i no puede satisfacer las necesidades de todos los *landmarks* que necesiten a l_i, es decir, $[min_n(l_i), max_n(l_i)] \not\subseteq [min_v(l_i), max_v(l_i)]$. Esta inconsistencia se añade a la lista *inconsistentList* para que se analice en la Sección 4.4.3.

Por ejemplo, (at T1 S2)$\prec_n$(at P2 S3) y (at T1 S2)$\prec_n$(at P1 S2), lo que implica que min_n(at T1 S2) será igual a 66 (min(90-2, 68-2)[11]) y max_n(at T1 S2) será igual a 150 (max(150-0, 70-0)) de acuerdo a los valores de los intervalos que se muestran en la Figura 4.12. En esta figura también puede observarse la inconsistencia que existe en el *landmark* (at T1 S2) ya que su intervalo de necesidad no está contenido en su intervalo de validez. Esta inconsistencia se añade a la lista *inconsistentList* de acuerdo con la línea 6 del Algoritmo 4.5.

La Figura 4.12 muestra el TLG que se obtiene tras realizar la propagación de la información temporal (paso 2 del algoritmo de construcción del TLG). Los intervalos de los *landmarks* del TLG se han actualizado con la información temporal existente y sólo resta comprobar si se ha detectado e introducido alguna inconsistencia en la lista *inconsistentList*. En la siguiente sección se realiza el análisis de las inconsistencias que se hayan detectado.

[11] En ambos casos se trata de una acción Unload cuya duración es 2.

4.4.3. Paso 3: Análisis de las inconsistencias temporales

El objetivo de esta sección consiste en identificar las inconsistencias que se puedan resolver. Si se detecta alguna inconsistencia irresoluble, el Algoritmo 4.6 devolverá que el problema no tiene solución. En nuestro modelo de *landmarks temporales* podemos detectar que un problema no tiene solución en dos momentos: (1) durante la construcción del $TRPG$ o (2) cuando se detecta una inconsistencia irresoluble.

Durante la construcción del $TRPG$, puede darse el caso de que $fa^t(g) = \emptyset$ para un objetivo g con un *deadline* t. Esto significa que ninguna acción consigue g en t y, por lo tanto, se puede afirmar que el problema es irresoluble. Por ejemplo, si se definiese la restricción (within 65 (at P1 S2)) para el problema de la Figura 4.7, el problema sería irresoluble ya que el primer instante en el que se puede obtener el objetivo (at P1 S2) es 68 (min_g(at P1 S2)=68) y la restricción establecida indica que debe conseguirse, como muy tarde, en 65 ($fa^{65}(g) = \emptyset$ ya que no hay ninguna acción que consiga g =(at P1 S2) en $t = 65$).

La segunda situación en la que se identifica un problema irresoluble es cuando se produce una inconsistencia en los intervalos temporales de un *landmark* que no se puede resolver. Al actualizar y propagar la información temporal en el TLG, se pueden encontrar situaciones en las que un *landmark* no es consistente con las relaciones de la Sección 4.3.1.2. Se distinguen tres tipos de inconsistencias en el TLG:

1. Un *landmark* l se necesita más allá de su intervalo de validez: $[min_n(l), max_n(l)] \not\subseteq [min_v(l), max_v(l)]$.

2. $max_g(l)$ de un *landmark* l no está contenido en su intervalo de validez: $max_g(l) \notin [min_v(l), max_v(l)]$.

3. El intervalo temporal de un *landmark* l está vacío: $min_g(l) > max_g(l)$, $min_v(l) > max_v(l)$ o $min_n(l) > max_n(l)$.

Los dos últimos casos reflejan una **inconsistencia irresoluble**, esto es, un problema sin solución, mientras que el primer caso es una indicación de la existencia de un ciclo implícito por necesidades competitivas entre dos *landmarks* del TLG

Algoritmo 4.6 Función *inconsistencyAnalysis*

```
1: Input: TLG = (N_T, E_T), events, inconsistentList
2: while inconsistentList ≠ ∅ do
3:   if ∃l : [min_n(l), max_n(l)] ⊈ [min_v(l), max_v(l)] ∈ inconsistentList then
4:     addCopy(TLG, l, ∅, events)
5:     return TRUE
6:   else
7:     inconsistencyCheck()
8:     if inconsistentList ≠ ∅ then
9:       return FALSE
10:    end if
11:  end if
12: end while
13: return TRUE
```

(**inconsistencia resoluble**). En concreto, la inconsistencia del caso 1 se puede resolver añadiendo otra instancia del *landmark* l, que denominaremos l', para ajustar el fragmento de $[min_n(l), max_n(l)]$ que no es compatible con $[min_v(l), max_v(l)]$. El Algoritmo 4.6 procesa los elementos de la lista *inconsistentList* (línea 2 del Algoritmo 4.6), selecciona una inconsistencia resoluble (inconsistencia tipo 1), si la hay, y procede a la resolución de la misma mediante la función *addCopy* (Algoritmo 4.7). Tras resolver la inconsistencia, el Algoritmo 4.6 devuelve $TRUE$ y propaga la información del nuevo landmark l' y sus relaciones de orden en el TLG de acuerdo a los pasos indicados en la Sección 4.4.2. Tras estos pasos se vuelve a realizar el análisis de inconsistencias y se selecciona de nuevo una inconsistencia del tipo 1.

Si no quedan inconsistencias resolubles en la lista *inconsistentList*, se comprueban si el resto de inconsistencias de la lista se han eliminado como consecuencia de los cambios producidos en el TLG por la resolución de una inconsistencia resoluble (tipo 1). Si tras realizar esta comprobación queda alguna inconsistencia en *incosistentList* entonces el problema no tiene solución y la línea 9 del Algoritmo 4.6 devuelve $FALSE$ para indicar esta situación.

Por ejemplo, durante la actualización y propagación de la información del TLG de acuerdo a la Sección 4.4.2.2 se añade la inconsistencia min_v(at P3 S2)$> max_v$(at T1 S2)$+\varepsilon$ (a la que llamaremos *inconsistencia1*) y durante el cálculo del intervalo de necesidad se añade la inconsistencia $[min_n(l), max_n(l)] \not\subseteq [min_v(l), max_v(l)]$ donde

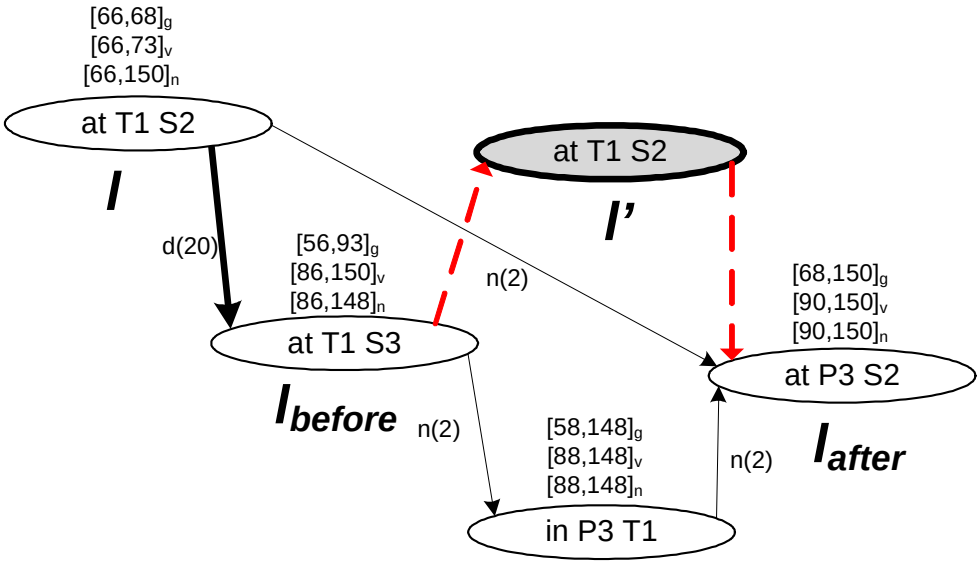

Figura 4.13: Extracto de la Figura 4.12 con los *landmarks* involucrados para resolver la *inconsistencia2 TLG*

l =(at T1 S2) (a la que llamaremos *inconsistencia2*), siendo la primera una inconsistencia irresoluble y la segunda resoluble. Se selecciona por tanto la *inconsistencia2* y se crea una copia (l') del *landmark* l =(at T1 S2) (línea 2 del Algoritmo 4.7).Una vez creado l', se inicializan sus intervalos temporales (líneas 3 y 4 del Algoritmo 4.7) y se añade al TLG (línea 5). A continuación se analizan los *landmarks* con los cuales l mantiene alguna dependencia, l_{after} y l_{before}, y que permitirán establecer las relaciones de orden con el *landmark* l'. La Figura 4.13 es un extracto de la Figura 4.12 donde se muestran los *landmarks* involucrados en la *inconsistencia2* y su resolución. De este modo, l_{after} es el *landmark* (at P3 S2), que mantiene una relación de necesidad con l y cuyo intervalo de necesidad no se satisface por la inconsistencia (línea 7 del Algoritmo 4.7); y l_{before} es el *landmark* (at T1 S3) que es *mutex* con l, $l \prec_d l_{before}$ y además l_{before} es anterior a l_{after} (línea 9 del Algoritmo 4.7). A continuación se añaden las relaciones de orden $l_{before} \prec_d l'$ y $l' \prec_n l_{after}$ (líneas rojas discontinuas de la Figura 4.13) y se elimina la relación de orden $l \prec_n l_{after}$ (línea 10 del Algoritmo 4.7). Por último, se añaden los eventos necesarios para que, posteriormente, se realicen las actualizaciones y propagaciones necesarias (líneas 11 y 12 del Algoritmo 4.7).

Algoritmo 4.7 Función *addCopy*

1: **Input:** $TLG = (N_T, E_T), l, l_{after}, events$
2: $l' = Copy(l)$
3: Set $min_v(l') = min_g(l') = min_n(l') = min_g(l)$
4: Set $max_v(l') = max_g(l') = max_n(l') = T_\Pi$
5: $N_T = N_T \cup \{l'\}$
6: **if** $l_{after} = \emptyset$ **then**
7: $\quad$ *Set* $l_{after} = (l_i \in N_T : l \prec_n l_i \in E_T \wedge max_g(l_i) = max_n(l) + DIS_L(l, l_i))$
8: **end if**
9: *Set* $l_{before} = (l_j \in N_T : l \prec_{\{n,d\}} l_j \in E_T \wedge l_j$ anterior $l_{after} \in E_T \wedge l, l_j$ son *mutex*)
10: $E = E \cup \{(l_{before}, l', \mathtt{d}), (l', l_{after}, \mathtt{n})\} - \{(l, l_{after}, \mathtt{n})\}$
11: $Push(events, (l_{after}, max_g))$
12: $Push(events, (l_{before}, min_v))$
13: **return** $TLG, events$

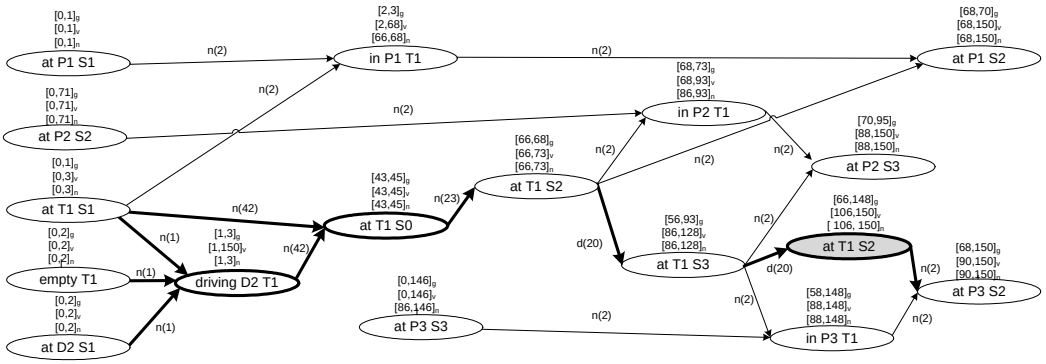

Figura 4.14: TLG final para el ejemplo 4.7

La Figura 4.14 muestra como se resuelve la *inconsistencia2* donde l' es el *landmark* sombreado que se añade al TLG y el resto de intervalos se actualizan en consecuencia. Tras resolver la *inconsistencia2* se procesa la *inconsistencia1* (irresoluble) y se comprueba previamente si ha sido eliminada como consecuencia de la resolución de la *inconsistencia2*. En este caso se puede observar que efectivamente ha sido así porque la relación de orden (at T1 S2)$\prec_n$(at P3 S2) se ha eliminado y se ha creado una nueva relación con la copia de (at T1 S2) cuyos intervalos de necesidad permiten la generación de (at P3 S2).

Una inconsistencia irresoluble se produciría si, por ejemplo, en el problema se hubiera definido (within 80 (at P2 S3)) en lugar de (within 95 (at P2 S3)). En este caso,

el intervalo de validez de (at P2 S3) sería [88,150] y se detectaría una inconsistencia con su intervalo de generación que sería $[70, 80]$. En concreto, la restricción límite para max_g(at P2 S3) sería 80 y este valor no está dentro de su intervalo de validez, por lo que se detecta una inconsistencia que no se puede resolver. Es importante destacar que esta inconsistencia no se detecta durante la construcción $TRPG$ porque el literal (at P2 S3) aparece por primera vez en el nivel t=70 del $TRPG$ (min_g((at P2 S3))=70). Por lo tanto, el $TRPG$ no detectaría inconsistencias en restricciones cuyo *deadline* es superior a 70. Sin embargo, el proceso de construcción del TLG detectaría que cualquier restricción cuyo *deadline* es inferior a 88 (min_v) no se puede satisfacer y el problema sería entonces irresoluble.

La Figura 4.14 muestra el estado final del TLG que se obtiene tras eliminar las inconsistencias existentes y que será utilizado durante la búsqueda de un plan solución.

En resumen, en nuestro modelo de *landmarks temporales* podemos detectar que un problema no tiene solución:

- Durante la construcción del $TRPG$ cuando $fa^t(g) = \emptyset$ para un objetivo g con un *deadline* t.
- Durante la propagación de la información temporal en el TLG cuando se detecta una inconsistencia entre intervalos que no se puede resolver.

4.5. Conclusiones

En este capítulo se ha presentado la construcción del grafo de *landmarks temporales* a partir del LG obtenido por la aproximación FULL del capítulo 3. Para la construcción de TLG se han utilizado los *landmarks temporales* caracterizados por un conjunto de intervalos que se van actualizando (restringiendo) mediante un proceso de propagación de la información temporal a partir de las relaciones de orden y de exclusión mutua. Se ha mostrado los dos tipos de inconsistencias que se detectan en el TLG: las resolubles, las cuales se pueden resolver y actualizan la información del grafo, y las irresolubles, que son una indicación de que el problema no tiene solución. Finalmente, el TLG construido se utilizará en el proceso de búsqueda para encontrar un plan solución que cumpla las restricciones temporales de $\mathcal{D}$.

Capítulo 5

TempLM: Planificador basado en *Landmarks Temporales*

5.1. Introducción

En este capítulo se presenta un planificador temporal independiente del dominio para problemas de planificación con restricciones, denominado TempLM. La construcción de TempLM utiliza, como punto de partida, las contribuciones y resultados iniciales presentados en el trabajo publicado en la conferencia *European Conference on Artificial Intelligence* (ECAI-08) [74], donde se establece la base del modelo de *landmarks temporales* y se comprueba la consistencia del modelo temporal mediante un CSP para detectar problemas irresolubles de planificación temporal con restricciones. En la Sección 5.2 se muestran los detalles de este trabajo preliminar. A partir de esta contribución se crea la primera versión TempLM, la cual realiza una búsqueda en un espacio de planes guiada por una función heurística resultado de adaptar la heurística h^{LM-cut} [56] a un contexto temporal. Esta primera versión de TempLM utiliza el TLG descrito en el Capítulo 4 para eliminar aquellos nodos del espacio de búsqueda que no cumplen los requisitos indicados en el TLG [76]. Posteriormente, se crea una segunda versión de TempLM donde se utiliza la información del plan de un nodo de árbol de búsqueda para actualizar su correspondiente TLG y, asimismo,

la información del TLG se utiliza para guiar el proceso de búsqueda. Esta nueva incorporación se denomina *feedback* ya que hace referencia a la retroalimentación existente entre la información de los nodos del árbol de búsqueda y sus grafos de *landmarks temporales* [77].

La Figura 5.1 muestra el esquema básico de TempLM. La parte sombreada muestra los pasos de la construcción del TLG, donde se identifican dos puntos en los que se puede detectar la irresolubilidad de un problema: cuando se construye el $TRPG$ o durante el proceso de propagación del TLG (como se ha visto en la Sección 4.4.3). La fase de búsqueda comienza cuando cuando finaliza la construcción del TLG. También es posible detectar la irresolubilidad de un problema durante el proceso de construcción del plan solución si todos los nodos del árbol de búsqueda resultan inconsistentes; es decir, si la información del plan contenida en cada nodo del árbol de búsqueda es inconsistente con la información de su TLG. Asimismo, las acciones contenidas en un nodo del árbol de búsqueda se emplean para actualizar sus correspondientes $TLGs$, añadiendo así nuevos *landmarks* y modificando los intervalos de los *landmarks* existentes (proceso de *feedback*). En la Sección 5.3 se muestran los detalles del proceso de búsqueda:

- En la Sección 5.3.1 se explica la adaptación de la heurística h^{LM-cut} a entornos de planificación temporal.
- En la Sección 5.3.2 se explica el proceso de generación de nodos sucesores.
- En la Sección 5.3.2.3 se explica la eliminación de un nodo del espacio de búsqueda que no es compatible con la información de su TLG.
- Por último, en la Sección 5.4 se muestra el proceso de *feedback* para enriquecer el TLG con la información del árbol de búsqueda y, a su vez, cómo incorporar el TLG actualizado en el propio proceso de búsqueda.

5.2. Modelo preliminar de TempLM: aproximación CSP

En la aproximación inicial de TempLM se establece la base del modelo de *landmarks temporales* y se comprueba la consistencia del modelo temporal mediante un resolutor de CSP para detectar problemas irresolubles de planificación temporal

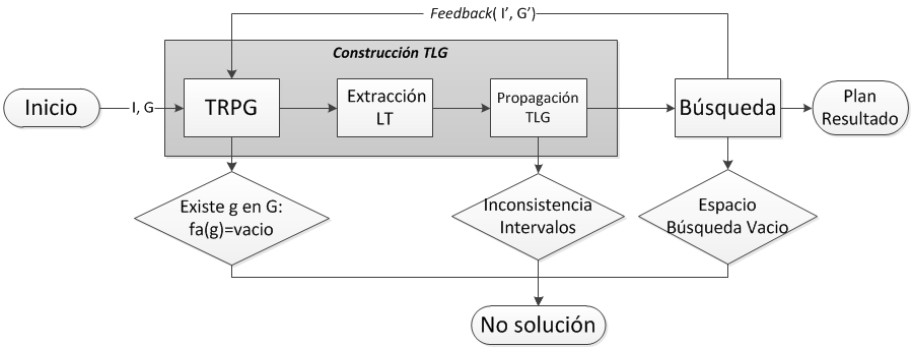

Figura 5.1: Esquema de TempLM

con restricciones [74]. El grafo de *landmarks temporales* que se propone en este este trabajo es similar al presentado en el Capítulo 4, con la salvedad que se utiliza una agenda para anotar las restricciones. En dicha agenda se anotan como restricciones las relaciones de consistencia de un *landmark temporal* (Sección 4.3.1.2), las relaciones de orden entre *landmarks temporales* (Sección 4.3.3), las relaciones que surgen de la propagación (Sección 4.4.2) y las restricciones definidas por el usuario (Sección 4.3.2). Por ejemplo, si entre dos *landmarks* l_i y l_j del TLG, existe una relación de orden $l_i \prec_{\{d,n\}} l_j$ se añade la restricción $min_v(l_i) < min_v(l_j)$ en la agenda. Por otro lado, si existe además una relación de exclusión mutua entre l_i y l_j $(mutex(l_i, l_j))$, se añade entonces la restricción $max_v(l_i) < min_v(l_j)$ a la agenda. En cambio, si $(mutex(l_i, l_j))$ y no se cumple $l_i \prec_{\{d,n\}} l_j$, se añade a la agenda la disyunción de restricciones $max_v(l_i) < min_v(l_j) \vee max_v(l_j) < min_v(l_i)$. En esta primera aproximación de TempLM, se construyen tantos CSPs como sean necesarios por la combinación de las restricciones disyuntivas y posteriormente se utiliza un resolutor de CSP para comprobar la consistencia de los CSPs construidos. Si alguno de los CSPs construidos es consistente, el problema puede tener solución y si todos son inconsistentes el problema se etiqueta como irresoluble. Por ejemplo, si hay una restricción disyuntiva, se crean dos CSPs cada uno con una restricción de la disyunción; si al menos uno de los dos CSPs es consistente entonces el problema puede ser resoluble; en caso contrario, el problema se etiqueta como irresoluble. Esta aproximación inicial de TempLM marcó el inicio de una prometedora línea de investigación para la detección de la irresolubilidad en problemas de planificación temporal. Sin embargo, el elevado coste de construcción y comprobación de los CSPs unido al

hecho de que sólo permite determinar si un problema es irresoluble o no, pero sin poder obtener el plan solución, la convierten en una aproximación muy limitada.

5.3. Proceso de búsqueda de TempLM

El algoritmo de búsqueda heurística de TempLM para resolver un problema de planificación temporal con restricciones $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ utiliza:

- La información contenida en el grafo de *landmarks temporales* (TLG) desarrollado en el capítulo anterior.
- Una adaptación de la heurística h^{LM-cut} [56] a un contexto temporal (Sección 5.3.1).

El proceso de búsqueda se realiza en un espacio de planes parciales donde un nodo se define de la siguiente forma:

Definición 5.1. *Un* ***nodo*** *del árbol de búsqueda es una tupla* (Π, S_t) *donde* Π *es un plan temporal parcial ejecutable tal que* $t = dur(\Pi)$ *y* S_t *representa el estado que se alcanza (en el instante* t*) tras la ejecución del plan* Π *desde el estado inicial* $\mathcal{I}$*.* ∎

El Algoritmo 5.1 describe el proceso de búsqueda de TempLM. SS representa el espacio de búsqueda que inicialmente contiene la tupla $(\emptyset, I)$, donde $\emptyset$ es el plan vacío e $\mathcal{I}$ el estado inicial del problema. Tras seleccionar un nodo (Π, S_t) de SS (los nodos en SS están ordenados de acuerdo al valor de su función de evaluación y se selecciona siempre el primer nodo de la lista) se comprueba si se cumplen en el estado S_t las condiciones at end de las acciones de Π que terminan en t; en caso contrario, el nodo (Π, S_t) se poda (líneas de la 7 a la 9). En caso que Π sea un plan solución, el proceso de búsqueda finaliza (líneas de la 10 a la 12). El resto del algoritmo describe el proceso encargado de generar los nodos sucesores (ver Sección 5.3.2). Para cada nodo sucesor, n, se calcula su función de evaluación $f(n)$ (se desarrolla en la Sección 5.3.1) y se añade a SS mediante la función $PushPriority$, almacenando el nodo en la posición correspondiente de acuerdo a su valor $f(n)$.

Algoritmo 5.1 Algoritmo para buscar un plan solución

1: **Input:** $\mathcal{P}, TLG$
2: {Inicialización del espacio de búsqueda}
3: $SS = \{(\emptyset, I)\}$
4: **while** $SS \neq \emptyset$ **do**
5: {Extraer el primer nodo de SS}
6: $(\Pi, S_t) = Pop(SS)$
7: **if** $\exists a \in \Pi : end(a) = t \wedge ECond(a) \notin S_t$ **then**
8: Se poda el nodo y se vuelve al paso 4
9: **end if**
10: **if** $isSolution((\Pi, S_t))$ **then**
11: **return** Π
12: **end if**
13: {Se generan los sucesores}
14: $Suc_t = successors((\Pi, S_t), TLG)$
15: **for** $n = (\Pi', S_{t'}) \in Suc_t$ **do**
16: {Se calcula el valor $f(n)$ para cada sucesor}
17: $f(n) = calculateF((\Pi', S_{t'}))$
18: {Se añade el nodo a SS según el valor de f }
19: $PushPriority(SS, (\Pi', S_{t'}), f)$
20: **end for**
21: **end while**
22: **return** no hay solución

5.3.1. Función de evaluación

El proceso de búsqueda se guía por la función de evaluación $f(n) = g(n) + h(n)$. En un problema de planificación temporal, el paralelismo de los planes debe tenerse en cuenta para obtener un valor exacto de $g(n)$. Por lo tanto, dado un nodo $n = (\Pi', S_{t'})$, definimos $g(n) = dur(\Pi')$. De esta manera, el valor $g(n)$ refleja exactamente el coste del nodo.

Para calcular el valor $h(n)$ se utiliza una adaptación de la heurística h^{LM-cut} (que denominaremos LM-cut). LM-cut es una heurística admisible que ofrece una de las mejores aproximaciones en tiempo polinómico de h^+ [56]. LM-cut se ha utilizado con éxito en planificación secuencial óptima ([56], [24], [12], [8]).

En líneas generales, LM-cut calcula un conjunto de grupos de acciones, donde cada grupo de acciones se denomina *cut* y donde cada *cut* corresponde a un *action landmark* del problema de planificación $\mathcal{P}$. El significado de un *cut* es el siguiente:

un plan solución para $\mathcal{P}$ debe incluir al menos una acción de cada *cut*; en caso contrario, $\mathcal{P}$ es irresoluble. Por lo tanto, el valor de la estimación de LM-cut para un estado del problema en dominios STRIPS es el ***número de cuts***.

En problemas que tienen acciones de diferente coste, un *cut* no representa un conjunto de acciones, sino un conjunto de *segmentos de acción*, lo que significa que el coste de una acción se distribuye, generalmente, a través de varios *cuts*. Dada una acción a cuyo coste es $c(a)$, un **segmento de acción** en un *cut* τ se denota por (a, c_a^τ), donde $c_a^\tau \leq c(a)$. Y un *cut* τ se define como un conjunto de segmentos de acción, es decir, $\tau = \{(a_1, c_{a_1}^\tau), (a_2, c_{a_2}^\tau), \ldots, (a_n, c_{a_n}^\tau)\}$. El coste de un *cut* τ se calcula como el mínimo coste de los segmentos de acción en el *cut*: $coste(\tau) = min_{\forall a_i \in \tau}(c_{a_i}^\tau)$. Además, para una acción a del problema, dado el conjunto de todos los *cut* obtenidos para un estado S ($Cuts(S)$) se cumple que:

$$\sum_{\forall \tau_i \in Cuts(S)} c_a^{\tau_i} \leq c(a) \tag{5.3.1}$$

Dado que un *cut* es un *action landmark*, al menos un segmento de acción de cada *cut* debe estar en todo plan de solución. La estimación de la heurística LM-cut para problemas con acciones de diferente coste sigue siendo el número de acciones que quedan para alcanzar los objetivos del problema; de este modo, cuando se selecciona dos segmentos de la misma acción, esto se contabiliza como una única acción. Esto implica que, a diferencia de los problemas STRIPS, la estimación del número de acciones de un plan en problemas de planificación con coste puede ser menor que el número de *cuts*.

En particular, para adaptar la heurística LM-cut a un contexto temporal (denominada T-LMcut) se asocia el coste de una acción a a la duración de la misma, es decir $c(a) = dur(a)$. La duración de una acción se distribuye a través de varios *cuts*, de forma que $c_a^\tau \leq dur(a)$. Por tanto, la estimación de T-LMcut para un estado S del problema ya no es el número de acciones sino el coste de las mismas de acuerdo a la siguiente ecuación:

$$h^{T-LMcut}(S) = \sum_{\forall \tau_i \in Cuts(S)} coste(\tau_i) \tag{5.3.2}$$

El valor de la heurística T-LMcut es una estimación basada en la duración de un plan secuencial. Evidentemente, este valor es una sobreestimación de la duración

Algoritmo 5.2 Función *successors*

```
1: Input: (Π, S_t), TLG
2: successorsList = ∅
3: A_t = initiallyApplicableActions(S_t)
4: for a ∈ A_t do
5:    t^a_E = EarliestStartTime(Π, S_t, a)
6:    (Π', S_t') = Succ(Π, S_t, a, t^a_E)
7:    if ¬invalidNode(Π', S_t', TLG) then
8:       Push (successorsList, (Π', S_t'))
9:    end if
10: end for
11: return successorsList
```

real del plan ya que se ignora el posible solapamiento de las acciones. Sin embargo, los buenos resultados mostrados por la heurística LM-cut en problemas STRIPS nos llevó a adoptar la heurística T-LMcut como una aproximación adecuada para planificación temporal subóptima.

5.3.2. Generación de sucesores

El Algoritmo 5.2 describe el proceso de generación de los sucesores de un nodo $n = (\Pi, S_t)$ que se ha seleccionado para su expansión. El cálculo de las acciones aplicables en el nodo n (línea 3) se muestra en la Sección 5.3.2.1. Posteriormente, se calcula el instante de tiempo donde se ejecutará cada acción a aplicable en el plan Π (Sección 5.3.2.2). A continuación se crea un nodo sucesor de n , $n' = (\Pi', S_{t'})$, insertando en Π' la acción aplicable en el nodo padre Π. Por último, si el nodo n' es un nodo válido (Sección 5.3.2.3) se añade a la lista *successorsList*.

5.3.2.1. Acciones aplicables

Una vez seleccionado el nodo a expandir, $n = (\Pi, S_t)$, se calcula el conjunto de acciones inicialmente aplicables de acuerdo a la siguiente definición:

Definición 5.2. *Una acción $a \in \mathcal{A}$ es una acción **inicialmente aplicable** en el estado S_t si $SCond(a) \cup Inv(a) \subseteq S_t$.* ∎

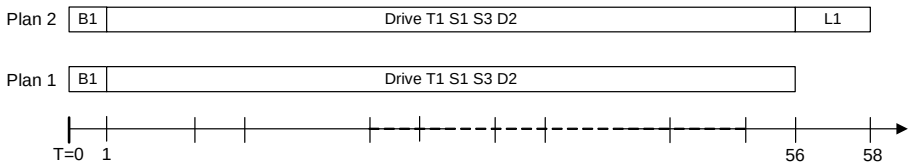

Figura 5.2: Aplicación de acciones reversibles para el ejemplo de la Figura 4.7

La Definición 5.2 calcula las acciones *inicialmente aplicables* sin tener en cuenta las condiciones at end, ($ECond$), ya que estas condiciones no se comprueban en el momento de generación del nodo sino en el momento de su expansión. Esto es así porque las condiciones $ECond$ se comprueban en el instante de finalización de la acción.

Una vez se han calculado las acciones *inicialmente aplicables* se descartan las acciones reversibles que conducen a un nodo ya visitado. Por ejemplo, la Figura 5.2 muestra dos planes distintos donde la acción (Drive T1 S3 S1 D2) sería una acción inicialmente aplicable en el estado S_{56} del $Plan1$[1] pero se descartaría porque conduciría al mismo estado ya visitado en S_1 (la acción (Drive T1 S3 S1 D2) es reversible de la acción que termina justamente en el instante t=56). Sin embargo, esta misma acción se puede aplicar en el estado S_{58} del plan parcial $Plan2$[2] que se muestra en la Figura 5.2 ya que, en este caso, no conduce a un estado previamente alcanzado.

La función $initiallyApplicableActions(S_t)$ de la línea 3 del Algoritmo 5.2 calcula y devuelve el conjunto de acciones que son *inicialmente aplicables* en el estado S_t.

5.3.2.2. Cálculo del instante de comienzo

Dada una acción $a \in A_t$, donde A_t es el conjunto de acciones *inicialmente aplicables* en S_t (línea 4 del Algoritmo 5.2), a es ejecutable en el instante t del nodo (Π, S_t), pero también puede ser ejecutable en algún instante anterior a t en el plan Π. Esta circunstancia es relevante porque determinará el *makespan* del plan resultante. Por esta razón, en la generación de sucesores se calcula el instante de comienzo más temprano para cada acción de A_t para lo cual es necesario introducir

[1]Donde $B1$ representa a la acción (Board D2 T1 S1)

[2]Donde $B1$ representa a la acción (Board D2 T1 S1) y $L1$ representa a la acción (Load P3 T1 S3)

las siguientes definiciones:

Definición 5.3. *Una acción* (a_i, t_{a_i}) *donde* t_{a_i} *es el instante de comienzo de la acción* a_i *se dice que* ***interfiere*** *con una acción* $(a_j, t_{a_j}) \in \Pi$ *donde* t_{a_j} *es el instante de tiempo en el que* a_j *comienza, si se produce alguna de las siguientes situaciones:*

- $\exists p \in Inv(a_j)$ *y* a_i *borra* p *dentro del intervalo de ejecución de* a_j*, es decir entre* t_{a_j} *y* $t_{a_j} + dur(a_j)$*.*
- $\exists q \in SCond(a_j)$ *y* a_i *borra* q *en un punto del intervalo definido entre el momento en el que se produce* q *y* t_{a_j}*.*
- $\exists r \in ECond(a_j)$ *y* a_i *borra* r *en un punto del intervalo definido entre el momento en el que se produce* r *y* $t_{a_j} + dur(a_j)$*.* ∎

En resumen, si a_i interfiere con a_j (que ya está en Π), entonces a_i no se añade a Π. Sin embargo, es posible encontrar otra rama del árbol donde estas dos acciones (a_i y a_j) aparecen en un plan junto con una tercera acción a_k que repone p, q o r (respectivamente). En esta situación las acciones se habrán añadido en este orden: $a_i < a_k < a_j$ y no se produce ninguna interferencia. Para calcular el instante de comienzo más temprano de una acción a se comprueba las interferencias con todos los pares de acciones (a,a') donde $a' \in \Pi$. El instante de comienzo más temprano de una acción se calcula del siguiente modo:

Definición 5.4. *Dado un nodo* (Π, S_t) *y una acción* $a_i \in A_t$*, el* ***instante de comienzo más temprano*** *de* a_i *(denotado como* $t_E^{a_i}$*) es el primer instante de tiempo desde* $\mathcal{I}$ *donde* a_i *es aplicable y no interfiere con ninguna acción de* Π*.* ∎

Sea el ejemplo de la Figura 5.3 donde $\Pi = Plan1$ está formado, únicamente, por la acción (Board D2 T1 S1) (en color verde); el objetivo es insertar la acción a=(Load P1 T1 S1) (en color blanco) en Π. Las flechas rojas marcan los instantes en los que la acción (Load P1 T1 S1) es aplicable. Evidentemente, (Load P1 T1 S1) es aplicable en el instante $t = 1$ ya que es una acción *inicialmente aplicable* es S_1,

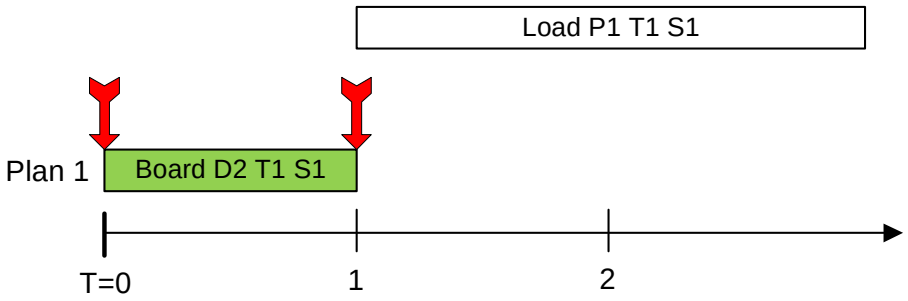

Figura 5.3: Cálculo del instante de comienzo más temprano para el ejemplo de la Figura 4.7

pero también es aplicable en el instante $t = 0$ ya que se cumplen las condiciones establecidas en la Definición 5.2. Si la acción (Load P1 T1 S1) se ejecuta en $t = 0$ no existirá ningún tipo de interferencia con la acción (Board D2 T1 S1) por lo que se podría conseguir el $Plan2$ que se muestra en la Figura 5.4 con las dos acciones en verde.

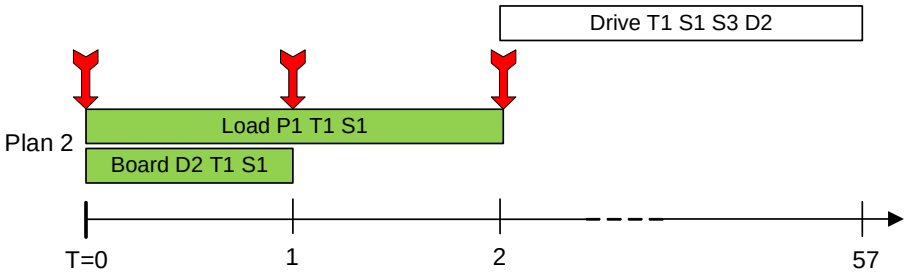

Figura 5.4: Cálculo del instante de comienzo más temprano para el ejemplo de la Figura 4.7

Una situación diferente que podría darse es la siguiente: consideremos el plan de la Figura 5.4, donde ya se dispone de las dos acciones en verde en Π, y se desea calcular y se desea calcular el instante de comienzo más temprano para la acción (Drive T1 S1 S3 D2). La acción (Drive T1 S1 S3 D2) es aplicable en $t = 2$ de acuerdo a la Definición 5.2. En el instante $t = 0$ no es aplicable porque no se satisfacen las condiciones de la Definición 5.2 (el conductor D2 no está en el camión $T1$). En el instante $t = 1$ se satisfacen las condiciones pero estas interfieren con la acción (Load P1 T1 S1) ya que la acción a_i =(Drive T1 S1 S3 D2) aplicada en $t_{a_i} = 1$ borraría p =(at T1 S1) que es una condición overall de la acción a_j =(Load P1 T1 S1), es

decir $p \in Inv(a_j)$. Por tanto, la acción (Drive T1 S1 S3 D2) sólo puede aplicarse en $t = 2$ tal y como se muestra en la Figura 5.5.

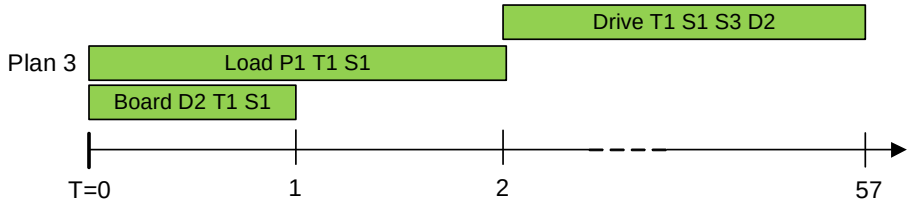

Figura 5.5: Resultado del cálculo del instante de comienzo más temprano para el ejemplo de la Figura 4.7

Una vez calculado el instante de comienzo más temprano $t_E^{a_i}$ para una acción a_i se crea un nodo sucesor de $n = (\Pi, S_t)$ donde se aplica la acción:

Definición 5.5. *El nodo* ***resultante de aplicar*** *en* $n = (\Pi, S_t)$ *una acción* a_i *en el instante* $t_E^{a_i}$ *es* $(\Pi', S_{t'})$ *donde* $\Pi' = \Pi \cup (a_i, t_E^{a_i})$ *y* $t' = dur(\Pi')$. $S_{t'}$ *es el resultado de la actualización sucesiva de los estados* S_t *donde* $t \in [t_E^{a_i}, t']$. *Es decir, se ejecutan todas las acciones* a_j *del plan* Π' *que comiencen o terminen en el intervalo* $(t_E^{a_i}, t']$. *Los estados se calculan de la siguiente forma:*

1. $S_{t+\varepsilon} = S_t - SDel(a_i) \cup SAdd(a_i)$
2. $\forall t_j \in (t_E^{a_i}, t']$
   a) $\forall a_j \in \Pi' \wedge t_E^{a_j} = t_j, S_{t_j} = S_{t_j - \varepsilon} - SDel(a_j) \cup SAdd(a_j)$
   b) $\forall a_k \in \Pi' \wedge t_E^{a_k} + dur(a_k) = t_j, S_{t_j} = S_{t_j - \varepsilon} - EDel(a_k) \cup EAdd(a_k)$

donde $S_{t'}$ *es el estado alcanzado tras la ejecución de todas las acciones del plan* Π'.

■

Sea el ejemplo de la Figura 5.6 donde $\Pi = Plan1$ está formado, únicamente, por la acción (Load P1 T1 S1) (en color verde). En el nodo $n = (\Pi, S_t)$ podemos observar los estados $S_0 = \{$`(at T1 S1)`, `(at D2 S1)`$\}$ y $S_2 = \{$`(at T1 S1)`, `(at D2 S1)`, `(in P1 T1)`$\}$. El objetivo es obtener el sucesor $(\Pi', S_{t'})$ tras insertar la acción

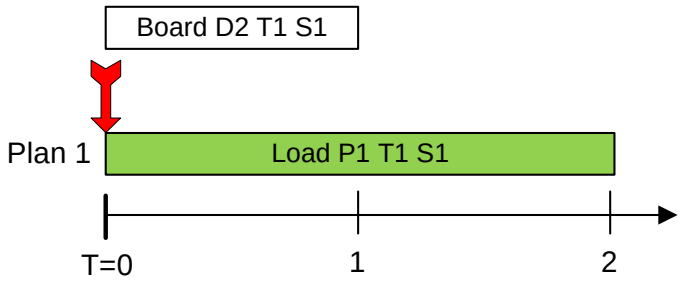

Figura 5.6: Cálculo del nodo resultante para el ejemplo de la Figura 4.7

a_i=(Load P1 T1 S1) (en color blanco) en Π. La acción a_i se puede aplicar en el instante $t_E^{a_i} = 0$ de acuerdo a la Definición 5.2. Los estados que se obtendrían en el nodo $(\Pi', S_{t'})$ serían los siguientes:

1. $S_0 = \{$`(at T1 S1)`$\}$
2. $S_1 = \{$`(at T1 S1)`, `(driving D2 T1)`$\}$
3. $S_2 = \{$`(at T1 S1)`, `(driving D2 T1)`, `(in P1 T1)`$\}$

El estado $S_{t'}$ sería el nuevo estado S_2 resultante de la aplicación de todas las acciones de Π'. Una vez se han calculado los sucesores $n' = (\Pi', S_{t'})$ de todas las acciones *inicialmente aplicables* (A_t) en un nodo $n = (\Pi, S_t)$, se descartan aquellos nodos n' que no son compatibles con la información del TLG.

5.3.2.3. Uso del grafo de landmarks temporales: Poda de nodos

Como se ha visto en el Capítulo 4, el TLG permite una rápida detección de problemas irresolubles durante la construcción del mismo, pero la información del TLG también es muy útil durante el proceso de búsqueda porque permite analizar si el conjunto de acciones de un nodo del árbol de búsqueda conduce o no a un plan solución.

Una vez que se han generado los nodos sucesores de un nodo n, se comprueba si cada nodo sucesor es válido de acuerdo a los *landmarks temporales*del TLG, sus relaciones de orden y sus intervalos temporales (líneas de la 7 a la 9 del Algoritmo 5.2). Todos los nodos del árbol de búsqueda se validan con respecto al TLG que se ha calculado para el estado inicial del problema y que se ha explicado en el capítulo

anterior. En la Sección 5.4 veremos que este proceso de validación se puede refinar calculando un TLG independiente para cada nodo del árbol.

Definición 5.6. *Un nodo $n = (\Pi, S_t)$ se dice que es **inválido** si se cumple alguna de estas tres condiciones:*

1. ***Intervalos temporales:** Dado un landmark l que se alcanza en el plan Π, se descarta el nodo n si $start(l)$ y/o $end(l)$ no son consistentes con el intervalo de validez y/o generación de l en el TLG. Así, el nodo se descarta si se cumple alguna de las siguientes condiciones:*

   a) $start(l) < min_v(l)$

   b) $start(l) > max_g(l)$

   c) $end(l) > max_v(l)$ *(sólo se aplica cuando se conoce $end(l)$)*

2. ***Relaciones de orden:** Dado un par de* landmarks *l_i y l_j en un plan Π, tal que $l_i \prec_{\{d,n\}} l_j$ en el TLG, si $start(l_i) \geq start(l_j)$, entonces el nodo se descarta.*

3. ***Inconsistencias:** Sea l_i un* landmark *en el TLG que no se ha conseguido en el plan Π y sea l_j un literal alcanzado en el plan Π que no es* landmark*, tal que l_i y l_j son* mutex *y tienen el mismo predicado ($predicate(l_i) = predicate(l_j)$). El nodo se descarta si*

$$start(l_j) + DIS_E(l_j, l_i) > max_g(l_i) \tag{5.3.3}$$

   Además, si se conoce $end(l_j)$ ($end(l_j) \neq \infty$) se descarta si el intervalo de validez de l_i solapa con la existencia de l_j en el plan:

$$[min_v(l_i), max_v(l_i)] \cap [start(l_j), end(l_j)] \tag{5.3.4}$$

■

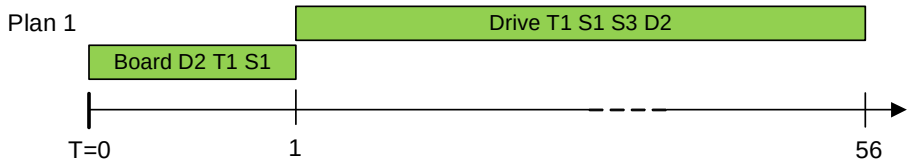

Figura 5.7: Plan parcial para el ejemplo de la Figura 4.7

La primera condición de la Definición 5.6 indica que un literal l del plan que es *landmark* en el TLG no satisface los intervalos temporales de su correspondiente *landmark* en el TLG. Por ejemplo, durante el proceso de búsqueda del plan solución para el ejemplo de la Figura 4.7, se puede obtener el plan parcial que se muestra en la Figura 5.7. El TLG obtenido para el ejemplo de la Figura 4.7 se muestra en la Figura 4.14. Este plan parcial añade el literal l =(at T1 S3) en el estado S_{56} y, por tanto, su instante de comienzo será $start(l) = 56$. De acuerdo con la información del TLG, se puede comprobar que $min_v(l) = 86$ y, por tanto, se cumple $start(l) < min_v(l)$, por lo que el nodo del árbol de búsqueda que contiene el plan de la Figura 5.7 se descarta. Esta decisión es correcta ya que, por un lado, en el plan de la Figura 5.7 el camión T1 se encuentra en S3 (at T1 S3) en el instante $t = 56$ y en dicho estado no se cumple el literal (in P1 T1) cuyo max_g(in P1 T1)=3; por otro lado, una vez conseguido l en $t = 56$, ya no es posible alcanzar (in P1 T1) en $t = 3$ porque la acción (Load P1 T1 S1) interfiere con (Drive T1 S1 S3).

La segunda condición de la Definición 5.6 permite descartar nodos en los que la relación de orden entre dos literales del plan es la inversa de la relación de orden entre estos mismos *landmarks* establecidas en el TLG. Por tanto, esta condición es complementaria a la anterior. Cuando se dispone de suficiente información, los intervalos de un *landmark* del TLG están suficientemente restringidos como para que la primera condición de la Definición 5.6 detecte inconsistencias entre los *landmark* del TLG y los literales del plan y por tanto se descarten los nodos inconsistentes. En cambio, cuando los intervalos de un *landmark* l_i en el TLG están muy poco restringidos, son las relaciones de orden de l_i con otros *landmarks* l_j del TLG las que pueden permitir detectar estas inconsistencias.

La tercera condición de la Definición 5.6 permite podar nodos en los que se ha alcanzado un estado S_t a partir del cual no es posible satisfacer el intervalo de generación de algún *landmark* l_i. A diferencia de las condiciones anteriores, en esta

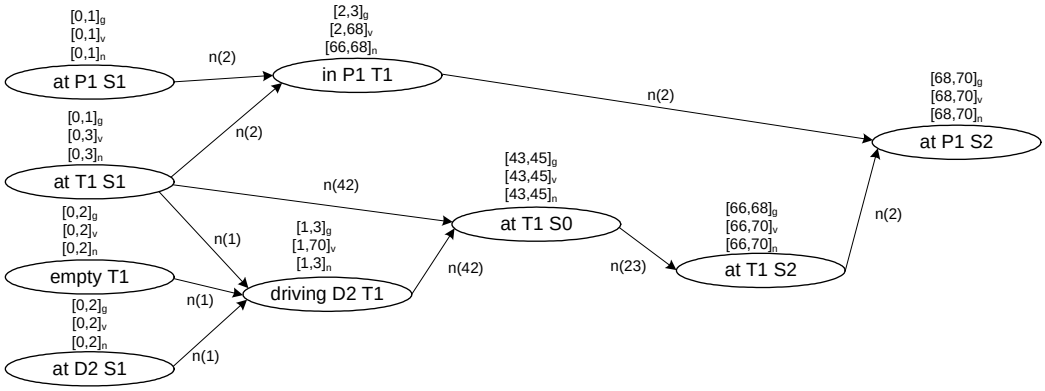

Figura 5.8: TLG para el ejemplo modificado (conseguir sólo el objetivo $g =$(at P1 S2)) de la Figura 4.7

condición se estudia la interferencia entre un literal l_j alcanzado en el plan que no pertenece al TLG y un *landmark* l_i del TLG. Por otro lado, la aplicación de esta condición permite adelantarse a situaciones de conflicto que podrían aparecer más adelante. Vamos a tomar como ejemplo el problema de la Figura 4.7 pero considerando que sólo se desea conseguir el objetivo $g =$(at P1 S2) con la restricción (within 70 (at P1 S2)) (la misma que en el problema original). En este caso, el TLG que se obtiene para este nuevo objetivo es el que se muestra en la Figura 5.8. Asumiendo que durante el proceso de búsqueda se ha generado el plan parcial de la Figura 5.9, entonces el literal $l_j =$(at T1 S3) se añade al estado S_{57}. Se puede observar que l_j no es un *landmark* del TLG de la Figura 5.8 y además es *mutex* con el *landmark* $l_i =$(at T1 S2) (entre otros), cuyo $max_g(l_i) = 68$. Teniendo en cuenta que $DIS_E(l_j, l_i) = 20$, se satisface la Condición 5.3.3, ya que $start(l_j) + 20 > max_g(l_i)$ por tanto el nodo se poda. Esta decisión es correcta ya que en el estado S_{57} el camión T1 se encuentra en S3 cargado con P1 y llegar desde S3 hasta S2 (destino de P1) tiene un coste de 20, es decir, T1 llegará a S2 en $t = 77$, lo que impedirá satisfacer la restricción temporal impuesta para descargar P1 en S2 en $t = 70$.

5.4. Proceso de *feedback* de TempLM

En esta sección se describe con detalle una nueva funcionalidad de TempLM que, en lugar de validar los nodos del árbol con el TLG del estado inicial del problema, se

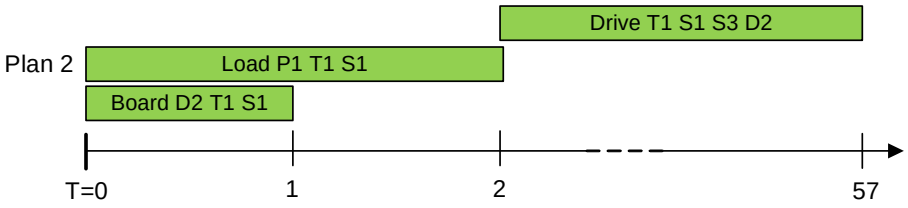

Figura 5.9: Plan parcial para el ejemplo modificado de la Figura 4.7

validan mediante un TLG construido para cada nodo $n = (\Pi, S_t)$ a partir del estado S_t. El objetivo es utilizar la información del plan parcial de un nodo del espacio de búsqueda para refinar su correspondiente TLG y utilizar asimismo el TLG del plan parcial para determinar si el nodo debe ser podado. Esta nueva funcionalidad se denomina *refinamiento* del TLG o *feedback* para denotar la retroalimentación existente entre las acciones que componen el plan parcial de un nodo del espacio de búsqueda y su grafo de *landmarks temporales* [77].

El proceso de refinamiento del TLG se define del siguiente modo: cuando se añade una nueva acción a un plan parcial, se toma una decisión que permite inferir nuevos *landmarks temporales* en su TLG. Por ejemplo, si en el problema de la Figura 4.7 hay dos camiones en la posición S1 y decidimos cargar el paquete P1 en el camión T1, se podrán añadir nuevos *landmarks* para indicar los desplazamientos que tiene que realizar el camión T1 con el objetivo de transportar el paquete P1 a su destino.

Para realizar el proceso de refinamiento de un TLG cada nodo debe disponer de su propio grafo de *landmarks temporales*:

Definición 5.7. *Un* ***nodo feedback*** *del árbol es una tripleta* (Π, S_t, TLG_Π) *donde* Π *es un plan temporal parcial ejecutable tal que* $t = dur(\Pi)$*,* S_t *representa el estado que se alcanza (en el instante* t*) tras la ejecución del plan* Π *desde* $\mathcal{I}$ *y* TLG_Π *representa el* TLG *del nodo.* ∎

El TLG asociado a un nodo *feedback* del árbol de búsqueda se utilizará para tomar decisiones sobre la poda de dicho nodo. En la siguiente subsección se detalla el refinamiento del TLG de un nodo a partir del TLG de su padre analizando las acciones contenidas en el plan parcial de dicho nodo.

5.4.1. Refinamiento del grafo de landmarks temporales

Sea $n = (\Pi, S_t, TLG_\Pi)$ un nodo *feedback* y sea $n' = (\Pi', S_{t'}, TLG_{\Pi'})$ un nodo *feedback* sucesor de n donde se ha añadido la acción a al plan Π en el instante t_E^a obteniendo el plan Π', es decir $\Pi' = \Pi \cup (a, t_E^a)$, y $S_{t'}$ se ha calculado de acuerdo a la Definición 5.5. $TLG_{\Pi'}$ se calcula a partir de TLG_Π del siguiente modo:

1. Inicialmente se copia el TLG del padre en el hijo ($TLG_{\Pi'} = TLG_\Pi$).

2. Sea $L_{add} = S_{t'}$; se añaden los literales de L_{add} a $TLG_{\Pi'}$. Algunos literales de L_{add} pueden estar ya presentes en $TLG_{\Pi'}$ porque ya aparecían en TLG_Π por lo que esta operación simplemente añade los literales nuevos de L_{add} que no están en TLG_Π. Por ejemplo, asumamos que esta operación inserta el literal l_{11} en $TLG_{\Pi'}$ tal y como se muestra en la parte superior de la Figura 5.10 (l_{11} se representa como un círculo de color verde).

3. Sea L_{del} el conjunto de literales que se eliminan en el cálculo de $S_{t'}$; se borran los literales contenidos en L_{del} de $TLG_{\Pi'}$. Al igual que con los literales L_{add} puede haber literales de L_{del} que no estén en $TLG_{\Pi'}$ porque se han borrado durante la construcción del TLG del padre. Siguiendo con el ejemplo de la Figura 5.10, asumamos que esta operación borra el *landmark* l_4 (ver parte superior de la Figura 5.10 donde l_4 está marcado con una X de color rojo).

4. Se completa $TLG_{\Pi'}$ para *reparar* los enlaces eliminados. En la parte superior de la Figura 5.10 se puede observar que dado que el l_4, el *landmark* predecesor de l_6 y l_7, se ha eliminado en el paso 3 (líneas rojas), es necesario calcular una nueva forma de satisfacer l_6 y l_7 a partir del conjunto L_{add}. Esto permite inferir nuevos *landmarks temporales* y nuevas relaciones de orden que completen $TLG_{\Pi'}$. En la parte inferior de la Figura 5.10 se muestran los literales y/o enlaces que se añaden para satisfacer a l_6 y l_7 donde se ha añadido un nuevo *landmark* l_{12} que junto con l_{11} permite restablecer l_6 y l_7 (círculo y líneas de color azul).

El paso 4 indicado arriba define una extracción local de *landmarks temporales* y relaciones de orden a partir de la información de Π' que sigue el mismo proceso explicado en el Capítulo 4. Por tanto, se construye un $TRPG$ desde $\mathcal{I}'$ a $\mathcal{G}'$ donde:

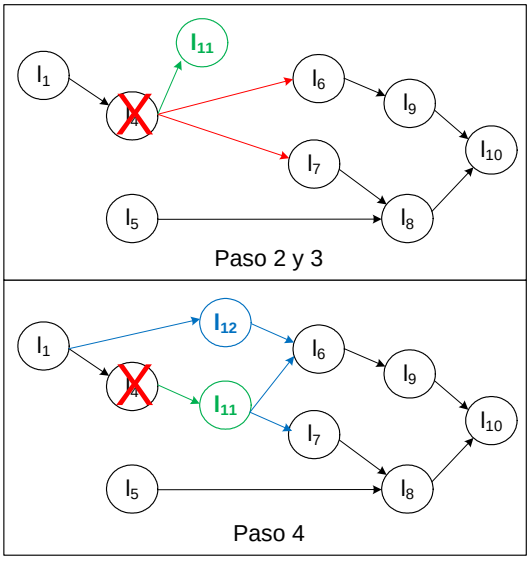

Figura 5.10: Refinamiento del $TLG_{\Pi'}$ en dos etapas: añadir/borrar literales y completar el TLG

- $\mathcal{I}'$ es el estado inicial, formado por el conjunto L_{add}. En lugar de situar todos los literales en $t = 0$ (el primer nivel del $TRPG$), cada literal se sitúa en el punto de tiempo (nivel) donde se ha conseguido en Π'. De esta forma, el $TRPG$ refleja una imagen fiable de la situación actual que está representada en el plan Π' .

- $\mathcal{G}'$ es el estado objetivo, formado por el conjunto de *landmarks* de TLG_{Π} que no se satisfecen en Π' y cuyas condiciones pertenecen a L_{del}.

El $TRPG$ desde $\mathcal{I}'$ a $\mathcal{G}'$ permite calcular $fa^t(l_j)$ y $lit\_labels_l^t(l_j)$ para cada $l_j \in \mathcal{G}'$. Si $l_j \in \mathcal{G}'$ no se consigue en el $TRPG$ antes de $max_g(l_j)$, el nodo n' se puede podar. En caso contrario, $lit\_labels_l^t(l_j)$ en $t = max_g(l_j)$ son los nuevos *landmarks temporales* que permiten conseguir l_j en $max_g(l_j)$ y que se añaden a $TLG_{\Pi'}$ junto con sus relaciones de orden. Por último, se propagan las restricciones a través del TLG y se realiza el análisis de inconsistencias.

Por ejemplo, consideremos de nuevo que el único objetivo del ejemplo de la Figura 4.7 es $g =$(at P1 S2) con la restricción (within 70 (at P1 S2)) (la misma que en el problema original) y que disponemos de otro camión T2 en S1, (at T2 S1). Ante esta nueva situación, el TLG que se obtiene a partir del estado inicial del problema

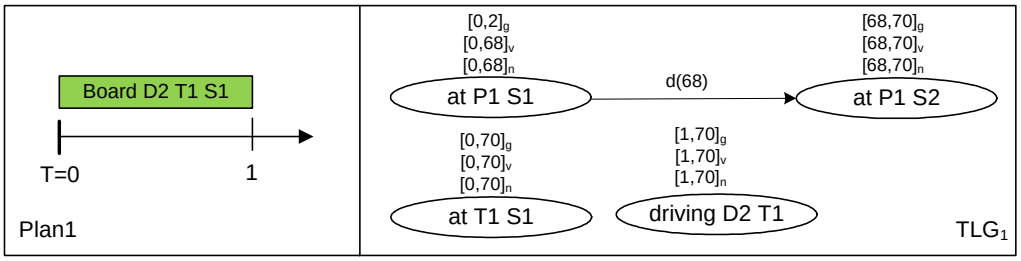

Figura 5.11: Plan y TLG tras aplicar la acción (Board D2 T1 S1)

estará formado únicamente por los literales de la situación inicial y el objetivo ya que no se dispone de información suficiente para poder añadir *landmarks* adicionales. La Figura 5.11 muestra el plan y el TLG que se obtiene tras aplicar la acción (Board D2 T1 S1). En TLG_1 se muestran los *landmarks* (driving D2 T1), (at T1 S1) y (at P1 S1) que forman el nuevo estado que se alcanza tras aplicar la acción (Board D2 T1 S1) (se han omitido el resto de literales de la situación inicial que no se utilizan en este proceso). Entre los posibles sucesores de $Plan1$ podemos encontrar los siguientes:

1. Aplicar la acción (Drive T1 S1 S0 D2) en $t = 1$ con una duración de 42 u.t.; el TLG de este nodo sucesor contendría los mismos landmarks que TLG_1 pero sustituyendo el *landmark* (at T1 S1) por (at T1 S0). De acuerdo a las condiciones establecidas en la Sección 5.3.2.3, este nodo se poda porque (at P1 S2) no se puede conseguir antes del instante 70 ya que no se ha cargado el paquete P1 en el camión T1 y el camión T1 no puede cargar el paquete P1 porque se ha desplazado a S0. Otra posibilidad es cargar el paquete P1 en el camión T2 que se encuentra en S1, pero en este caso no disponemos de otro conductor que pueda conducir el camión T2 para llevar el paquete P1 a S2.

2. Aplicar la acción (Load P1 T1 S1) en $t = 0$ con una duración de 2 u.t.; se obtendría el $Plan2$ que se muestra en la Figura 5.12.

Tras la simulación de la ejecución de las acciones de $Plan2$ se obtendría el conjunto $L_{del} = \{$(at P1 S1)$\}$ y el estado S_t de $Plan2$ que será el nuevo estado inicial $\mathcal{I}'$ para calcular el $TRPG$. Por tanto, $\mathcal{I}'$ contendría:

1. literales de $\mathcal{I}$ que no se han borrado: (at D1 S0), (at P2 S1), (at P3 S3), (at T2 S1), (empty T2), (at T1 S1);

2. el literal (driving D2 T1) en $t = 1$ que resulta de la ejecución de la acción del $Plan$ 2;

3. el literal (in P1 T1) en $t = 2$ que resulta de aplicar la acción (Load P1 T1 S1).

A continuación obtendríamos el estado $\mathcal{G}'$, formado por el literal (at P1 S2), que se debe volver a conseguir antes de max_g(at P1 S2)= 70 porque su condición (at P1 S1) pertenece a la lista L_{del}. Una vez definido $\mathcal{I}'$ y $\mathcal{G}'$ sólo queda calcular el $TRPG$ y extraer los literales que pertenecen a $lit\_labels_l^{70}$(at P1 S2). Los nuevos *landmarks* que se añaden a TLG_2 serán (at T1 S0) y (at T1 S2). Es decir, una vez se toma la decisión en el plan de cargar el paquete P1 en el camión T1, se puede deducir cuál será la secuencia de localizaciones que debe recorrer dicho camión para transportar P1 hasta S2. La Figura 5.12 muestra TLG_2, el TLG correspondiente al $Plan2$.

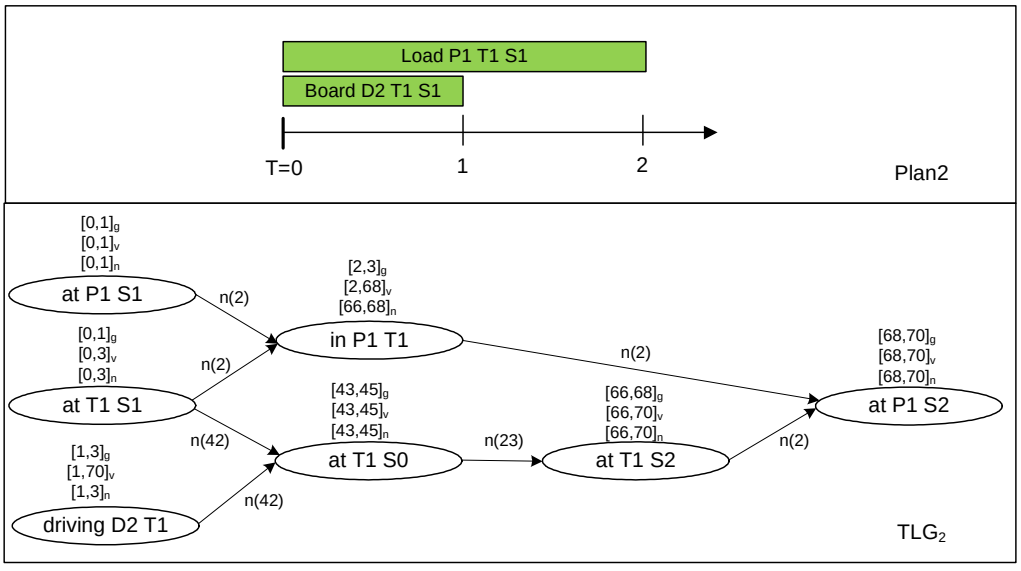

Figura 5.12: $Plan2$ y su TLG tras el proceso de refinamiento

A continuación se muestran dos versiones del proceso de *Feedback* que sólo se diferencian en el momento en el que se aplica el proceso.

Algoritmo 5.3 Función *successorsFeedback*

```
1: Input: (Π, S_t), TLG_Π
2: successorsList = ∅
3: A_t = initiallyApplicableActions(S_t)
4: for a ∈ A_t do
5:    t^a_E = EarliestStartTime(Π, S_t, a)
6:    (Π', S_t') = Succ(Π, S_t, a, t^a_E)
7:    if TLG_Π' = refinementTLG(Π', S_t', TLG_Π) then
8:       if ¬invalidNode(Π', S_t', TLG_Π') then
9:          Push (successorsList, (Π', S_t', TLG_Π'))
10:      end if
11:   end if
12: end for
13: return successorsList
```

5.4.2. *Feedback* V1

En esta primera versión, el proceso de *feedback* se realiza cuando se ha generado un plan Π' a partir del plan Π de su nodo padre. El Algoritmo 5.3 es una versión adaptada del Algoritmo 5.2 que construye $TLG_{\Pi'}$ aplicando el proceso de refinamiento sobre TLG_{Π} en la línea 7.

La construcción de $TLG_{\Pi'}$ consiste en realizar una copia de TLG_{Π} y aplicar el refinamiento comentado en la Sección 5.4.1. Realizar la copia de un TLG es un proceso muy costoso debido a la gran cantidad de información que contiene el grafo: *landmarks temporales* con sus intervalos, restricciones de orden, restricciones temporales, etc. En el Algoritmo 5.3 se crea $TLG_{\Pi'}$ a partir de TLG_{Π} en la línea 7 y a continuación se comprueba si Π' es un plan consistente con la información de $TLG_{\Pi'}$. Sin embargo, se puede comprobar la consistencia de Π' utilizando TLG_{Π} (es decir el TLG del padre). Evidentemente, la información de $TLG_{\Pi'}$ es más ajustada al contenido de Π' pero dado el elevado coste temporal de la copia y refinamiento del TLG una opción es postponer la generación de $TLG_{\Pi'}$ una vez se ha comprobado que el nodo es válido con la información de TLG_{Π}. En concreto los pasos que se realizan en *Feedback V1* son los siguientes:

1. Se genera el nodo $n' = (\Pi', S_{t'}, TLG_{\Pi})$

2. Se comprueba si n' es consistente con el TLG del padre (TLG_{Π})

3. Si n' es consistente, se construye su $TLG_{\Pi'}$ y se añade $n' = (\Pi', S_{t'}, TLG_{\Pi'})$ a la lista $successorsList$.

A efectos de rendimiento, *Feedback* V1 mejora el coste temporal de TempLM porque evita la construcción del TLG para nodos que no son válidos. Sin embargo, el coste sigue siendo excesivo porque el proceso de refinamiento del TLG se aplica sobre una gran cantidad de nodos que no se expanden nunca. Por este motivo se crea la versión 2 que se comenta en la siguiente sección.

5.4.3. *Feedback* V2

Como se ha mencionado en la sección anterior, el principal inconveniente del proceso de *feedback* radica en el tiempo necesario para realizar la copia del TLG. En *Feedback* V2 se trata de reducir el coste del proceso de copia realizándolo únicamente cuando se selecciona un nodo para su expansión. De este modo, el Algoritmo 5.1 se modifica para aplicar el refinamiento del TLG en el momento en que se generan los sucesores de un nodo. En el Algoritmo 5.4, el proceso de refinamiento se realiza en la línea 13 tras haber comprobado que se cumplen las condiciones at end y que el nodo no es un plan solución. En el caso que no sea posible construir el $TRPG$ (paso 4 en la Sección 5.4.1) durante el proceso de refinamiento debido a que un objetivo de $\mathcal{G}'$ no se puede conseguir antes de su *deadline* max_g, entonces no se construye $TLG_{\Pi'}$ y el nodo se poda. En resumen, los pasos que se realizan en *Feedback* V2 son los siguientes:

1. Se genera el nodo $n' = (\Pi', S_{t'}, TLG_{\Pi})$
2. Se comprueba si n' es consistente con el TLG del padre (TLG_{Π})
3. Si n' es consistente, se añade $n' = (\Pi', S_{t'}, TLG_{\Pi})$ a la lista $successorsList$
4. Cuando el nodo $n' = (\Pi', S_{t'}, TLG_{\Pi})$ se expande, se construye su $TLG_{\Pi'}$ y por tanto $n' = (\Pi', S_{t'}, TLG_{\Pi'})$

Feedback V2 permite resolver un mayor número de problemas que *Feedback* V1 y dentro de unos tiempo razonables aunque hay dominios en los que su consumo de tiempo aún es excesivo. La versión *Feedback* V2 es la versión utilizada para la realización de los experimentos que se muestran en el Capítulo 6.

Algoritmo 5.4 Algoritmo para buscar un plan solución con *feedback*

1: **Input:** $\mathcal{P}, TLG_0$
2: {Inicialización del espacio de búsqueda}
3: $SS = \{(\emptyset, I, TLG_0)\}$
4: **while** $SS \neq \emptyset$ **do**
5: {Extraer el primer nodo de SS}
6: $(\Pi, S_t, TLG_\Pi) = Pop(SS)$
7: **if** $\exists a \in \Pi : end(a) = t \wedge ECond(a) \notin S_t$ **then**
8: Se poda el nodo y se vuelve al paso 4
9: **end if**
10: **if** $isSolution((\Pi, S_t, TLG_\Pi))$ **then**
11: **return** Π
12: **end if**
13: **if** $TLG_{\Pi'} = refinementTLG(\Pi, S_t, TLG_\Pi)$ **then**
14: {Se generan los sucesores}
15: $Suc_t = successors((\Pi, S_t, TLG_{\Pi'}), TLG_{\Pi'})$
16: **for** $(\Pi', S_{t'}, TLG_{\Pi'}) \in Suc_t$ **do**
17: {Se calcula el valor f para cada sucesor}
18: $f = calculateF((\Pi', S_{t'}, TLG_{\Pi'}))$
19: {Se añade el nodo a SS según el valor de f }
20: $PushPriority(SS, (\Pi', S_{t'}, TLG_{\Pi'}), f)$
21: **end for**
22: **end if**
23: **end while**
24: **return** no hay solución

5.5. Análisis de la complejidad

En general, un problema de planificación independiente del dominio es un problema *PSPACE-completo* [14]. La complejidad de los problemas de planificación temporal que pueden reducirse a problemas secuenciales es también *PSPACE-completo* mientras que para problemas concurrentes, la complejidad es *EXPSPACE-completo* [97]. TempLM no maneja dominios con acciones concurrentes en las que se requiera que una acción termine en un instante concreto debido a que las acciones inicialmente aplicables se sitúan en su *instante de comienzo más temprano* y no se permite retrasar estas acciones, de modo que la complejidad de TempLM es *PSPACE-completo*.

Los tres algoritmos más relevantes de TempLM son los siguientes:

- La construcción del $TRPG$ para la extracción de los *landmarks temporales*.

- La construcción del TLG, actualización y propagación de la información temporal.

- El proceso de búsqueda de un plan solución.

El algoritmo para la construcción del $TRPG$ funciona de acuerdo a la Definición 4.2 y su coste es polinómico en función del número de nodos literal (L) y del número de nodos acción (A). Así:

$$T_{TRPG}(L, A) \in O(L \cdot A) \tag{5.5.1}$$

Por otro lado, en el algoritmo para la construcción del TLG hay que destacar que la parte más costosa es el proceso de actualización y propagación de la información temporal que se explicó en la Sección 4.4.2 donde se actualiza la información de todos los *landmarks temporales* a través de la relaciones de orden. La complejidad del Algoritmo 4.3 en el caso peor donde se tiene que actualizar cada *landmark temporal* y propagar su información al resto de *landmarks temporales* a través de la pila *events* sería:

$$T_{TLG}(N) \in O(N^2) \tag{5.5.2}$$

donde N es el número de *landmarks temporales* en el TLG.

Por último, la búsqueda del plan se realiza de acuerdo a una búsqueda heurística de tipo A guiada por la función heurística T-LMcut. Como se ha comentado en la Sección 5.3.1, T-LMcut es una heurística no admisible aunque hay que destacar que, para calcular el TLG del estado inicial del problema, se dispone de un *deadline* o límite temporal $T_{\mathcal{P}}$ para los planes solución de un problema $\mathcal{P}$. Este *deadline* se utiliza como cota de la duración de un plan Π durante el proceso de búsqueda $dur(\Pi) \leq T_{\mathcal{P}}$ de forma que el coste del proceso de búsqueda de acuerdo con el Algoritmo 5.1 sería exponencial en función del factor de ramificación y del *deadline*, es decir el coste de un algoritmo A^*:

$$T_{busqueda}(b) \in O(b^{(T_{\mathcal{P}}/min\_coste\_accion)}) \tag{5.5.3}$$

donde b es el factor de ramificación y min_coste_accion el mínimo coste de las acciones del problema.

5.6. Conclusiones

En este capítulo se ha presentado TempLM, un planificador temporal independiente del dominio para problemas de planificación con restricciones. TempLM utiliza la heurística T-LMcut para guiar el proceso de búsqueda, siendo una heurística muy adecuada para planificación temporal subóptima por los buenos resultados mostrados por la heurística LM-cut en problemas STRIPS. También se describe el proceso para podar nodos del espacio de búsqueda que no satisfacen los requisitos indicados en el TLG. Adicionalmente, se muestra una nueva funcionalidad que establece una retroalimentación (*feedback*) entre la información de los planes parciales de los nodos del espacio de búsqueda y sus grafos de *landmarks temporales*. La aplicación del proceso de *feedback* requiere que se asocie un TLG distinto a cada nodo, prescindiendo así del TLG inicial cuando se han creado los sucesores del nodo raíz. La utilización del *feedback* acelera el proceso de búsqueda al descartar los nodos que no conducen a un plan solución utilizando tanto la información de su plan parcial como la de su propio TLG.

Capítulo 6

Resultados experimentales

6.1. Introducción

En este capítulo se presentan los resultados experimentales obtenidos con la aproximación TempLM para la resolución de problemas de planificación temporal con restricciones.

Como se ha comentado en la Sección 2.4.2, el lenguaje PDDL3.0 se introdujo en la *IPC* 2006 ([43]) donde se realizó la primera y única competición de planificación con gestión de restricciones de trayectorias de estado y preferencias. Dos planificadores participaron en esta competición, MIPS-XXL [28] y SGPLAN5 [16]. Centrándonos exclusivamente en los experimentos sobre dominios con restricciones temporales (*Time Constraints Track*), los resultados obtenidos fueron [1]:

- el número de problemas resueltos para cada dominio fue: truck-TimeConstraints (MIPS-XXL (4), SGPLAN5 (20)), pipesworld-MetricTimeConstraints (MIPS-XXL (1), SGPLAN5 (0)), storage-TimeConstraints MIPS-XXL (4), SGPLAN5 (9)) y TPP-MetricTimeConstraints (MIPS-XXL (5), SGPLAN5 (18)). A raíz de estos resultados, se puede concluir que el número de problemas resueltos por SGPLAN5 fue muy superior al de MIPS-XXL.

[1] Ver http://idm-lab.org/wiki/icaps/ipc2006/deterministic/)

- la calidad de los planes de MIPS-XXL (*makespan*) fue siempre igual o superior a la de SGPLAN5, obteniendo planes significativamente más cortos que SGPLAN5.

- el rendimiento de SGPLAN5 fue siempre muy superior al de MIPS-XXL, obteniendo en algunos dominios tiempos de ejecución varios órdenes de magnitud inferior.

Las pruebas de la *Time Constraints Track* no incluía problemas irresolubles, al menos generados conscientemente. En la página web de la competición se indica literalmente que *"aunque se ha intentado generar constraints de modo que los problemas sean resolubles, no hay garantía de que así sea en todas las instancias de problemas; desafortunadamente, no hay planificador que se pueda utilizar para realizar esta comprobación"*. Y es precisamente en este punto donde reside nuestro principal objetivo, que TempLM sea un referente para la detección de problemas irresolubles de planificación temporal con restricciones.

Para nuestros propósitos, hemos diseñado una serie de experimentos donde se compara el comportamiento de TempLM con MIPS-XXL y SGPLAN5 así como con el planificador OPTIC. Particularmente, en el trabajo presentado en la conferencia *International Conference on Automated Planning and Scheduling* (ICAPS-2012) [5], los autores de OPTIC demuestran que la calidad de los planes de este planificador para los problemas de la competición *IPC* 2006 es muy superior a la de MIPS-XXL y SGPLAN5. Los resultados muestran que OPTIC obtiene la mejor solución en la mayoría de problemas además de presentar una buena escalabilidad. A tenor de estos resultados, nuestros experimentos se organizan de la siguiente forma:

- Comparativa I: una primera comparativa donde se compara TempLM con MIPS-XXL, SGPLAN5 y OPTIC para problemas irresolubles, problemas resolubles con restricciones ajustadas (*tight*) y problemas resolubles con restricciones holgadas (*loose*). Unos resultados preliminares de la comparativa entre TempLM y SGPLAN se puede ver en el trabajo publicado en [74]. La Sección 6.3 analiza en detalle los resultados de la primera comparativa.

- Comparativa II: una segunda comparativa donde se resalta la aportación del uso de *landmarks temporales* y el mecanismo de *feedback* en la resolución de

problemas. Esta segunda comparativa se ha realizado únicamente con el planificador OPTIC, lo cual se justifica a la vista de los resultados de la comparativa I. La Sección 6.4 analiza en detalle los resultados de la segunda comparativa.

- Comparativa III: una tercera comparativa de TempLM, OPTIC y OPTIC enriquecido con una nueva aproximación de Karpas *et al.* presentada en la conferencia *International Conference on Automated Planning and Scheduling* (ICAPS-15) [66]. La Sección 6.5 analiza en detalle los resultados de la comparativa III.

Previamente a la presentación de resultados, la siguiente sección resume las principales características de los dominios utilizados así como la configuración de los experimentos y los problemas *loose*, *tight* e irresolubles.

6.2. Dominios y configuración de los experimentos

En esta sección se describe brevemente la configuración de los experimentos, cómo se han generado los problemas irresolubles, *tight* y *loose*, y se ofrece una pequeña descripción de los dominios utilizados en los experimentos así como las razones que justifican la no inclusión de otros dominios.

Nuestro objetivo es probar problemas donde se impone un *deadline* para la resolución del mismo, es decir, para la consecución de los objetivos del problema. Para ello, tal y como se explicó en la Sección 4.2, para un problema $\mathcal{P} = \langle \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ hay que definir un *deadline* $T_{\mathcal{P}}$ como $T_{\mathcal{P}} = \max_{(t,l)\in\mathcal{D}} (t)$. Asimismo, definir $T_{\mathcal{P}}$ requiere definir un *deadline* para cada objetivo en G. Adicionalmente, es posible definir un *deadline* para cualquier otro literal que se alcance en el problema.

Tal y como se ha visto a lo largo de este documento, nuestro modelo TempLM es capaz de manejar todas las restricciones temporales de PDDL3.0 que representan un *deadline* mediante los operadores modales within, always-within, sometime-before y sometime-after, así como *deadlines* expresados mediante *time initial literals (TILs)*. Esto es posible porque todas estas restricciones pueden traducirse internamente al formato (t, l) requerido para definir el componente D de un problema de planificación temporal con restricciones $\mathcal{P}$. Igualmente, MIPS-XXL y SGPLAN5 pueden manejar *deadlines* expresados con cualquiera de los operadores mencionados. Sin embargo,

el compilador de OPTIC sólo acepta restricciones expresadas con within y $TILs$. Por este motivo, las restricciones que se han incorporado en los experimentos se expresan siempre con within o $TILs$. Obsérvese que el funcionamiento interno de TempLM sería independiente de la restricción de PDDL3.0 gracias a la conversión interna de un *deadline* al formato (tiempo, landmark) (ver Sección 4.3.2).

Respecto a la generación de problemas, se crearon dos baterías de problemas aleatorios. La primera batería contiene un conjunto de problemas 'no clasificados', es decir, se desconoce si el problema es irresoluble o no, y se analiza la capacidad de detección de irresolubilidad de cada una de los planificadores así como la calidad de los planes de aquellos problemas que sí son resolubles. La segunda batería de problemas genera un conjunto de problemas 'clasificados' en irresolubles, *tight* y *loose*.

La generación de los problemas de las dos baterías requiere disponer del *makespan* del plan óptimo o, en su defecto, del *makespan* del mejor plan conocido. Estos valores se obtuvieron consultando los resultados de las diferentes competiciones de planificación y recuperando el mejor *makespan* entre todos los planificadores participantes. Así, por ejemplo, para los dominios utilizados de la *IPC* 2011 se tomó como referencia el mejor *makespan* proporcionado en el informe de resultados de la competición (http://www.plg.inf.uc3m.es/ipc2011-deterministic/). Para los dominios de otras competiciones, se compararon los planes obtenidos por los distintos planificadores tomando como referencia el *makespan* del plan más corto.

Para generar un problema de planificación con restricciones $\mathcal{P}$, se toma el mejor *makespan* conocido (M) para el problema sin restricciones, y se utiliza un intervalo de valores $[x, y]$ para delimitar el *makespan* de $\mathcal{P}$:

- un **problema no clasificado** $\mathcal{P}$ se genera utilizando un valor aleatorio $M' \in [0.75M, 1.5M]$ y estableciendo M' como el *deadline* para todos los objetivos del problema.
- un **problema clasificado** $\mathcal{P}$ se genera utilizando diversos intervalos para los *deadlines* de los objetivos del problema:
  - un problema $\mathcal{P}$ clasificado como *loose* se genera calculando un valor aleatorio $M' \in [1.2M, 1.4M]$, que generalmente será distinto, para cada uno de los objetivos del problema.

- un problema $\mathcal{P}$ clasificado como *tight* se genera calculando un valor aleatorio $M' \in [M, 1.2M]$, que generalmente será distinto, para cada uno de los objetivos del problema.

- un problema $\mathcal{P}$ clasificado como irresoluble se genera a partir de un problema *tight* restringiendo alguno de los *deadlines* de los objetivos hasta conseguir que el problema no tenga solución.

A continuación se describen los dominios utilizados para los experimentos así como una pequeña explicación de las restricciones impuestas en los objetivos del problema.

En primer lugar, cabe destacar que TempLM no maneja variables numéricas (*fluents*) por lo que no ha sido posible realizar pruebas en el dominio Travelling and Purchase Problem Domain (TPP, *IPC* 2006), donde todos los objetivos del problema se definen como condiciones numéricas de dos variables. Por otro lado, la gran mayoría de instancias de problema del dominio TPP presentan restricciones de trayectoria de estado que no son *deadlines*, principalmente mediante el uso del operador always; por ejemplo, (always (>= (stored goods1) (stored goods1))). Con respecto los dominios Storage (*IPC* 2006) y Openstacks (*IPC* 2006, *IPC* 2008, *IPC* 2011), tampoco se han podido ejecutar problemas porque la heurística LM-Cut resulta ser muy poco informativa para estos dominios lo que provoca una explosión combinatoria del árbol de búsqueda. TempLM no maneja dominios concurrentes lo que excluye dominios como Matchcellar, Tms y Turnandopen (*IPC* 2011).

A continuación se describen los dominios que se han probado en las distintas comparativas realizadas.

Driverlog (*IPC* 2002, *IPC* 2014). Este dominio se ha detallado a lo largo de esta memoria de tesis. El dominio consta de un conjunto de camiones, paquetes y conductores. Los camiones requieren un conductor para poder moverse entre carreteras y los conductores se pueden desplazar andando hasta una localización donde se encuentra un camión a través de una red de caminos. El objetivo es transportar los paquetes desde su lugar origen a su lugar destino.

ZenoTravel (*IPC* 2002). Se trata de un dominio de transporte en el que varias personas deben trasladarse mediante aviones entre distintas ciudades. Los aviones

consumen combustible y pueden volar en modo lento o rápido. Los desplazamientos rápidos consumen una mayor cantidad de combustible. Los pasajeros pueden embarcar o desembarcar de los aviones y los aviones pueden repostar combustible.

Depots (*IPC* 2002). Este dominio se ideó con el fin de comprobar el comportamiento de los planificadores al combinar las características de los dominios de Blocksworld y de Logistics. Es un dominio en el que los camiones pueden transportar cajas entre almacenes que disponen de *pallets* para apilar las cajas. Las cajas se pueden cargar y descargar de los camiones mediante grúas. Los camiones no tienen que cargar las cajas en un orden determinado.

Satellite (*IPC* 2002, *IPC* 2004, *IPC* 2014). En este dominio, se utiliza un conjunto de satélites para realizar diversas tareas de observación, recogida y envío de datos a una estación terrestre. Los satélites están equipados con diferentes instrumentos, cada uno con diferentes características. Esto conlleva diversas operaciones como dirigir los satélites hacia la dirección correcta, activar, desactivar y calibrar los instrumentos, y tomar imágenes. Los datos que generan los instrumentos se almacenan en el satélite hasta que se pueda establecer la comunicación con la estación terrestre. En la *IPC* 2004 se utilizan *TIL* para establecer las ventanas temporales durante las cuales se puede transmitir la información.

Pipesworld (*IPC* 2004, *IPC* 2006). Los planificadores controlan el flujo de los derivados del petróleo a través de una red de tuberías, obedeciendo a diversas limitaciones, tales como la compatibilidad del producto o restricciones de los depósitos. Un aspecto interesante del dominio es que, si uno inserta algo en el un extremo de un segmento de tubería, algo potencialmente completamente diferente sale en el otro extremo. Esto da lugar a varios fenómenos sutiles que pueden surgir en la creación de un plan.

Trucks (*IPC* 2006). Es un dominio de logística que consiste en repartir paquetes mediante camiones bajo ciertas restricciones. El espacio de carga de cada camión se organiza por áreas: un paquete se puede cargar o descargar en el área de un camión solo si las áreas que se encuentran situadas entre el área de carga/descarga y la puerta del camión están libres. En este dominio se establecen deadlines para el reparto de los paquetes y es importante encontrar planes de buena calidad. Sin embargo, para muchas instancias de problema de este dominio incluso encontrar simplemente un plan solución puede ser una tarea muy compleja.

Floortile (*IPC* 2011, *IPC* 2014). Este dominio, que se utilizó por primera vez en la *IPC* 2011 y posteriormente en la *IPC* 2014, consta de un conjunto de robots encargados de pintar las baldosas de un suelo. Los robots se pueden mover en cuatro direcciones (arriba, abajo, derecha e izquierda) y disponen de pinturas de varios colores que pueden alternar pero una baldosa solo puede pintarse de un color determinado. Por otro lado, a un robot solo le está permitido pintar la baldosa de arriba o abajo y, una vez pintada, no puede situarse encima de ella.

Parking (*IPC* 2011, *IPC* 2014). Este dominio es original de la competición de aprendizaje de la *IPC* 2008 pero se utilizó por primer vez en la competición de temporal en la *IPC* 2011. Los problemas de este dominio consisten en aparcar coches en una calle que dispone de N posiciones de aparcamiento y donde los coches pueden estar aparcados en doble fila pero no en triple fila. El objetivo es conseguir, a partir de una configuración inicial de los coches, otra configuración distinta mediante movimientos de los coches de una posición a otra de la calle. Los problemas de la competición contienen 2*(N-1) coches, dejando así una posición libre en la calle que garantiza la resolubilidad del problema.

Pegsol (*IPC* 2011). Este dominio modela un juego de solitario en el que se dispone de un tablero con agujeros algunos de los cuales tienen un palo y otros no. El objetivo es vaciar los palos del tablero dejando únicamente un palo en el agujero central del tablero. Para ello se puede realizar únicamente un movimiento: dados tres agujeros A1, A2 y A3, situados en una misma fila, y donde dos agujeros consecutivos contienen un palo (por ejemplo, A1 y A2), se puede pasar el palo de A1 a A3 eliminando el palo de A2.

Todos los problemas de las comparativas que se muestran en las siguientes secciones han sido ejecutados en una máquina Intel-Core-i5-3.2-GHz con 16GB-RAM. El tiempo de resolución se presenta en segundos y se ha restringido a 30 minutos en todos los casos.

6.3. Comparativa I: TempLM versus OPTIC, MIPS-XXL y SGPLAN5

En esta sección se realiza una comparativa entre las distintas configuraciones de TempLM y los planificadores MIPS-XXL, SGPLAN5 y OPTIC. Se trata de una comparativa preliminar con el fin de analizar la mejora que introduce la información del TLG durante el proceso de búsqueda con respecto a otros planificadores capaces de tratar problemas con deadlines. Concretamente, estudiaremos estas tres configuraciones:

- TempLM únicamente con el proceso de búsqueda A básico introducido en el Capítulo 5 sin utilizar información del TLG (denominado TempLM-Búsq)
- TempLM utilizando información procedente del TLG inicial para guiar el proceso de búsqueda (denominado TempLM-TLG)
- TempLM utilizando el proceso de *feedback* durante la búsqueda, es decir, generando nueva información para el TLG en cada iteración (denominado TempLM-FB).

Para realizar esta comparativa se generó una batería de *problemas clasificados* (Sección6.2), con el objetivo de medir el número de problemas *tight* y *loose* resueltos y el número de problemas irresolubles detectados por cada aproximación. Se generaron problemas para tres dominios particulares: Driverlog, Depots y ZenoTravel. La selección de estos tres dominios responde a las diferentes características de planificación que ofrece cada uno de ellos y a nuestro objetivo de evaluar la incorporación de *deadlines* en dominios de diferente naturaleza. Específicamente, el dominio Depots se caracteriza por disponer de muchos efectos colaterales, el dominio ZenoTravel permite conseguir un mismo objetivo con acciones que tienen distinta duración y el dominio Driverlog se caracteriza porque existen alternativas de distinta duración para conseguir un recurso en una localización determinada (por ejemplo, un conductor). Es decir, en el dominio Driverlog existen diferentes alternativas de satisfacer las condiciones de una acción (por ejemplo, disponer de un conductor para transportar un paquete) mientras que el dominioZenoTravel ofrece diferentes opciones de satisfacer un objetivo del problema (por ejemplo transportar un pasajero).

		TempLM					
Deadlines	#	Búsq	TLG	FB	MIPS-XXL	SGPLAN5	OPTIC
Driverlog	**16**						
Irresol.	6	0	4	5	2	0	3
Tight	5	5	5	4	1	1	2
Loose	5	5	5	3	1	3	4
Depots	**12**						
Irresol.	4	0	1	4	4	0	0
Tight	4	4	4	1	4	1	4
Loose	4	4	4	1	4	4	4
ZenoTravel	**19**						
Irresol.	7	0	5	6	2	0	2
Tight	6	6	6	6	2	0	3
Loose	6	6	6	6	2	4	5
Total	47	30	40	36	22	13	27

Tabla 6.1: Resumen de los resultados obtenidos

La Tabla 6.1 muestra los resultados obtenidos para las tres versiones de TempLM comentadas, MIPS-XXL, SGPLAN5 y OPTIC. SGPLAN5 es el planificador que resuelve un menor número de problemas, comparado con los otros planificadores. Los planes solución de SGPLAN5 para la mayoría de problemas que resuelve son soluciones de baja calidad; por ejemplo, el *makespan* medio de los planes devueltos por SGPLAN5 es un 20 % superior a la media del *makespan* de los planes de TempLM-TLG. No obstante, SGPLAN5 es el planificador que ofrece mejor rendimiento, con una media de 0.03 segundos para resolver un plan frente a los 15 segundos de la aproximación TempLM-TLG. Por otro lado, cabe destacar que SGPLAN5 no detecta ninguno de los problemas irresolubles y que en muchos casos detiene su ejecución inmediatamente y no reporta ningún resultado[2].

MIPS-XXL resuelve un mayor número de problemas que SGPLAN5, particularmente resuelve más problemas de tipo *tight* y, además, es capaz de detectar ocho problemas irresolubles, aunque para ello necesita agotar el espacio de búsqueda con el consiguiente coste computacional. Concretamente, MIPS-XXL necesita un tiempo medio de 395.56 segundos para la detección de problemas irresolubles frente a una media de 175.84 segundos de la aproximación TempLM-FB. En el dominio Depots MIPS-XXL resuelve todos los problemas *tight* y *loose* y detecta todos los problemas

[2]Esta anomalía fue comentada en el trabajo de Benton *et al.* [5] donde los autores de este trabajo realizaron un experimento en el que SGPLAN5 fue incapaz de resolver algunos problemas cuya definición original fue ligeramente modificada con una acción "dummy", es decir, sin efecto en la resolución del problema. Nuestra conclusión es que ese mismo problema se ha presentado en nuestros experimentos.

irresolubles.

Por otro lado, en la Tabla 6.1 podemos observar que OPTIC resuelve más problemas *tight* y *loose* que MIPS-XXL y SGPLAN5 aunque el número de problemas irresolubles detectados es ligeramente inferior al de MIPS-XXL. Sin embargo, OPTIC necesita un tiempo medio de 3.17 segundos frente a una media de 155.81 segundos de MIPS-XXL en aquellos problemas que ambos resuelven o en donde ambos detectan irresolubilidad. Por estos dos motivos, OPTIC será el planificador de referencia que utilizaremos en el resto de comparativas. Adicionalmente, podemos observar en la Tabla 6.1 que OPTIC resuelve menos problemas que las configuraciones de TempLM. Esto es especialmente notable en la detección de problemas irresolubles donde TempLM-TLG y TempLM-FB superan claramente a OPTIC (ver filas $Irresol.$ de la Tabla 6.1).

Analizando el comportamiento de las distintas configuraciones de TempLM, es decir TempLM-Búsq, TempLM-TLG y TempLM-FB. TempLM-Búsq no puede detectar ningún problema irresoluble, agotando el tiempo de cómputo disponible antes de agotar el espacio de búsqueda. Sin embargo, las configuraciones que utilizan el TLG, TempLM-TLG y TempLM-FB, detectan 10 y 15 problemas irresolubles, respectivamente. Especialmente destacable es el caso de TempLM-FB que es capaz de detectar 15 de los 17 problemas irresolubles gracias al proceso de *feedback*.

Con respecto a los problemas *tight* y *loose*, tanto TempLM-Búsq como TempLM-TLG resuelven el mismo número de problemas, mientras que TempLM-FB tiene más dificultades debido al coste del proceso de *feedback*. Este proceso no es suficientemente informativo para explorar eficientemente el espacio de búsqueda en estos dominios.

A tenor de estos resultados preliminares, el resto de comparativas se centrarán en las configuraciones TempLM-TLG y TempLM-FB, ya que TempLM-Búsq ha sido igualado o superado por éstas, y en el planificador OPTIC.

6.4. Comparativa II: beneficios de la utilización de landmarks y *feedback*

En esta sección se realiza una comparación de problemas de planificación con *deadlines* originalmente presentados en las competiciones, de *problemas no clasificados* y de *problemas clasificados* de cada uno de los dominios presentados en la Sección 6.2.

6.4.1. Problemas originales de las competiciones *IPC* 2004 e *IPC* 2006

	TempLM-TLG		TempLM-FB		OPTIC	
	Makespan	Tiempo	Makespan	Tiempo	Makespan	Tiempo
Satellite (*IPC* 2004)						
PF1	176.69	0.4	176.69	1.5	176.69	2.47
PF2	210.47	0.86	210.47	3.12	191.28	6.2
PF3	106.77	0.64	106.77	2.46	106.77	4.53
PF4	207.18	35.13	165.92	45.84	169.18	72.85
PF5	180.46	34.95	180.46	124.303	183.91	16.28
PF6	-	T.E.	145.72	307.89	-	T.E.
PF7	134.38	1467.77	-	T.E.	140.13	13.85
PF8	-	T.E.	-	T.E.	119.94	35.2
PipesWorld (*IPC* 2004)						
PF1	6	0.16	6	0.24	6	1.44
PF2	18	0.56	18	6.09	18	46.22
PF3	14	0.59	14	2.78	14	3.5
PF4	16	7.39	16	17.41	20	245
PF5	12	1.84	12	6.89	14	4.12
PF6	14	1.3	14	5.18	14	118.85
PF7	12	2.12	12	4.42	12	1179.8
PF8	14	3.09	14	6.57	14	4.29
PF9	18	22.46	18	71.88	18	4.5
PF10	22	1234.37	24	653.88	-	T.E.
Trucks (*IPC* 2006)						
PF1	843.2	0.03	-	T.E.	1582.4	3.97
PF2	1711.4	0.66	1711.4	3.97	3510.2	4.63
PF3	1470.1	3.83	-	T.E.	2043.5	5.16
PF4	2629.4	20.17	2672.7	14.52	3703.7	6.94
PF5	1671.6	349.92	-	T.E.	3476.8	22.94

Tabla 6.2: Resultados para los problemas originales de dominios de *IPC* 2004 e *IPC* 2006.

El concepto de *deadline* se introduce por primera vez en la *IPC* 2004 definiendo las restricciones temporales con $TILs$. Posteriormente, se introduce el lenguaje

PDDL3.0 en la *IPC* 2006 donde se organizó una competición de dominios con restricciones definiendo los *deadlines*, adicionalmente, con el operador modal within. Para realizar la comparativa de esta sección se han utilizado los dominios Satellite y Pipesworld de la *IPC* 2004 por ser los únicos que utilizaban TIL y el dominio Trucks de la *IPC* 2006 por ser el único que tenía restricciones expresadas con within. El resto de dominios de estas dos competiciones utilizan operadores modales que OPTIC no maneja, como por ejemplo el operador modal always-within en el dominio Pipesworld de la *IPC* 2006.

La Tabla 6.2 muestra los resultados obtenidos por las dos aproximaciones de TempLM, TempLM-TLG sin *feedback* y TempLM-FB con *feedback*, y OPTIC para los 10 primeros problemas de cada uno de los tres dominios que alguna de las tres aproximaciones resuelve. En la tabla se muestra la duración del plan solución (*makespan*) y el tiempo de cómputo para la obtención de dicha solución.

En general, TempLM-TLG y TempLM-FB obtienen planes de menor *makespan* en menos tiempo que OPTIC. Concretamente, en el dominio Satellite, el *makespan* de los planes obtenidos por las tres aproximaciones es similar. Las aproximaciones TempLM tardan menos en el 50 % de los problemas en comparación con OPTIC. En este dominio, TempLM-FB presenta un comportamiento desigual ya que para algunos problemas obtiene planes de buena calidad en poco tiempo y en otros casos es incapaz de resolver el problema.

En el dominio Pipesworld la tendencia es similar. Todos los planes obtenidos por TempLM-TLG son de igual (en el 70 % de los casos) o mejor *makespan* (en el 30 % de los casos) que los obtenidos con las otras dos aproximaciones. Respecto al tiempo de cómputo, TempLM-TLG emplea menos tiempo en el 90 % de los problemas y TempLM-FB resuelve el 70 % de los problemas en menos tiempo que OPTIC. Más concretamente, TempLM-FB tarda 4.42 segundos en el problema PF7 frente a los 1179.8 de OPTIC o en el problema PF10 TempLM-FB tarda 653.88 segundos y OPTIC no es capaz de resolverlo.

En el dominio Trucks, el *makespan* de los planes obtenidos por OPTIC es significativamente mayor que los obtenidos por TempLM-TLG. Tal y como muestran los resultados de TempLM-FB, la información que aporta el proceso de *feedback* no compensa la explosión combinatoria que se produce en este dominio, lo que impide que pueda resolver los problemas PF1, PF3 y PF5.

En resumen, TempLM-TLG supera claramente tanto a TempLM-FB como a OPTIC en este conjunto de problemas. Esto sugiere que la información que aporta el TLG inicial es muy útil para la poda de nodos durante el proceso de búsqueda. Sin embargo, el proceso de refinamiento del TLG en cada nodo del árbol de búsqueda supone un elevado coste, por lo que TempLM-FB no es capaz de resolver varios de estos problemas en los 30 minutos establecidos.

	TempLM-TLG		TempLM-FB		OPTIC	
	Makespan	Tiempo	Makespan	Tiempo	Makespan	Tiempo
Depots						
D01T	27	**0.23**	27	4.01	27	2.72
D02T	34	36.73	-	T.E.	34	6.79
D03T	34	41.65	-	T.E.	34	6.45
D04T	51	4.5	-	T.E.	51	4.05
Driverlog						
D01T	**91**	**0.05**	**91**	0.12	92	1.31
D02T	47	5.18	**40**	26.25	**40**	**11.76**
D03T	**51**	**58.4**	**51**	163.82	80	309.37
D04T	49	69.56	49	141.74	49	**23.42**
D05T	**98**	302.85	**98**	**131.02**	144	130.58
D06T	128	264.72	128	1108.77	**126**	**158.84**
D07T	**37**	**24.90**	**37**	55.79	-	T.E.
D08T	**98**	**97.74**	**98**	344.28	120	389.63
D09T	**76**	**24.61**	**76**	94.782	-	T.E.
D10T	49	81.69	49	142.73	49	**21.46**
ZenoTravel						
Z01T	592	**0.12**	592	0.45	592	3.45
Z02T	592	**0.11**	592	0.38	592	3.43
Z03T	**393**	**8.8**	**393**	18.12	-	T.E.
Z04T	542	**17.52**	542	34.07	**536**	625.35
Z05T	**522**	**24.95**	529	8.49	536	47.03
Z06T	**320**	**34.36**	**320**	38.29	-	T.E.
Z07T	**333**	**26.54**	**333**	37.31	-	T.E.
Z08T	**665**	**35.14**	**665**	54.21	-	T.E.
Z09T	**430**	**1144.29**	559	1547.08	576	1729.69
Z10T	**665**	380.92	**665**	**33.43**	-	T.E.
Rovers						
R01T	47	0.50	47	**0.45**	47	2.1
R02T	57	**2.10**	57	6.59	57	3.02
R03T	50	**0.07**	50	0.66	**45**	270.04
R04T	**93**	**0.8**	**93**	4.25	-	T.E.
R05T	**45**	**0.31**	**45**	0.82	104	1021.41
R06T	**125**	22.28	**125**	**10.54**	-	T.E.
R07T	**73**	10.15	**73**	22.16	78	**9.34**
R08T	**88**	**5.76**	**88**	6.49	93	33.18
R09T	**145**	**30.29**	**145**	53.074	-	T.E.
R10T	-	T.E.	**115**	**14.98**	130	226.3

Tabla 6.3: Resultados de problemas *tight* en dominios de la *IPC* 2002

6.4.2. Problemas clasificados como *tight*

Las Tablas 6.3 y 6.4 muestran los resultados obtenidos para problemas de planificación clasificados como *tight*.

Concretamente, en el dominio Depots (Tabla 6.3), TempLM-FB sólo resuelve uno de los problemas. TempLM-TLG y OPTIC obtienen planes de idéntico *makespan*,

	TempLM-TLG		TempLM-FB		OPTIC	
	Makespan	Tiempo	Makespan	Tiempo	Makespan	Tiempo
Trucks						
T01T	**843.2**	**0.038**	-	T.E.	845.2	21.53
T02T	1711.4	**0.59**	1711.4	1.89	1711.4	21.71
T03T	**1470.1**	**3.97**	-	T.E.	1482.9	15.16
T04T	**2629.4**	**14.54**	-	T.E.	2770.2	30.39
T05T	**1671.6**	**168.86**	-	T.E.	1673.6	300.16
Pegsol						
PE01T	7	**0.4**	7	3.26	7	21.46
PE02T	**6**	**1.01**	**6**	1.94	8	56.92
PE03T	7	**0.49**	7	4.13	7	53.28
PE04T	**9**	**28.86**	**9**	107.61	-	T.E.
PE05T	**8**	**25.46**	**8**	47.66	-	T.E.
PE06T	10	**22.9**	10	167.47	10	236.86
PE07T	7	**0.1**	7	0.99	7	9.13
PE08T	7	**1.94**	7	14.73	7	118.56
PE09T	7	**4.01**	7	25.47	8	1300.06
PE10T	10	**84.84**	10	514.46	-	T.E.
Floortile						
F01T	**15**	**0.68**	**15**	3.19	-	T.E.
F02T	**12**	**0.92**	**12**	4.97	-	T.E.
F03T	**22**	**4.02**	**22**	24.61	-	T.E.
F04T	**19**	**1.03**	**19**	4.66	-	T.E.
F05T	**19**	**1.3**	**19**	6.58	-	T.E.
F06T	**20**	**1.03**	**20**	4.83	-	T.E.
F07T	-	T.E.	**22**	**50.69**	-	T.E.
F08T	**20**	**2.48**	**20**	5.88	-	T.E.
F09T	**21**	**5.31**	**22**	11.73	-	T.E.
F10T	-	T.E.	**19**	**49.37**	-	T.E.

Tabla 6.4: Resultados de problemas *tight* en dominios de la *IPC* 2006, *IPC* 2008 e *IPC* 2011

aunque el tiempo de cómputo de OPTIC es en 3 de los 4 problemas menor que el de TempLM-TLG. En este dominio OPTIC resuelve dos problemas más, concretamente D06T y D08T, que no resuelve ninguna de las aproximaciones de TempLM.

En el dominio Driverlog (Tabla 6.3), OPTIC es la única aproximación que no resuelve todos los problemas. Para el resto de problemas que resuelven las tres aproximaciones, TempLM-FB obtiene la solución de mejor *makespan* en el 87 % de los casos, TempLM-TLG en el 75 % y OPTIC en el 50 % de los casos. Por ejemplo, en el problema D02T TempLM-FB mejora el *makespan* respecto a TempLM-TLG aunque a costa de un mayor tiempo de cómputo y en el problema D06T OPTIC es la aproximación que obtiene la mejor solución.

En el dominio Zenotravel (Tabla 6.3), el rendimiento de OPTIC es muy inferior al de las dos aproximaciones de TempLM, resolviendo únicamente el 40 % de los problemas. El *makespan* de los planes resultantes es similar en las tres aproximaciones excepto en el problema Z09T donde claramente TempLM-TLG obtiene una solución mucho mejor. Por otro lado, es significativo que OPTIC es un 50 % más lento que TempLM-TLG en la resolución de estos problemas.

En el dominio Rovers (Tabla 6.3), TempLM-FB resuelve todos los problemas, OPTIC resuelve el 70 % de los problemas y TempLM-TLG no consigue resolver uno de los problemas. De los problemas que resuelven las tres aproximaciones, el *makespan* es similar excepto en el problema R05T donde OPTIC obtiene una solución mucho peor. También es significativo que OPTIC es un 98 % más lento que TempLM-TLG en la resolución de estos problemas.

En el dominio Trucks (Tabla 6.4) ninguna de las tres aproximaciones consiguió resolver más problemas de los que se muestran en la tabla ya que se aumentan el número de camiones y el número de areas de carga de los mismos lo que provoca un aumento en la combinatoria de las posibilidades de planificación. En este dominio TempLM-FB sólo resuelve uno de los problemas mientras que TempLM-TLG y OPTIC obtienen un *makespan* similar en los planes que ambas aproximaciones resuelven. Al igual que en el resto de dominios, OPTIC es más lento que TempLM-TLG en todos los problemas.

En el dominio Pegsol (Tabla 6.4), OPTIC sólo resuelve el 70 % de los problemas y las tres aproximaciones obtienen planes de similar *makespan* en aquellos problemas resueltos por los tres planificadores. Especialmente significativo es que OPTIC es un 98 % más lento que TempLM-TLG.

En el dominio Floortile (Tabla 6.4), OPTIC no resuelve ningún problema mientras que TempLM-TLG resuelve 8 problemas. El *makespan* es idéntico en los problemas que ambas aproximaciones resuelven. Por otro lado, el tiempo de cómputo de TempLM-FB es un 76 % mayor que el de TempLM-TLG.

En general, a raíz de los datos de las Tablas 6.3 y 6.4 podemos concluir que la aproximación que obtiene mejores resultados, tanto con respecto al *makespan* de los planes calculados como al tiempo de cómputo, es TempLM-TLG. TempLM-FB consigue obtener planes de la misma calidad, pero siempre a costa de un tiempo de cómputo mayor. En cuanto a OPTIC mejora el *makespan* respecto a las dos aproximaciones de TempLM sólo en tres problemas. Respecto al número de problemas resueltos, TempLM-TLG resuelve el 95 % de los problemas, TempLM-FB resuelve el 88 % de los problemas y OPTIC resuelve el 59 % de los problemas.

La Tabla 6.5 muestra el número de nodos generados y expandidos por las configuraciones TempLM-TLG y TempLM-FB en la resolución de estos problemas *tight*. No se muestran resultados de los dominios Depots y Trucks, ya que TempLM-FB

solamente es capaz de resolver un problema en cada dominio, por lo que los valores no serían comparables. En el resto de los casos, solo se han tenido en cuenta aquellos problemas resueltos por ambas configuraciones. Tal y como muestra dicha tabla, el número de nodos generados y expandidos por la configuración TempLM-FB es siempre menor comparado con la configuración TempLM-TLG, concretamente el número de nodos generados y expandidos por TempLM-FB es un 57 % y un 54 %, respectivamente, menor que TempLM-TLG. Sin embargo, el tiempo requerido para generar un nuevo nodo en TempLM-TLG es mucho menor, lo que permite resolver los problemas más eficientemente. Aún así, en el dominio ZenoTravel, donde la reducción del número de nodos generados/expandidos es alrededor del 50 %, los tiempos de cómputo de TempLM-FB son cercanos a los de TempLM-TLG. Este dato sugiere que sería necesaria una gran reducción en el espacio de búsqueda para que la utilización del proceso de *feedback* compense en los problemas *tight*[3].

	TempLM-TLG		TempLM-FB	
	expandidos	generados	expandidos	generados
Driverlog Tight	78851	289003	**51115**	**166063**
ZenoTravel Tight	30184	283006	**14653**	**161912**
Satellite Tight	5090	191338	**1260**	**17542**
Pipesworld Tight	100553	100553	**18920**	**18920**
Pegsol Tight	16550	77714	**13007**	**62190**
Floortile Tight	2513	9024	**1844**	**6650**
Rovers Tight	2291	16769	**1462**	**12670**

Tabla 6.5: Número de nodos generados y expandidos para problemas clasificados como *tight*

6.4.3. Problemas clasificados como irresolubles

Las Tablas 6.6, 6.7 y 6.8 muestran los resultados para problemas irresolubles, en los que al menos un planificador fue capaz de detectar dicha irresolubilidad. Junto al tiempo computacional (en segundos) requerido por cada aproximación, en el caso de las dos aproximaciones de TempLM se muestra además la etapa en la que se detectó la irresolubilidad: durante la construcción del $TRPG$, durante la propagación de las restricciones temporales en el TLG o durante la búsqueda.

[3]El proceso actual de copia del TLG de un nodo al siguiente en el árbol de búsqueda es un proceso muy costoso. Una posibilidad para incrementar el rendimiento de TempLM-FB sería mejorar este proceso de copia.

	TempLM-TLG		TempLM-FB		OPTIC	
	Detección	Tiempo	Detección	Tiempo	Detección	Tiempo
Depots						
D01U	-	T.E.	Busq.	**6.99**	Irresol.	0.14
D02U	-	T.E.	Busq.	**595.62**	Irresol.	0.13
D03U	-	T.E.	Busq.	**628.37**	-	T.E.
D04U	TRPG	0.01	TRPG	0.01	-	T.E.
Driverlog						
D01U	TRPG	0.01	TRPG	0.01	Irresol.	0.14
D02U	TRPG	0.01	TRPG	0.01	Irresol.	0.13
D03U	Busq.	295.13	Search	**9.33**	-	T.E.
D04U	-	T.E.	Busq.	**679.35**	-	T.E.
D05U	-	T.E.	Busq.	**1095.74**	-	T.E.
D06U	TRPG	0.01	TRPG	0.01	Irresol.	0.16
D07U	Busq.	**395.29**	Busq.	792.49	-	T.E.
D08U	TRPG	0.01	TRPG	0.01	Irresol.	0.11
D09U	Busq.	351.14	Busq.	**39.11**	-	T.E.
D10U	Busq.	346.03	Busq.	**34.99**	-	T.E.
ZenoTravel						
Z01U	TRPG	0.01	TRPG	0.01	Irresol.	0.14
Z02U	Busq.	1.11	Busq.	**0.27**	-	T.E.
Z03U	-	T.E.	Busq.	**14.41**	-	T.E.
Z04U	Busq.	704.15	Busq.	**31.37**	-	T.E.
Z05U	Busq.	592.26	Busq.	**5.16**	-	T.E.
Z06U	Busq.	575.35	Busq.	**4.77**	-	T.E.
Z07U	-	T.E.	Busq.	**104.71**	-	T.E.
Z08U	TRPG	0.01	TRPG	0.01	Irresol.	0.16
Z09U	-	T.E.	Busq.	**96.46**	-	T.E.
Z10U	-	T.E.	Busq.	**36.25**	-	T.E.
Rovers						
R01U	TLG	0.01	TLG	0.01	-	T.E.
R02U	Busq.	886.62	Busq.	**77.92**	-	T.E.
R03U	TLG	0.01	TLG	0.01	-	T.E.
R04U	TLG	0.01	TLG	0.01	-	T.E.
R05U	TLG	0.01	TLG	0.01	-	T.E.
R06U	TRPG	0.01	TRPG	0.01	Irresol.	0.14
R07U	TRPG	0.01	TRPG	0.01	Irresol.	0.17
R08U	TLG	0.01	TLG	0.01	-	T.E.
R09U	Busq.	553.67	Busq.	**89.94**	-	T.E.
R10U	TRPG	0.01	TRPG	0.01	-	T.E.

Tabla 6.6: Resultados de problemas irresolubles de dominios de la *IPC* 2002

De los resultados de las tres tablas, TempLM-FB muestra los mejores resultados en la detección de problemas irresolubles, siendo capaz de detectar todos ellos mientras que OPTIC tiene dificultades para trabajar con este tipo de problemas, ya que sólo ha podido detectar el 28 % de los problemas irresolubles. Es significativo el hecho de que 16 de los 21 problemas en los que OPTIC detecta irresolubildad se corresponden con problemas en los que TempLM-FB detecta la irresolublidad durante la construcción del $TRPG$, ya que OPTIC realiza un razonamiento similar. Por otro lado, en dos problemas del dominio Depots (Tabla 6.6) y en uno del dominio Pegsol (Tabla 6.8), OPTIC identifica estos problemas irresolubles que TempLM-FB detecta durante el proceso de búsqueda y en dos problemas del dominio Pegsol que OPTIC detecta como irresolubles, TempLM-FB los identifica durante la propagación de las restricciones en el TLG. Si comparamos las dos configuraciones de TempLM, los resultados revelan que TempLM-TLG fue capaz de detectar el 84 % de problemas

	TempLM-TLG		TempLM-FB		OPTIC	
	Detección	Tiempo	Detección	Tiempo	Detección	Tiempo
Pipesworld-04						
PI01U	TRPG	**0**	TRPG	**0**	Irresol.	3.45
PI02U	Busq.	**0.85**	Busq.	1.92	-	T.E.
PI03U	Busq.	31.79	Busq.	**27.50**	-	T.E.
PI04U	Busq.	54.99	Busq.	**35.26**	-	T.E.
PI05U	Busq.	397.79	Busq.	**190.22**	-	T.E.
PI06U	Busq.	368.97	Busq.	**133.28**	-	T.E.
PI07U	Busq.	135.78	Busq.	**33.38**	-	T.E.
PI08U	Busq.	1120.21	Busq.	**81.6**	-	T.E.
PI09U	-	TE	Busq.	**157.41**	-	T.E.
PI10U	-	TE	Busq.	**282.34**	-	T.E.
Satellite						
S01U	TRPG	0.01	TRPG	0.01	Irresol.	0.28
S02U	Busq.	914.21	Busq.	1425.13	-	T.E.
S03U	TRPG	0.01	TRPG	0.01	Irresol.	0.13
S04U	TLG	0.01	TLG	0.01	-	T.E.
S05U	TRPG	0.01	TRPG	0.01	Irresol.	0.17
S06U	TRPG	0.01	TRPG	0.01	-	T.E.
S07U	TRPG	0.01	TRPG	0.01	Irresol.	0.2
S08U	TLG	0.01	TLG	0.01	-	T.E.
S09U	Busq.	0.07	Busq.	0.1	-	T.E.
S10U	TRPG	0.01	TRPG	0.01	Irresol.	0.11

Tabla 6.7: Resultados de problemas irresolubles para dominios de la *IPC* 2004

irresolubles mientras que TempLM-FB los detectó todos.

Con respecto al tiempo de computación, TempLM-FB mejora a OPTIC en todos los problemas excepto en dos problemas del dominio Pegsol. Respecto a TempLM-TLG, TempLM-FB mejora o iguala el tiempo de computación en el 87 % de los problemas irresolubles detectados por ambas aproximaciones de TempLM.

La Tabla 6.9 muestra el número de nodos generados y expandidos por las dos configuraciones de TempLM en los problemas irresolubles que ambas aproximaciones han detectado. No se muestran resultados del dominio Depots ya que TempLM-TLG solamente es capaz de detectar un problema irresoluble, por lo que los valores no serían comparables. Tal y como muestra la Tabla 6.9, el número de nodos generados y expandidos por la configuración TempLM-FB es siempre mucho menor comparado con la configuración TempLM-TLG, concretamente un 74 % menor. Por dicho motivo, TempLM-FB es capaz de detectar un mayor número de problemas irresolubles, al reducir significativamente el espacio de búsqueda. Se observa que en el caso de los problemas irresolubles, la reducción en el número de nodos generados/expandidos es mucho mayor que en el caso de los problemas *tight* (ver Tabla 6.5). Esto se debe a

	TempLM-TLG		TempLM-FB		OPTIC	
	Detección	Tiempo	Detección	Tiempo	Detección	Tiempo
Pegsol						
PE01U	TLG	**0.01**	TLG	**0.01**	Irresol.	87.2
PE02U	Busq.	**8.86**	Busq.	9.17	-	T.E.
PE03U	Busq.	**71.69**	Busq.	946.65	-	T.E.
PE04U	Busq.	**63.52**	Busq.	807.14	-	T.E.
PE05U	Busq.	**14.21**	Busq.	150.32	Irresol.	27.53
PE06U	TRPG	0.01	TRPG	0.01	Irresol.	0.11
PE07U	TRPG	0.01	TRPG	0.01	Irresol.	0.11
PE08U	TLG	**0.01**	TLG	**0.01**	Irresol.	183.09
PE09U	Busq.	**3.27**	Busq.	50.43	-	T.E.
PE10U	Busq.	**72.77**	Busq.	992.32	-	T.E.
Floortile						
F01U	Busq.	**441.49**	Busq.	525.28	-	T.E.
F02U	Busq.	186.67	Busq.	**77.25**	-	T.E.
F03U	Busq.	303.27	Busq.	**246.24**	-	T.E.
F04U	Busq.	215.75	Busq.	**119.86**	-	T.E.
F05U	Busq.	251.63	Busq.	**168.54**	-	T.E.
F06U	TLG	**0.01**	TLG	**0.01**	-	T.E.
F07U	-	T.E.	Search	**1254.86**	-	T.E.
F08U	Busq.	300.66	Busq.	**290.63**	-	T.E.
F09U	Busq.	232.22	Busq.	**136.81**	-	T.E.
F10U	Busq.	369.11	Busq.	**295.33**	-	T.E.

Tabla 6.8: Resultados de problemas irresolubles para dominios de la *IPC* 2008 e *IPC* 2011

que las restricciones más estrictas de los problemas irresolubles junto con el cálculo sucesivo del TLG en cada nodo permite añadir información muy útil para podar nodos del espacio de búsqueda.

6.4.4. Problemas no clasificados

En esta sección se muestran los resultados obtenidos para los problemas cuyos *deadlines* han sido generados de forma aleatoria y se desconoce a priori si el problema es irresoluble, *tight* o *loose*.

La Tabla 6.10 muestra el número total de problemas que se generaron para cada dominio, la columna $Totalresueltos$ muestra el número de problemas que fueron resueltos por al menos uno de los tres planificadores y, por último, se muestra cuántos de estos problemas resueltos resultaron ser *loose*, *tight* o irresolubles, respectivamente. Las Tablas 6.11 y 6.12 muestran el número total de problemas resueltos por cada planificador para cada uno de los dominios propuestos.

Como podemos observar en las Tablas 6.11 y 6.12, TempLM-TLG resuelve más

	TempLM-TLG		TempLM-FB	
	expandidos	generados	expandidos	generados
Driverlog Irresol.	202626	202626	**91047**	**91047**
ZenoTravel Irresol.	122865	122865	**12903**	**12903**
Satellite Irresol.	38222	38222	**26060**	**26060**
Pipesworld Irresol.	100553	100553	**18920**	**18920**
Pegsol Irresol.	59147	59147	**56620**	**56620**
Floortile Irresol.	456292	456292	**82441**	**82441**
Rovers Irresol.	151545	151545	**8132**	**8132**

Tabla 6.9: Número de nodos generados y expandidos en los problemas irresolubles

Domain	Total Generados	Total Resueltos	Total Loose	Total Tight	Total Irresolubles
Driverlog	60	30	22	6	2
Depots	60	20	10	10	0
ZenoTravel	60	22	11	10	1
Parking	60	31	22	9	0
Pegsol	60	60	37	15	8
Floortile	40	15	11	4	0
Rovers	51	46	31	15	0
Satellite	60	20	10	10	0
Total	451	244	154	79	11

Tabla 6.10: Número de problemas resueltos de cada dominio

instancias en los tres tipos de problemas. OPTIC obtiene mejores resultados en los problemas *loose* pero su rendimiento empeora cuando se trata de problemas *tight* o irresolubles. En cambio, el comportamiento de TempLM-FB es el opuesto de OPTIC ya que su rendimiento mejora cuanto más ajustados son los problemas. Los resultados de estas tablas también muestran que el rendimiento de los tres planificadores varia en función de los dominios. Así, los resultados de OPTIC son mejores en los dominios en los que existen varios planes alternativos para un problema como ocurre en los dominios Driverlog, Parking o Satellite. En cambio, TempLM-TLG y TempLM-FB presentan mejores resultados en los dominios en los que existen muchas interacciones entre los objetivos como son los dominios Pegsol o Floortile.

Dados estos resultados, se han generado unas gráficas (que se muestran en las figuras de 6.1 a 6.10) que muestran una comparativa entre TempLM-TLG, TempLM-FB y OPTIC en los problemas que resuelven los tres planificadores, tanto en *makespan* de los planes como en tiempo de cómputo. Los dominios que se han utilizado en dicha comparativa son Driverlog, ZenoTravel, Rovers, Satellite y Pegsol, donde el número

de problemas resueltos por ambas aproximaciones es significativo.

Se observa como, en general, TempLM (en cualquiera de sus dos versiones) es capaz de obtener planes de menor *makespan* en menos tiempo. En muchos casos el *makespan* de las aproximaciones TempLM-TLG y TempLM-FB es similar y no se distingue en las gráficas. Concretamente, en el dominio Driverlog (Figuras 6.3 y 6.4), TempLM-TLG es capaz de obtener planes de igual o mejor calidad en todos los casos. El tiempo de cómputo de OPTIC es menor a TempLM-TLG en muchos casos, aunque el tiempo de cómputo de TempLM-TLG permanece estable entre 0 y 400 segundos, mientras que en el caso de OPTIC o TempLM-FB llega a alcanzar en varias ocasiones valores por encima de 500 segundos.

En el dominio ZenoTravel (Figuras 6.5 y 6.6), para las nueve primeras instancias, los resultados obtenidos por todas las aproximaciones son prácticamente idénticos. A partir de la instancia 10, TempLM-TLG obtiene planes de mejor calidad con un tiempo de cómputo menor.

En el dominio Satellite (Figuras 6.9 y 6.10), para las nueve primeras instancias, los resultados obtenidos por todas las aproximaciones son prácticamente idénticos. A partir de la instancia 10, TempLM-TLG obtiene planes de mejor calidad aunque con un tiempo de cómputo mayor.

En el dominio Rovers (Figuras 6.7 y 6.8), los resultados obtenidos por las aproximaciones TempLM son prácticamente idénticas y de mejor calidad que las que obtiene OPTIC. Respecto a los tiempos de cómputos son similares para las tres aproximaciones excepto para un pico en OPTIC y otro en TempLM-FB.

Por último, en el dominio Pegsol (Figuras 6.1 y 6.2), la calidad de los planes calculados por TempLM-TLG es siempre igual o mejor que la de los planes calculados por OPTIC. Además, el rendimiento de TempLM-TLG siempre mejora al de OPTIC, que en algunos casos es significativamente peor.

Estos resultados vienen a confirmar la tendencia de que TempLM funciona mejor que OPTIC en estos dominios.

Dominio	Loose			Tight		
	TempLM-TLG	TempLM-FB	OPTIC	TempLM-TLG	TempLM-FB	OPTIC
Driverlog	16	14	18	6	6	6
Depots	7	2	5	10	4	5
ZenoTravel	10	9	9	11	10	8
Parking	4	3	22	5	4	6
Pegsol	37	32	23	12	12	15
Floortile	11	11	0	4	4	0
Rovers	24	23	24	14	13	13
Satellite	7	6	9	9	7	8
Total	116	100	110	71	60	61

Tabla 6.11: Número de problemas *tight* y *loose* resueltos de cada dominio

Dominio	Irresoluble		
	TempLM-TLG	TempLM-FB	OPTIC
Driverlog	2	2	1
Depots	0	0	0
ZenoTravel	1	1	1
Parking	0	0	0
Pegsol	8	7	0
Floortile	0	0	0
Rovers	0	0	0
Satellite	0	0	0
Total	11	11	2

Tabla 6.12: Número de problemas irresolubles detectados de cada dominio

6.5. Comparativa III: comparativa con una nueva aproximación que utiliza landmarks temporales

En la reciente conferencia ICAPS-2015 se presentó otro enfoque que también utiliza *landmarks temporales* [66]. A diferencia de TempLM, que se centra principalmente en la resolución de problemas temporales altamente restringidos, este nuevo enfoque se centra en la solución de los problemas temporales concurrentes. Se definen dos tipos de *landmarks temporales*: *landmarks temporales* de acción y *landmarks temporales* de proposición. Un *landmark temporal* de acción consiste en un conjunto de eventos (comienzo/final de las acciones) y el punto de tiempo en el que debe producirse cada uno de estos eventos. Por otro lado, un *landmark temporal* de proposición es un conjunto de proposiciones que deben ser ciertos durante determinados

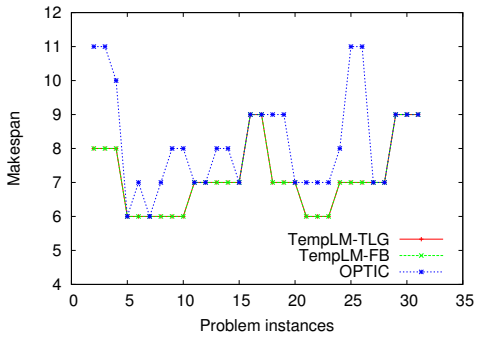

Figura 6.1: Pegsol (makespan).

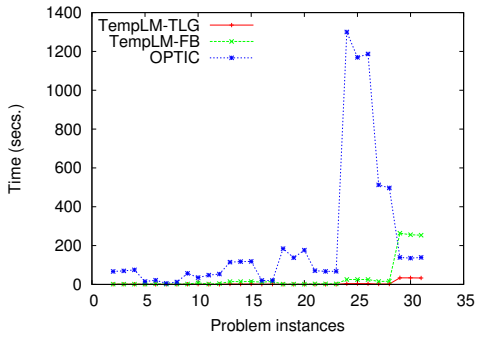

Figura 6.2: Pegsol (time).

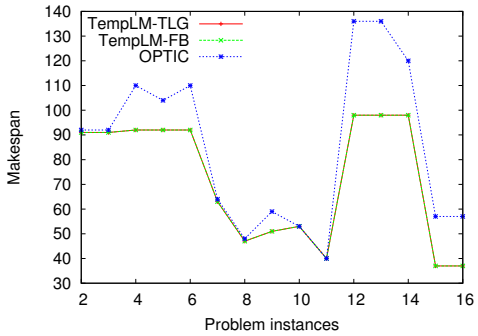

Figura 6.3: Driverlog (makespan).

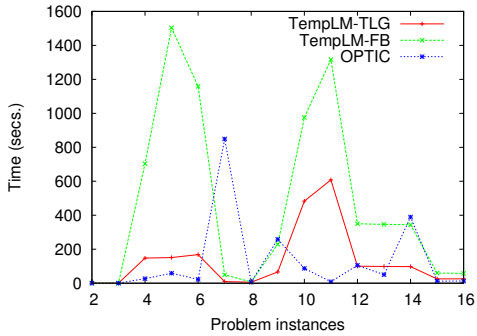

Figura 6.4: Driverlog (time).

intervalos de tiempo. Esta aproximación también utiliza una *Simple Temporal Network* (STN) con un conjunto de restricciones temporales simples entre los puntos de tiempo asociados a los *landmarks*. Por tanto, a partir de un problema de planificación (codificado en un fichero de dominio y otro fichero de problema en $PDDL$), se extraen los *landmarks* de acción y de proposición y se construye la STN. Esta información se compila en unos nuevos ficheros de dominio y problema que pueden ser utilizados por cualquier planificador temporal. Los resultados empíricos presentados en el trabajo de Karpas *et al.* [66] muestran que los *landmarks temporales* permiten mejorar el rendimiento de planificadores temporales cuando trabajan con problemas con interacciones temporales complejas.

Con el fin de comparar el rendimiento de este enfoque con TempLM, solicitamos a los autores que generaran los ficheros de dominio y de problema de diversas instancias seleccionadas y hemos utilizado OPTIC para resolver dichos problemas. Concretamente, se seleccionaron las siguientes instancias de los dominios Driverlog,

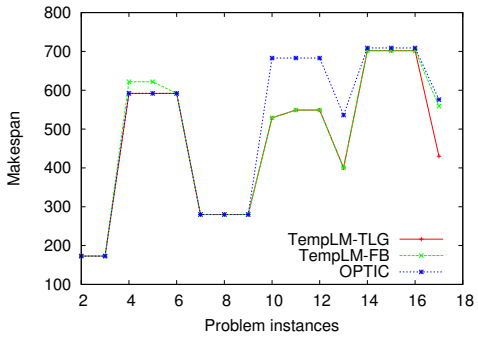

Figura 6.5: Zeno (makespan).

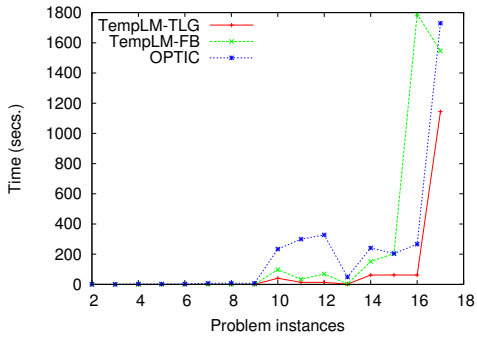

Figura 6.6: Zeno (time).

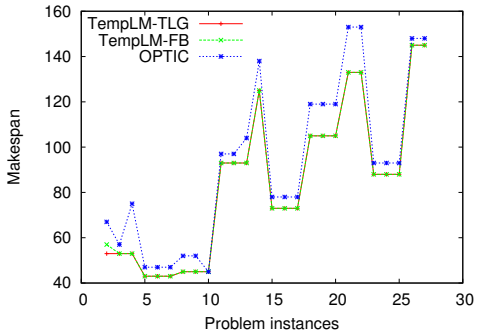

Figura 6.7: Rovers (makespan).

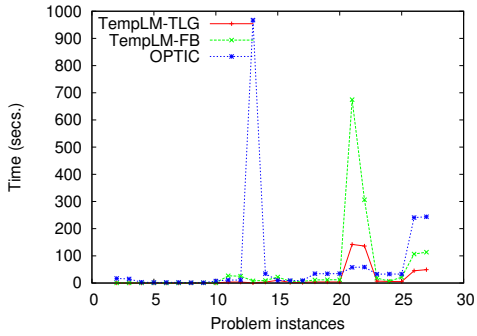

Figura 6.8: Rovers (time).

Zeno y Floortile: D03T, D04U, Z05T, Z07U, F03T, F05U y F06U de las Tablas 6.3, 6.4, 6.6 y 6.8. A partir del fichero de dominio y de problema de cada una de ellas, se generaron 3 pares de ficheros dominio-problema: para *landmarks* individuales, para *landmarks disyuntivos* con hasta cuatro proposiciones y para *landmarks disyuntivos* sin restricción en el número de proposiciones.

La Tabla 6.13 muestra el tiempo de cómputo en segundos de encontrar una solución para problemas *tight* (T) o de detectar que el problema no tiene solución para problemas irresolubles (U). Se han utilizado las aproximaciones TempLM-TLG, TempLM-FB y OPTIC. Se ha ejecutado OPTIC con los ficheros compilados que se han obtenido del enfoque de Karpas *et al.*: OPTIC-enrich-1 utiliza únicamente *landmarks* individuales, OPTIC-enrich-4 utiliza *landmarks* individuales y *landmarks disyuntivos* con hasta 4 proposiciones y OPTIC-enrich-inf utiliza *landmarks* individuales y *landmarks disyuntivos* sin restricción en el número de proposiciones.

Como puede observarse de los resultados de la Tabla 6.13 este enfoque no es capaz

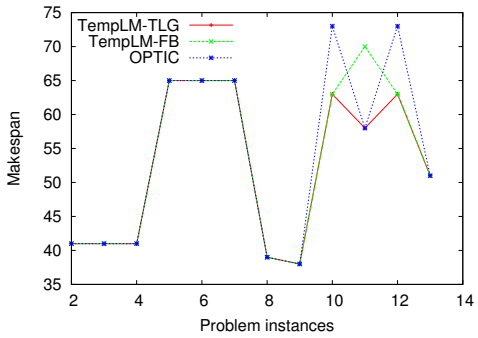

Figura 6.9: Satellite (makespan).

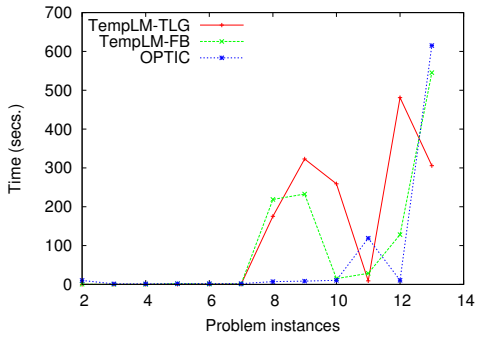

Figura 6.10: Satellite (time).

Dominio	TempLM-TLG	TempLM-FB	OPTIC	OPTIC enrich-1	OPTIC enrich-4	OPTIC enrich-inf
Driverlog						
D03T	58.4	163.82	309.37	T.E.	T.E.	T.E.
D04U	T.E.	679.35	T.E.	T.E.	T.E.	T.E.
ZenoTravel						
Z05T	24.95	8.49	47.01	T.E.	T.E.	T.E.
Z07U	T.E.	104.71	T.E.	T.E.	T.E.	T.E.
Floortile						
F03T	4.02	24.61	T.E.	1315.16.	T.E.	T.E.
F05U	251.63	168.54	T.E.	T.E.	T.E.	T.E.
F06U	0.01	0.01	T.E.	T.E.	T.E.	T.E.

Tabla 6.13: Instancias de problemas utilizadas con el enfoque de Karpas *et al.*

de mejorar a ninguna de las configuraciones de TempLM. Además, con respecto a OPTIC, se aprecia una mejora en el resultado para la instancia F03T, ya que OPTIC no encuentra solución y OPTIC-enrich-1 sí resuelve el problema, aunque con un coste temporal mucho mayor que el de TempLM. De acuerdo con este experimento, este enfoque no es tan eficiente en la resolución de problemas altamente restringidos o para detectar problemas irresolubles.

6.6. Conclusiones

En esta sección se han mostrado diferentes experimentos realizados con el fin de analizar el comportamiento de las dos configuraciones de TempLM, TempLM-TLG y TempLM-FB, en problemas con restricciones ajustadas (*tight*) y en problemas irresolubles.

En primer lugar, se ha constatado que TempLM-TLG es una aproximación válida para resolver cualquier problema de planificación con restricciones temporales, ya que ha demostrado que es capaz de resolver problemas de una amplia variedad de dominios. Además, su rendimiento mejora en muchas ocasiones al de un planificador del estado del arte en planificación con restricciones como es OPTIC. Esto se observa fundamentalmente en los experimentos realizados con problemas no clasificados, donde la batería de problemas incluye distintos grados de restricciones, desde restricciones holgadas hasta restricciones ajustadas.

En segundo lugar, el análisis de los resultados obtenidos con problemas clasificados como *tight* indica que uno de los beneficios de TempLM es que, cuanto más restringido está un problema, más eficiente es la búsqueda. Así, en este tipo de problemas TempLM-TLG es la aproximación que obtiene los mejores resultados, tanto en *makespan* de los planes como en tiempo de cómputo.

En tercer lugar, con respecto a los problemas irresolubles, se ha demostrado que el proceso de *feedback* es una fuente de información muy valiosa para detectar este tipo de problemas. A pesar de su elevado coste computacional, la aplicación del proceso de *feedback* permite reducir en gran medida el espacio de búsqueda en problemas irresolubles y, por tanto, dar una respuesta dentro del tiempo disponible.

Por último, respecto al experimento realizado con el reciente enfoque de Karpas *et al.*, que también utiliza *landmarks temporales*, se ha demostrado que TempLM es más eficiente en la resolución de problemas altamente restringidos o para detectar problemas irresolubles.

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones generales de este trabajo de tesis y se apuntan algunas líneas de trabajo futuro. En el primer apartado se describen las características y la necesidad de la utilización de técnicas de planificación para afrontar eficientemente problemas temporales con restricciones. En el segundo apartado se describe la aproximación propuesta y se enumeran las aportaciones y los resultados más relevantes. En el tercer y último apartado se describen varias líneas de trabajos futuros derivadas de este trabajo de tesis.

7.1. Introducción

Las técnicas de planificación temporal permiten permite hacer frente a problemas más realistas que requieren una gestión explícita del tiempo. La definición de acciones con duración introduce una mayor complejidad en este tipo de problemas. El clásico criterio de minimización del número de acciones en planificación STRIPS no siempre es consistente con el criterio de optimizar el *makespan* del plan, generándose ahora un mayor número de alternativas a estudiar. Esto es consecuencia no solo de las diferentes combinaciones de acciones que se pueden utilizar para resolver un objetivo particular, sino también de las diferentes instanciaciones temporales para cada una de estas acciones.

En el mundo real existen diferentes tipo de aplicaciones donde el objetivo no

es encontrar el plan de menor duración sino encontrar un plan que satisfaga los objetivos del problema en un plazo determinado. Por ejemplo, en el transporte de productos perecederos es imprescindible que un producto no supere un tiempo máximo de transporte. En este caso, la definición de *deadlines* en un problema implica que se debe garantizar el cumplimiento de las mismas para que el plan sea válido.

Sin embargo, trabajar con problemas temporales con *deadlines* implica un cambio en la resolución de los mismos al tener que garantizar el cumplimiento de los *deadlines*, lo que hace que existan muy pocos planificadores temporales que manejen este tipo de restricciones. Además, la inclusión de *deadlines* en un problema puede implicar que éste sea irresoluble, y este es uno de los puntos débiles de los planificadores actuales.

7.2. Conclusiones

La principal motivación de este trabajo de tesis reside en las dificultades que tienen los planificadores temporales actuales en la detección de problemas irresolubles. Posteriormente, este trabajo se ha centrado en el desarrollo de un algoritmo de planificación temporal independiente del dominio incorporando el modelo de *landmarks temporales* para resolver problemas altamente restringidos.

7.2.1. Contribuciones de la tesis

Se ha presentado un modelo de *landmarks temporales* para la resolución de problemas altamente restringidos, es decir, detección de problemas irresolubles o resolución de problemas con restricciones muy ajustadas. Además, se ha desarrollado TempLM, un planificador temporal donde se ha integrado el modelo de *landmarks temporales* lo que permite tratar eficientemente con *deadlines*.

Este trabajo se estructura en tres partes: la primera es la construcción del grafo de *landmarks* STRIPS, la segunda parte es la construcción del modelo de *landmarks temporales* y, la última parte describe el desarrollo de TempLM. Al ser un diseño modular, se puede sustituir cualquiera de estos tres módulos por otro que constituya una mejora, siempre y cuando se respete la estructura de la información entre cada

módulo.

Las siguientes secciones presentan las conclusiones más interesantes de cada una de estas partes y un resumen de los resultados obtenidos.

7.2.1.1. Construcción del grafo de *landmarks* STRIPS

En esta parte se diseña la aproximación FULL como una combinación de las distintas aproximaciones de extracción de *landmarks* STRIPS, analizando las fortalezas y debilidades de cada una de ellas. La aproximación LM establece las bases para la extracción de *landmarks* y es la aproximación que consigue más *landmarks* individuales. La aproximación DL realiza un proceso para la extracción de *landmarks* ligeramente distinto al realizado por LM lo que le permite conseguir más *landmarks disyuntivos*. La aproximación DTG adapta las aproximaciones LM y DL al formalismo SAS^+ y añade un conjunto adicional de *landmarks* extraído a partir de los grafos de transición de dominio. Por último, la aproximación Pro realiza la extracción de *landmarks* (de acción y de literal) a través de la propagación de *etiquetas* en un PG. A partir de estas aproximaciones se integran las diferentes técnicas y se diseña la aproximación FULL. FULL-v3 utiliza LM como base para la extracción del conjunto inicial de *landmarks*, a continuación se realiza una propagación de literales y de acciones similar a la realizada por Pro pero sobre un RPG y finalmente se utiliza DL para conseguir el conjunto de *landmarks disyuntivos*. Se realizó una comparativa del número de *landmarks*, *landmarks disyuntivos* y órdenes que obtiene cada una de las aproximaciones donde se demostró que FULL-v3 es la aproximación que mejores resultados obtiene.

A partir de los *landmarks*, *landmarks disyuntivos* y órdenes que obtiene la aproximación FULL-v3 se crea un grafo de *landmarks* que se utiliza en la siguiente parte del trabajo.

7.2.1.2. Construcción del Modelo de *landmarks temporales*

El grafo de *landmarks* obtenido por la aproximación FULL se utiliza para la inicialización del TLG. Este TLG inicial se refina añadiendo nuevos *landmarks* que provienen de los *deadlines* establecidos en el problema. Los *landmarks* del grafo de *landmarks temporales* se caracterizan por disponer de los intervalos de generación,

validez y necesidad que se van actualizando (restringiendo) mediante un proceso de propagación de la información temporal a partir de las relaciones de orden, de exclusión mutua y de las propias restricciones temporales. En este proceso de actualización y propagación pueden aparecer inconsistencias entre los intervalos de validez y necesidad de los *landmarks temporales*. Estas inconsistencias pueden ser de dos tipos: resolubles, las cuales se pueden resolver y actualizan la información del grafo, e irresolubles, que son una indicación de que el problema no tiene solución. Finalmente, el TLG que se ha construido se utiliza en la siguiente parte del trabajo.

7.2.1.3. Desarrollo del planificador

En esta parte del trabajo se ha desarrollado TempLM, un planificador temporal independiente del dominio para problemas de planificación con *deadlines*. El planificador realiza una búsqueda en un espacio de planes parciales utilizando la heurística T-LMcut para guiar el proceso de búsqueda. TempLM incorpora el TLG obtenido durante la construcción del modelo de *landmarks temporales* para utilizarlo durante la búsqueda, de forma que un plan parcial del espacio de búsqueda que no sea compatible con la información del grafo es eliminado. Posteriormente, se crea una nueva versión de TempLM donde se realiza un proceso de *feedback*. En esta versión de TempLM con *feedback*, cada nodo del árbol de búsqueda dispone de su propio TLG que se refina a partir de la información del plan parcial contenido en dicho nodo.

7.2.1.4. Resultados de la tesis

Los resultados más importantes de la tesis se enumeran a continuación:

- **Respecto a la extracción de *landmarks* STRIPS**. La aproximación FULL-v3 es capaz de extraer un mayor número de *landmarks* individuales, *landmarks* disyuntivos y órdenes respecto a las aproximaciones LM, DL, DTG y Pro. En los experimentos realizados se muestra que FULL-v3 supera a la aproximación que mejores resultados obtuvo para cada dominio.
- **Respecto al modelo de *landmarks temporales***. Se ha demostrado la efectividad del modelo de *landmarks temporales* en la detección de problemas

irresolubles durante la construcción del $TRPG$ y durante la construcción del TLG debido a las inconsistencias que no se pueden resolver durante la actualización y propagación de los intervalos de generación, validez y necesidad de los *landmarks temporales* del TLG. Esto permite la detección rápida y eficiente de problemas irresolubles.

- **Respecto a la resolución de problemas altamente restringidos**. Los experimentos realizados demuestran que TempLM es una buena aproximación para la resolución de problemas temporales con *deadlines* y que cuanto más restringido está un problema, más eficiente es la búsqueda. Se realizó una comparativa inicial donde se comparaban las tres configuraciones de TempLM (TempLM-Búsq, TempLM-TLG y TempLM-FB) con los planificadores temporales del estado del arte que manejan *deadlines* MIPS-XXL, SGPLAN5 y OPTIC. Se observó que OPTIC superaba al resto de planificadores del estado del arte mientras que TempLM superaba a OPTIC en la mayoría de los casos. Por este motivo, el resto de experimentos se centraron en comparar OPTIC con TempLM-TLG y TempLM-FB.

  Más concretamente, TempLM-TLG obtiene los mejores resultados para una amplia variedad de problemas frente a OPTIC, siendo la aproximación que obtiene los mejores resultados, tanto en el *makespan* de los planes como en tiempo de cómputo en problemas clasificados como *tight*. Por otro lado, TempLM-FB es capaz de detectar el mayor número de problemas irresolubles gracias a la aplicación del proceso de *feedback* que permite reducir en gran medida el espacio de búsqueda, aunque su excesivo coste temporal le impide resolver de forma eficiente un gran número de problemas clasificados como *tight*. También se ha demostrado la efectividad del modelo de *landmarks temporales* en la detección de problemas irresolubles durante la construcción del $TRPG$ y durante la construcción del TLG debido a las inconsistencias que no se pueden resolver.

Concretamente, los puntos fuertes de este trabajo de tesis son los siguientes:

- Un modelo de extracción de *landmarks* STRIPS que consigue casi la totalidad de *landmarks* individuales que sirven para inicializar el TLG.
- El modelo de *landmarks temporales* donde se introduce por primera vez el

concepto de *landmark temporal* y donde se realiza una detección rápida y eficiente de problemas irresolubles mediante la actualización y propagación de los intervalos de generación, validez y necesidad de los *landmarks temporales*.

- El modelo de *landmarks temporales* se integra en TempLM lo que permite resolver problemas altamente restringidos o detectar aquellos problemas irresolubles que el modelo de *landmarks temporales* no ha sido capaz de detectar. En este sentido, TempLM-TLG es la aproximación que más problemas clasificados como *tight* resuelve, mientras que TempLM-FB es la aproximación que más problemas irresolubles detecta gracias a la aplicación del proceso de *feedback*.

7.2.2. Publicaciones de la tesis

Adicionalmente, los resultados de este trabajo de tesis han dado lugar a una serie de publicaciones, que aparecen referenciadas a lo largo de la memoria. De entre ellas se destacan las siguientes:

- E. Marzal, L. Sebastia, E. Onaindia. *Temporal Landmark graphs for solving overconstrained planning problems*. Knowledge-Based System (KBS), ELSEVIER. 2015. Enviado

- E. Marzal, L. Sebastia, E. Onaindia. *Temporal landmarks for overconstrained planning problems with deadlines*. 27th IEEE International Conference on Tools with Artificial Intellegence. IEEE Computer Society. (ICTAI 2015)

- E. Marzal, L. Sebastia, E. Onaindia. *On the Use of Temporal Landmarks for Planning with Deadlines*. 24th International Conference on Automated Planning and Scheduling. AAAI Press. (ICAPS 2014) LNCS, Springer Verlag.

- E. Marzal, L. Sebastia, E. Onaindia. *Full extraction of landmarks in propositional planning tasks*. 12th International Conference of the Italian Association for Artificial Intelligence. LNCS, Springer. (IAAI 2011).

- E. Marzal, L. Sebastia, E. Onaindia. *Detection of unsolvable temporal planning problems through the use of landmarks*. 18th European Conference on Artificial Intelligence. IOS Press. (ECAI 2008)

- E. Marzal, L. Sebastia, E. Onaindia. *Extracción de landmarks en dominios temporales*. 12ª Conferencia de la Asociación Española para la Inteligencia Artificial. (CAEPIA 2007)

- L. Sebastia, E. Marzal, E. Onaindia. *Extracting landmarks in temporal planning domains*. 7th International Conference on Artificial Intelligence. CSREA Press. (ICAI 2007)

7.3. Trabajos futuros

Este trabajo de tesis tiene las siguientes debilidades:

- Dificultad de la aproximación TempLM-FB para resolver los problemas clasificados con *tight*. Esto se debe, fundamentalmente, al tiempo de cómputo requerido en el proceso de *feedback* para hacer la copia del TLG de cada nodo.

- Imposibilidad de TempLM de resolver problemas con acciones concurrentes debido a cómo se aplica una acción durante la generación de los nodos sucesores.

- No se puede garantizar la obtención de la mejor solución debido al uso de la heurística T-LMcut que no es admisible.

Abordar estas dificultades daría lugar a los siguientes trabajos futuros:

La gran desventaja del proceso de *feedback* es el tiempo de cómputo requerido para hacer la copia del TLG debido a la gran cantidad de información que hay que transferir. Una posible mejora sería modificar la **estructura y compactar la información** necesaria para la actualización y propagación de los nodos feedback. Reducir este tiempo permitiría resolver más problemas y mejorar la eficiencia de la aproximación TempLM-FB.

Otra mejora sería poder **resolver problemas que requieran acciones concurrentes**. Adaptar la forma de aplicar las acciones al plan permitiendo también poder aplicarlas lo más tarde posible permitiría poder tratar con este tipo de acciones y poder resolver más problemas.

Incorporar al proceso de búsqueda de TempLM una **heurística admisible** permitiría garantizar la obtención de la mejor solución así como mejorar la eficiencia del propio proceso de búsqueda.

Otros trabajos futuros podrían ser los siguientes:

Como se ha comentado el diseño modular de este trabajo permite el uso de sus componentes con otras configuraciones, una de las posibilidades sería **utilizar el modelo de *landmarks temporales* en otros planificadores**. La información contenida en el modelo de *landmarks temporales* puede ser información muy útil para ser utilizada por otros planificadores, tanto para la definición de heurísticas como para la definición de estrategias que permitan reducir el espacio de búsqueda.

En la versión actual de nuestro modelo se pueden manejar todas las restricciones temporales de PDDL3.0 que representan un *deadline* mediante los operadores modales within, always-within, sometime-before y sometime-after. También se soporta las restricciones temporales definidas utilizando los TIL. Nuestro modelo puede adaptarse fácilmente a las características de cualquier modelo temporal, por ejemplo se podrían incluir las **restricciones sobre intervalos de Allen** [2].

Apéndice A

Representación de los dominios en PDDL

A.1. Especificación del dominio *depots*

En esta sección se incluye especificación en PDDL del problema de transporte del dominio *depots*.

```
(define (domain Depot)
(:requirements :typing)
(:types place locatable - object
depot distributor - place
        truck hoist surface - locatable
        pallet crate - surface)

(:predicates (at ?x - locatable ?y - place)
             (on ?x - crate ?y - surface)
             (in ?x - crate ?y - truck)
```

```
            (lifting ?x - hoist ?y - crate)
            (available ?x - hoist)
            (clear ?x - surface)
            (adjacent ?x - place ?y - place))

(:action Drive
:parameters (?x - truck ?y - place ?z - place)
:precondition (and (at ?x ?y) (adjacent ?y ?z))
:effect (and (not (at ?x ?y)) (at ?x ?z)))

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

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

(:action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (lifting ?x ?y))
:effect (and (not (lifting ?x ?y)) (in ?y ?z) (available ?x)))
```

```
(:action Unload
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (available ?x) (in ?y ?z))
:effect (and (not (in ?y ?z)) (not (available ?x)) (lifting ?x ?y)))
)
```

A.2. Especificación del problema del dominio *depots* del Capítulo 2

```
(define (problem depotprob0) (:domain Depot)

(:objects
          loc1 loc2 - place
          r1 - truck
          p1 p2 - pallet
          c1 c2 c3 - crate
          crane1 - hoist)
(:init
      (at p1 loc1)
      (at p2 loc1)
      (in c1 p1)
      (in c3 p1)
      (clear c3)
      (on c3 c1)
      (on c1 p1)
```

```
      (in c2 p2)
      (clear c2)
      (on c2 p2)
      (at crane1 loc1)
      (available crane1)
      (adjacent loc1 loc2)
      (adjacent loc2 loc1)
      (at r1 loc2)
)
(:goal (and (on c3 c2))
))
```

A.3. Especificación del problema del dominio *depots* del Capítulo 3

```
(define (problem depotprob1) (:domain Depot)
(:objects
D0 D1 D2 - Distributor
T0 T1 - Truck
P0 P1 P2 - Pallet
C0 C1 - Crate
H0 H1 H2 - Hoist)
(:init
(at P0 D0)
(clear C1)
(at P1 D1)
```

```
(clear C0)
(at P2 D1)
(clear P2)
(at T0 D1)
(at T1 D0)
(at H0 D0)
(available H0)
(at H1 D1)
(available H1)
(at H2 D2)
(available H2)
(at C0 D1)
(on C0 P1)
(at C1 D0)
(on C1 P0)
(adjacent D0 D1)
    (adjacent D1 D0)
    (adjacent D0 D2)
    (adjacent D2 D0)
    (adjacent D1 D2)
    (adjacent D2 D1)
)

(:goal (and
(on C0 P2)
(on C1 P1))
))
```

Bibliografía

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. 42

[2] J. F. Allen. A general model of action and time. *Artificial Intelligence*, 23(2), 1984. 184

[3] C. Bäckström and B. Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11(4):625–655, 1995. 68

[4] R. Bellman. *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, 1978. 2

[5] J. Benton, A. J. Coles, and A. Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, volume 77, pages 2–10, 2012. 4, 5, 25, 44, 47, 152, 159

[6] J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. An evolutionary metaheuristic based on state descomposition for domain-independent satisficing planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, (ICAPS)*, pages 18–25, 2010. 45

[7] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997. 3, 20, 25, 26, 39, 61, 69

[8] B. Bonet and J. Castillo. A complete algorithm for generating landmarks. In *Proceedings of the 21st International Conference Conference on Automated Planning and Scheduling (ICAPS)*, pages 315–318, 2011. 129

[9] B. Bonet and H. Geffner. GPT: a tool for planning with uncertainty and partial information. In *Proceedings of the IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, pages 82–87, 2001. 4

[10] B. Bonet and H. Geffner. Heuristic Search Planer 2.0. *AI Magazine*, 22(3):77–80, 2001. 30

[11] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence Journal*, 129:5–33, 2001. 21, 29

[12] B. Bonet and M. Helmert. Strengthening landmark heuristics via hitting sets. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 329–334, 2010. 129

[13] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI)*, pages 714–719, 1997. 30

[14] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994. 13, 28, 147

[15] Y. Chen, C.W. Hsu, and B.W. Wah. SGPlan: Subgoal partitioning and resolution in planning. *Proceedings of the 4th International Planning Competition Booklet (IPC)*, 2004. 4

[16] Y. Chen, B.W. Wah, and C.W. Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research*, 26:323–369, 2006. 4, 5, 44, 151

[17] A.J. Coles, A. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 42–49, 2010. 4, 25, 48

[18] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52:49–86, 1991. 3

[19] T. Dean, R.J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4(3):381–398, 1988. 43

[20] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1):61–95, 1991. 43

[21] M.B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence*, 132(2):151–182, 2001. 20

[22] M.B. Do and S. Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. *Recent Advances in AI Planning, 6th European Conference on Planning (ECP)*, pages 109–120, 2001. 42

[23] M.B. Do and S. Kambhampati. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, 2003. 35

[24] C. Domshlak, M. Katz, and S. Lefler. When abstractions met landmarks. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 50–56, 2010. 129

[25] B. Drabble and R. Kirby. Associating ai planner entities with an underlying time point network. In *Proceedings of the European Workshop on Planning*, pages 27–38. Springer, 1991. 43

[26] S. Edelkamp and J. Haoffmann. Pddl2.2: The language for the classical part of ipc-4. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 2–6, 2004. 35

[27] S. Edelkamp and M. Helmert. Mips: The model-checking integrated planning system. *AI Magazine*, 22(3):67–72, 2001. 20

[28] S. Edelkamp, S. Jabbar, and M. Nazih. Large-scale optimal pddl3 planning with mips-xxl. In *Proceedings of the 5th International Planning Competition Booklet (IPC)*, pages 28–30, 2006. 5, 151

[29] A. El-Kholy and B. Richards. Temporal and resource reasoning in planning: the parcPLAN approach. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI)*, pages 614–618, 1996. 25

[30] P. Eyerich. Preferring properly: Increasing coverage while maintaining quality in anytime temporal planning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI). Including Prestigious Applications of Artificial Intelligence (PAIS-2012)*, pages 312–317, 2012. 47

[31] P. Eyerich, R. Mattmüller, and G. Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009. 4, 44, 47

[32] 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. 2, 13, 20

[33] M. Fox and L. Derek. STAN4: a hybrid planning strategy based on subproblem abstraction. *AI Magazine*, 22(3):81–84, 2001. 34

[34] M. Fox and L. Derek. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002. 4, 34

[35] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998. 28

[36] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 956–961, 1999. 28

[37] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003. 36, 86

[38] A. Garrido, M. Arangú, and E. Onaindia. A constraint programming formulation for planning: from plan scheduling to plan generation. *J. Scheduling*, 12(3):227–256, 2009. 43

[39] A. Garrido, M. Fox, and D. Long. A temporal planning system for durative actions of pddl2.1. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, pages 586–590. Citeseer, 2002. 37, 39

[40] A. Garrido and E. Onaindia. On the application of least-commitment and heuristic search in temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 942–947, 2003. 4, 28

[41] A. Garrido and E. Onaindia. Domain-independent temporal planning in a planning-graph-based approach. *AI Communications*, pages 341–367, 2006. 40

[42] H. Geffner. Functional STRIPS: a more flexible language for planning and problem solving. *Logic-Based Artificial Intelligence, Jack Minker (Ed.), Kluwer*, 2000. 13

[43] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009. 5, 49, 50, 86, 88, 151

[44] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003. 4, 44, 46, 90, 91

[45] A. Gerevini, A. Saetti, I. Serina, and Toninelli. P. LPG-TD: a fully automated planner for PDDL2.2 domains. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, 2004. 46

[46] A. Gerevini and L. Schubert. Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, 5:95–137, 1996. 25

[47] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the Conference on AI Planning Systems (AIPS)*, pages 112–121, 2000. 4

[48] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998. 3, 14

[49] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proceedings of the International Conference on AI Planning and Scheduling (AIPS)*, pages 61–67, 1994. 25, 43

[50] M. Ghallab, D. Nau, and P. Traverso. *Automated Task Planning. Theory & Practice*. Elsevier, 2004. 12, 15, 39, 43

[51] C. Green. Applications of theorem proving to problem solving. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 741–747, 1969. 2

[52] P. Haslum. Improving heuristics through relaxed search. *Journal of Artificial Research*, 25:233–267, 2006. 42, 44

[53] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proceedings of the International Conference on AI Planning and Scheduling (AIPS)*, pages 140–149, 2000. 30

[54] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proceedings of the European Conference on Planning (ECP*, pages 121–132, 2001. 42

[55] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. 44, 47

[56] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th Conference on Automated Planning and Scheduling (ICAPS)*, 2009. 125, 128, 129

[57] J. Hoffman. Local search topology in planning benchmarks: A theoretical analysis. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pages 379–387, 2002. 45

[58] J. Hoffman and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 4, 21, 29

[59] J. Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium of Methodologies for Intelligent Systems (ISMIS)*, pages 216–227, 2000. 30

[60] J. Hoffmann and S. Edelkamp. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005. 5, 49, 86

[61] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–287, 2004. 53, 54, 55, 56, 57, 58, 59, 60, 102, 107

[62] IPC. International planning competition. *http://www.plg.inf.uc3m.es/ipc2011-deterministic/*, 2011. 47

[63] IPC. International planning competition. *http://www.icaps-conference.org/index.php/Main/Competitions*, 2015. 34, 44, 70

[64] D. Joslin and M.E. Pollack. Least-cost flaw repair: A plan refinement strategy for partial-order planning. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, pages 1004–1009, 1994. 25

[65] S. Kambhampati. Notes from the ASU planning seminar: Planning methods in artificial intelligence, 2000. 12

[66] E. Karpas, D. Wang, B. C. Williams, and P. Haslum. Temporal landmarks: What must happen, and when. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling, ICAPS*, pages 138–146, 2015. 153, 172, 173

[67] H.A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992. 20

[68] H.A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1194–1201, 1996. 3

[69] M. R. Khouadjia, M. Schoenauer, V. Vidal, J. Dréo, and P. Savéant. Multi-objective ai planning: Evaluating dae yahsp on a tunable benchmark. In *Proceedings of the Evolutionary Multi-Criterion Optimization*, pages 36–50. Springer, 2013. 4, 44

[70] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proceedings of the 4th European Conference in Planning (ECP)*, pages 273–285, 1997. 28

[71] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999. 4, 27

[72] D. Long and M. Fox. Progress in AI planning research and applications. *UPGRADE. The European Online Magazine for the IT Professional*, 3(5):10–24, 2002. 3, 28

[73] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003. 101

[74] E. Marzal, L. Sebastia, and E. Onaindia. Detection of unsolvable temporal planning problems through the use of landmarks. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, pages 919–920, 2008. 125, 127, 152

[75] E. Marzal, L. Sebastia, and E. Onaindia. Full Extraction of Landmarks in Propositional Planning Tasks. In *Proceedings of the 12th Conference of the Italian Association for Artificial Intelligence (AI*IA)*, volume 6934, pages 383–388, 2011. 53, 54

[76] E. Marzal, L. Sebastia, and E. Onaindia. On the use of temporal landmarks for planning with deadlines. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 172–180. AAAI Press, 2014. 125

[77] E. Marzal, L. Sebastia, and E. Onaindia. Temporal landmarks for overconstrained planning problems with deadlines. In *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 437–444. IEEE Computer Society, 2015. 126, 140

[78] D.A. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)*, pages 634–639, 1991. 3

[79] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969. 12, 13

[80] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, volume 3, pages 142–149, 1996. 30

[81] D. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000. 14

[82] N. Muscettola. Hsts: Integrating planning and scheduling. Technical report, DTIC Document, 1993. 42

[83] A. Newell and H. Simon. GPS, a program that simulates human thought. *Computers and Thought*, pages 279–293, 1963. 2

[84] X. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 459–466, 2001. 21

[85] E. Onaindia, L. Sebastia, and E. Marzal. 4SP: a four-stage incremental planning approach. In *Proceedings of the ECAI-00 Workshop Planning and Configuration (PuK)*, pages 115–122, 2000. 31

[86] E. Onaindia, L. Sebastia, and E. Marzal. Incremental local search for planning problems. In *Lecture Notes in Artificial Intelligence. ECAI-2000 Workshop on Local Search for Planning & Scheduling*. Springer - Verlag, 2001. 31

[87] E.P.D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 324–332, 1989. 14

[88] J.S. Penberthy and D.S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3rd International Conference on the*

Principles of Knowledge Representation and Reasoning (KR), pages 103–114, 1992. 3, 20, 24

[89] J.S. Penberthy and D.S. Weld. Temporal planning with continuous change. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1010–1015, 1994. 25, 35, 43

[90] J. Penix, C. Pecheur, and K. Havelund. Using model checking to validate ai planner domain models. In *Proceedings of the 23rd Annual Software Engineering Workshop*, 1998. 43

[91] M. Peot and D. Smith. Threat-removal strategies for partial-order planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 492–499, 1993. 22

[92] J. Porteous and S. Cresswell. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*, pages 45–54, 2002. 59, 65

[93] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP)*, pages 37–48. Springer Verlag, 2001. 31, 54, 56, 59

[94] E. Rich and K. Knight. *Introduction to Artificial Intelligence*. McGraw-Hill, 1991. 2

[95] S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 945–982. AAAI Press, 2008. 60, 68

[96] S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. 70

[97] J. Rintanen. Complexity of concurrent temporal planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 280–287, 2007. 147

[98] S. Russell and P. Norving. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995. 2

[99] E. Rutten and J. Hertzberg. Temporal planner= nonlinear planner+ time map management. *AI Communications*, 6(1):18–26, 1993. 43

[100] E. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 206–214, 1975. 3

[101] O. Sapena, E. Onaindia, and A. Torreño. FLAP: applying least-commitment in forward-chaining planning. *AI Communications*, 28(1):5–20, 2015. 4, 25

[102] L. Sebastia and E. Marzal. Splin: Una herramienta para el diseño, ejecución y evaluación de problemas de planificación. *Revista Iberoamericana de Inteligencia Artificial*, 7:54–64, 1999. 25

[103] L. Sebastia, E. Onaindia, and E. Marzal. Beneficios de una gesti'on eficiente de variables en planificaci'on. In *Actas de la VIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA)*, 1999. 25, 31

[104] L. Sebastia, E. Onaindia, and E. Marzal. Improving expressivity and efficiency in partial-order causal link planners. In *Proceedings of the 18th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*, pages 124–136, 1999. 25

[105] L. Sebastia, E. Onaindia, and E. Marzal. A graph-based approach for POCL planning. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, pages 531–535. IOS Press, 2000. 31

[106] B. Selman and H. A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 93, pages 46–51, 1993. 46

[107] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000. 25, 42

[108] D.E. Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 326–337, 1999. 4, 28, 33, 35, 39, 40

[109] V. Vidal. Yahsp2: Keep it simple, stupid. *In IPC 2011 planner abstracts*, pages 83–90, 2011. 4, 44, 45

[110] V. Vidal, L. Bordeaux, and Y. Hamadi. Adaptive k-parallel best-first search: A simple but efficient algorithm for multi-core domain-independent planning. In *Proceedings of the Third Annual Symposium on Combinatorial Search*, 2010. 46

[111] V. Vidal et al. A lookahead strategy for heuristic search planning. In *ICAPS*, pages 150–160, 2004. 44, 45

[112] D.S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4), 1994. 21

[113] D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999. 11

[114] D.S. Weld, C.R. Anderson, and D.E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 897–904, 1998. 28

[115] Q. Yang. *Intelligent Planning. A Decomposition and Abstraction Based Approach*. Spinger-Verlag, 1997. 23

[116] H.L.S. Younes and R.G. Simmons. On the role of ground actions in refinement planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS)*, pages 54–61, 2002. 25

[117] H.L.S. Younes and R.G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003. 20, 35, 43

[118] L. Zhu and R. Givan. Landmark Extraction via Planning Graph Propagation. In *Notes of ICAPS Doctoral Consortium*, 2003. 60, 69, 73, 76, 77