UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

TFLAP: resolución de problemas de planificación con coordinación temporal

Autor: D Miguel Puigcerver Calbo

Dirigido por:
Dª Eva Onaindia de la Rivaherrera
D Óscar Sapena Vercher

Trabajo fin de máster: Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Dpto. de Sistemas Informáticos y Computación
Universitat politècnica de València
Valencia, España

12 de julio de 2015

TFLAP: resolución de problemas de planificación con coordinación temporal

Autor: D Miguel Puigcerver Calbo

Dirigido por:
D[a] Eva Onaindia de la Rivaherrera
D Óscar Sapena Vercher

Resumen

A partir del planificador FLAP desarrollado en el subgrupo de investigación GRPS del GTI-IA de la UPV, en este trabajo se aborda la ampliación de dicho planificador para gestionar problemas con coordinación temporal, característica imprescindible para obtener planes realistas. En el TFM se describe el estado del arte en planificación, se especifican las extensiones realizadas que han dado lugar al planificador temporal TFLAP, el cual se compara con diferentes planificadores del estado del arte, se analiza el rendimiento de TFLAP en la resolución de problemas temporales de las competiciones de planificación así como su aplicabilidad para la resolución de problemas reales y se plantean modificaciones adicionales del planificador para mejorar sus resultados y aplicabilidad. Se analiza con especial detalle los problemas de concurrencia temporal, un punto débil de la mayoría de planificadores y que resulta vital para aplicaciones prácticas.

Índice general

Capítulo 1

Introducción

1.1. Motivación y objetivos

La motivación de este trabajo es añadir la capacidad de planificación temporal al planificador FLAP [22] desarrollado por el grupo GRPS de la UPV (http://users.dsic.upv.es/grupos/grps/). El resultado es un planificador denominado TFLAP cuyo funcionamiento es comparable con otros planificadores del estado del arte. Asimismo, la inclusión de características temporales permite resolver problemas del mundo real como los que se muestran en este documento. De esta forma se completa una parte pendiente de FLAP permitiendo que se pueda comparar con otros planificadores del estado del arte también en problemas de mayor realismo por la inclusión de consideraciones temporales.

Además de poder usar TFLAP con problemas temporales, el objetivo es obtener unos resultados de mayor calidad demostrando la viabilidad de la aproximación de orden parcial de FLAP; es decir, el objetivo es mejorar los resultados de otros planificadores del estado del arte. Por tanto, la motivación de este trabajo es diseñar la extensión temporal del planificador FLAP y analizar su funcionamiento respecto a tres parámetros:

- La cobertura de problemas resueltos: resolver la mayor cantidad posible de problemas.
- La calidad de los planes, esto es, que el planificador proporcione planes con la menor duración posible.
- El rendimiento del planificador: conseguir el menor tiempo de ejecución posible.

En los problemas temporales encontramos una problemática adicional: la concurrencia entre acciones. Es conocido que muchos planificadores no consiguen resolver satisfactoriamente problemas con concurrencia. De esta forma, un objetivo adicional es diseñar TFLAP de modo que sea capaz de resolver problemas concurrentes que hoy por hoy no pueden resolver algunos de los planificadores del estado del arte.

1.2. Estructura del documento

Este documento está estructurado en diferentes capítulos. Tras esta introducción, en el capítulo 2 se describe el estado del arte en planificación, incluyendo la descripción de los problemas y diferentes aproximaciones. El capítulo 3 se describe el planificador FLAP y el trabajo realizado para adaptarlo a problemas temporales generando el planificador TFLAP.

A continuación hay dos capítulos de resultados en los que se compara TFLAP con distintos planificadores del estado del arte. En el capítulo 4 se realiza un análisis de los problemas temporales concurrentes y de la capacidad de diferentes planificadores para resolver dichos problemas. Adicionalmente se analiza la aplicabilidad de TFLAP a la resolución de problemas reales. El capítulo 5 contiene la descripción de la batería de pruebas realizada junto a los resultados obtenidos y su correspondiente análisis.

Por último, en el capítulo 6 se presentan las conclusiones a las que se ha llegado y se proponen líneas de trabajo futuro.

Capítulo 2

Planificación en inteligencia artificial

Planificación en inteligencia artificial es el proceso de establecer una secuencia de acciones que permiten transformar una situación inicial en una situación final deseada. Desde el punto de vista computacional se trata de un problema de búsqueda que consiste en buscar las acciones que resuelven el problema y establecer la ordenación de las mismas. Un plan o conjunto de acciones que resuelve un problema de planificación puede ser una secuencia estrictamente ordenada y serializada de acciones o bien tratarse de un plan que contiene acciones que se pueden ejecutar en paralelo (o simultáneamente).

En este capítulo se explican los conceptos básicos de problema de planificación, y se presenta un resumen del estado del arte en planificación temporal, que es el objetivo del presente trabajo. Concretamente, el capítulo contiene:

- Los elementos principales para la definición de un problema de planificación.
- Las principales aproximaciones a la planificación clásica.
- Un resumen del estado del arte en planificación temporal.
- Finalmente se presenta el lenguaje usado para especificar problemas de planificación, PDDL (*Planning Domain Definition Language*), tanto su versión clásica como la última versión basada en variables estado. También se incluye la extensión del lenguaje PDDL para describir las características temporales de un problema de planificación.

2.1. Definición de un problema en planificación clásica

Un problema de planificación clásica es un problema de búsqueda cuyo objetivo es encontrar un conjunto de acciones o plan que conducen a un sistema,

desde un estado inicial hasta un estado objetivo. La descripción de un problema de planificación se compone de un conjunto de operadores, un estado inicial y una descripción del objetivo a conseguir.

De esta forma, el objetivo de la planificación en inteligencia artificial es el control de sistemas dinámicos. Un sistema dinámico es un entorno cuyo estado evoluciona en el tiempo. Esa evolución se puede modelar como una función de transición que toma como entrada el estado anterior del sistema y lo transforma en un nuevo estado. Un sistema dinámico es controlable si es posible influir en las transiciones mediante la aplicación de acciones. Es decir, si es posible controlar el comportamiento del sistema dinámico para llegar al estado objetivo. El estado del sistema en un instante de tiempo t, dependerá de su estado en $t-1$ y de la acción que se aplique en $t-1$ [16].

Durante el proceso de búsqueda, a medida que se añaden acciones, se crean dependencias y restricciones entre las acciones. Además, se pueden encontrar diferentes conjuntos de acciones que transformen el estado inicial en el estado final deseado.

Para reducir la complejidad del problema, la planificación clásica realiza una serie de asunciones [11]:

- El problema se modela mediante un número finito de estados alcanzables.
- El sistema es totalmente observable, es decir, el estado del mundo y todos sus componentes son totalmente conocidos en todo momento.
- El sistema es determinista, es decir, la aplicación de una acción a_1 en un estado e_1 siempre lleva al mismo estado e_2 y ambos estados son conocidos.
- El sistema es estático, lo que significa que la transformación de estados se produce única y exclusivamente por la aplicación de acciones.
- Los objetivos son conocidos desde el principio y no cambian.
- Un plan solución es una secuencia finita de acciones.

La planificación clásica incluye una serie adicional de suposiciones:

- El plan solución es una secuencia linealmente ordenada de acciones.
- Las acciones son instantáneas, es decir, son acciones puntuales en el tiempo y no tienen una duración asociada.
- Se asume una planificación offline, es decir, se construye un plan completo antes de ejecutar cualquier parte del mismo.

Debido a la variedad de combinaciones de acciones que permiten alcanzar el estado objetivo desde el estado inicial del problema, el objetivo de la mayoría de planificadores no es solo encontrar un plan sino optimizar dicho plan de acuerdo a algún parámetro. En planificación clásica, el criterio utilizado por los planificadores secuenciales es optimizar el número de acciones, y el criterio utilizado por planificadores paralelos, los cuales son capaces de manejar un orden parcial entre las acciones, es minimizar el número de acciones o bien el número de pasos de ejecución del plan o puntos de tiempo del plan.

A continuación se describen los elementos que intervienen en la definición de un problema de planificación tomando como referencia [11].

Un problema de planificación se puede modelar mediante un **lenguaje lógico de primer orden** $\mathcal{L}$, el cual permite definir completamente el problema de planificación. Los átomos del lenguaje $\mathcal{L}$ se modelan con símbolos de **predicado** y sus correspondientes argumentos, los cuales se usan para definir las propiedades de los objetos. Un ejemplo de predicado podría ser el siguiente; un símbolo de predicado `localizado` y dos argumentos para indicar el objeto y su localización, con lo que resultaría el átomo `(localizado ?objeto ?lugar)`. Este es el mecanismo que tradicionalmente se ha usado de forma exclusiva para modelar los átomos de $\mathcal{L}$. Al sustituir los argumentos de un átomo por sus posibles valores se obtiene un átomo instanciado que se denomina **literal** o **hecho**. Un ejemplo de literal sería `(localizado o1 l1)` que indica que el objeto `o1` está situado en la localización `l1`. El conjunto de todos los literales relevantes de un problema se denomina $\mathcal{H}$.

Muchos planificadores actuales combinan la utilización de un lenguaje de primer orden con la representación de la información del problema mediante **variables estado** tal como hace *FMAP* [25]. Una variable estado describe una propiedad de un objeto del problema mediante una palabra clave (la propiedad del objeto) y el objeto al que hace referencia: `propiedad-objeto`. Una variable estado equivale a la parte del literal que identifica un atributo de un objeto. Una variable estado v tiene asociado un conjunto de valores D_v mutuamente excluyentes que se denomina **dominio de la variable**. En cualquier instante, v tomará uno y solo uno de los valores de D_v, de forma que al asignar un nuevo valor a la variable, se elimina su valor anterior. Por ejemplo, `localizado-o1` es una variable que representa la posición del objeto `o1`. Si las posibles localizaciones del objeto `o1` son `l1,l2,l3` entonces $D_{localizado-o1}$ ={`l1,l2,l3`}. Un **fluent** representa la asignación de un valor $d \in D_v$ a una variable estado v, y se describe mediante la tupla $\langle v, d \rangle$. Por ejemplo, `<localizado-o1, l1>` indica que el objeto `o1` está situado en la localización `l1`. El conjunto de todos los fluents relevantes de un problema se denomina $\mathcal{F}$.

Los primeros planificadores representaban la información de un problema de planificación utilizando única y exclusivamente $\mathcal{L}$. En cambio, los planificadores actuales combinan la especificación del problema mediante predicados y variables estado. Un literal se puede equiparar a un fluent cuyo dominio es de tipo booleano, de forma que ambas representaciones son equivalentes. De este modo, es posible describir un problema (o que el planificador trabaje internamente) con fluents, literales o una combinación de ambos. Debido a esta equivalencia, los planificadores modernos usan preferentemente una representación basada en variables estado y fluents, pues aporta mayor flexibilidad, restringiendo el uso de predicados para variables estado cuyo dominio es booleano.

A continuación se describen los principales elementos que intervienen en la definición de un problema de planificación.

- Un **estado** es un conjunto finito de fluents (o literales) del sistema: $S \subseteq \{\mathcal{F}, \mathcal{H}\}$. Por simplicidad, en el resto del documento hablaremos únicamente de fluents $\mathcal{F}$ para denotar la información de un estado de un problema de planificación, pudiendo éste estar formado por fluents o literales.

- El **enunciado** de un problema de planificación es una tripleta $P = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ cuyos componentes son:
  - Un **estado inicial** $\mathcal{I} \subseteq \mathcal{F}$ especifica los fluents de la situación inicial del problema, es decir, los valores iniciales de las variables estado que modelan el problema.
  - El **estado objetivo** $\mathcal{G} \subseteq \mathcal{F}$ es la conjunción de fluents que representa las condiciones que se quiere alcanzar. Un estado S_n es un estado objetivo si contiene todos los fluents de $\mathcal{G}$: $\mathcal{G} \subseteq S_n$. Es posible alcanzar diferentes estados que sean estado objetivo, en cuyo caso se diferenciarán por el conjunto de fluents que no pertenecen a $\mathcal{G}$.
  - Un **operador de planificación** o define un tipo de operación que se puede realizar en el dominio del problema. $\mathcal{O}$ es el conjunto de los operadores relevantes del problema. Un operador $o \in \mathcal{O}$ se identifica como la tupla $o = \langle pre(o), eff(o) \rangle$ compuesta de:
    - $pre(o)$ es el conjunto de **precondiciones** que deben cumplir las variables estado en un estado para poder aplicar el operador o.
    - $eff(o)$ representa los **efectos** del operador; es un conjunto finito de operaciones de asignación sobre las variables estado que representan los cambios que se producen en el estado tras la aplicación del operador o.

Una **acción** a se obtiene sustituyendo los parámetros (variables estado) de un operador del problema por unos valores concretos. Una acción a, por lo tanto, es una instancia concreta de un operador. En general, todos los planificadores trabajan internamente con acciones u operadores totalmente instanciados, asignando un valor concreto a cada variable estado del operador. De esta forma una acción a se define, al igual que un operador, por un conjunto de precondiciones $pre(a)$ y un conjunto de efectos $eff(a)$. Una acción a es aplicable en un estado S si $pre(a) \subseteq S$; el resultado de aplicar a en S genera un nuevo estado S' en el que se aplican los cambios en las variables estado representados en $eff(a)$. Un efecto de la forma $(v = d')$ de una acción a que se aplica en un estado S asigna a la variable estado v el valor d' generando el fluent $\langle v, d' \rangle$ en el estado S'. Adicionalmente, si existe un fluent $\langle v, d \rangle, d \neq d'$ en S, éste no se añade a S' para reflejar el cambio de valor de la variable en el nuevo estado. Por otro lado, todos los fluents $< v, d >$ de S cuyas variables estado no se ven afectadas por $eff(a)$ se mantienen en el nuevo estado S'.

Un **problema de planificación** $\mathcal{P}$ se define por tanto como una tripleta $(\mathcal{A}, \mathcal{I}, \mathcal{G})$ sobre un enunciado $P = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ en donde $\mathcal{A}$ es el conjunto de acciones obtenidas a partir de la instanciación de los operadores del conjunto $\mathcal{O}$. Finalmente, un **plan solución** para un problema $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ es un conjunto de acciones $\pi = \{a_1, a_2, \cdots, a_n\} \in \mathcal{A}^*$ que permite alcanzar el estado objetivo $\mathcal{G}$ partiendo del estado inicial $\mathcal{I}$.

2.2. Aproximaciones de planificación clásica

Esta sección describe los paradigmas clásicos más utilizados en planificación, los cuales se diferencian, entre otras cosas, por el tipo de representación interna del problema: representación basada en estados, representación basada en planes o utilización de grafos de planificación.

La primera aproximación utilizada para resolver un problema de planificación fue plantear un problema como un proceso de búsqueda en un espacio de estados donde cada nodo del árbol de búsqueda representa un estado del problema de planificación.

Dado un nodo S del árbol de búsqueda, un nodo sucesor S' representa un estado que se puede alcanzar tras la aplicación de una acción en S. Por tanto, las aristas en un espacio de estados representan acciones aplicables en un estado. La aproximación habitual de búsqueda en un espacio de estados es la **búsqueda hacia adelante**. En la búsqueda hacia adelante el nodo raíz del árbol S_0 se corresponde con el estado inicial: $S_0 = \mathcal{I}$. El proceso de búsqueda consiste en elegir un nodo a expandir generando sus sucesores mediante la aplicación de acciones de $\mathcal{A}$ hasta que se alcanza un nodo S_g tal que $\mathcal{G} \subseteq S_g$. El camino que se define desde el nodo raíz S_0 hasta el nodo S_g que contiene los fluents de $\mathcal{G}$ es el plan solución del problema. Uno de los problemas de esta aproximación 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 se necesita reducir el espacio de búsqueda o bien utilizar funciones heurísticas informadas que permitan guiar la búsqueda hacia el estado objetivo. Una opción para reducir el factor de ramificación es realizar un proceso de **búsqueda hacia atrás** o regresiva, como hace el planificador STRIPS [6]. La búsqueda hacia atrás, sigue un proceso inverso, partiendo de un estado S_g y expandiendo los nodos mediante la aplicación de las acciones de $\mathcal{A}$ en sentido inverso hasta llegar a un nodo que contenga el estado inicial S_i tal que $\mathcal{I} \subseteq S_i$. La búsqueda regresiva permite reducir el factor de ramificación pero también presenta las dificultades propias de un proceso de búsqueda basado en estados que obtiene un plan secuencial donde las acciones se infieren en el sentido inverso en el que éstas se ejecutan.

Debido a los problemas de las aproximaciones basadas en estados, a partir de los años 90 surgen otro tipo de paradigmas de resolución de problemas de planificación que se resumen brevemente en esta sección.

2.2.1. Planificación de orden parcial

La Planificación de Orden Parcial (POP) surge como un intento de abordar las dificultades computacionales de la búsqueda basada en estados. A diferencia de la búsqueda basada en estados, en POP no se establece un orden entre dos acciones del plan hasta que esta decisión sea absolutamente necesaria. Es lo que se conoce como principio de *menor compromiso* [28].

En el caso de un planificador de orden parcial se utiliza un árbol de búsqueda donde cada nodo del árbol es un plan parcial, es decir, un plan donde no tiene

que existir necesariamente un orden total entre todo par de acciones. Un proceso de búsqueda POP termina cuando se encuentra un nodo en el árbol que contiene un plan solución.

El nodo raíz del árbol de búsqueda es un plan que contiene dos acciones especiales, una acción inicial que solo tiene efectos correspondiente al estado inicial $a_0 =< \emptyset, \mathcal{I} >$ y una acción final que solo tiene precondiciones correspondiente al estado objetivo $a_n =< \mathcal{G}, \emptyset >$. En cada paso del proceso de búsqueda se selecciona el plan más prometedor a expandir de acuerdo a la utilización de una heurística en particular. Una vez escogido un nodo, existen tres tipos de operaciones que se pueden aplicar sobre un nodo:

- Resolver una precondición de una acción del plan insertando una nueva acción o utilizando una acción ya existente en el plan.
- Resolver un conflicto entre dos acciones.
- Asignar un valor concreto a un parámetro de una acción del plan.

La resolución de una precondición de una acción a_j se realiza mediante la introducción de un **enlace causal** con otra acción a_i que genera el efecto necesario para a_j. Un enlace causal $a_i \xrightarrow{f} a_j$ indica que un fluent ($f = \langle v, d \rangle$) efecto de la acción a_i ($f \in eff(a_i)$) soporta la precondición de la acción a_j ($f \in pre(a_j)$). Esto significa que la acción a_i que genera el efecto debe ir antes que la acción a_j, lo que implica que la acción a_i está ordenada antes que a_j.

Un enlace causal $a_i \xrightarrow{f} a_j$ no solo establece el orden entre dos acciones sino que implica que el fluent $f = \langle v, d \rangle$ debe permanecer inalterado desde que la acción productora a_i lo genera hasta que la acción consumidora a_j lo necesita. De esta forma, una nueva acción a_k, **amenaza** el enlace causal $a_i \xrightarrow{f} a_j$ si tiene efectos que deshacen los efectos deseados de la acción productora a_i; esto es, existe un fluent $f' = \langle v, d' \rangle$ tal que $f' \in eff(a_k), d \neq d'$.

La resolución de una amenaza se puede resolver estableciendo una relación de orden, que representaremos con el símbolo $\prec$, entre las acciones involucradas en la amenaza. Una amenaza se puede resolver ejecutando la acción amenazante a_k antes de la acción productora a_i o después de la acción consumidora a_j. Estas ordenaciones se conocen como **democión** ($a_k \prec a_i$) y **promoción** ($a_j \prec a_k$) [20].

Al establecer una relación de orden entre dos acciones, también hay que considerar las relaciones de orden ya existentes entre acciones del plan. Si la nueva ordenación es incompatible con el resto de restricciones de orden del plan, se descarta la resolución de la amenaza por esa vía. Por ejemplo, si $a_a \prec a_b \prec a_c$, la opción de resolver una amenaza ordenando a_c antes que a_a, no es válida porque ya existe una relación de orden $a_a \prec a_c$, esa opción de descarta. La resolución de una amenaza por promoción se puede hacer siempre que la acción final del enlace causal, no corresponda con el estado objetivo. Mientras que la resolución por democión se podrá hacer cuando la acción inicial del enlace causal no sea el estado inicial.

Una forma alternativa de resolver conflictos es mediante la aplicación de **separación de variables**, restringiendo los valores de los parámetros de una

acción que no esté totalmente instanciada para evitar que se forme la amenaza. La separación de variables se puede aplicar si las acciones están parcialmente instanciadas, es decir, alguno de sus parámetros puede tener un valor sin asignar. La resolución de la amenaza consiste entonces en acotar los valores que puede tomar una variable de estado x de una acción:

- Igualdad: $x = y$, siendo y una constante u otra variable.
- Desigualdad: $x \neq y$, siendo y una constante u otra variable.

Si el planificador trabaja con acciones parcialmente instanciadas, una de las operaciones que puede aplicar en la expansión de un nodo es la resolución de una variable x de una acción, es decir, generar un nodo sucesor por cada uno de los valores en D_x.

Descripción formal de un POP

Un plan de orden parcial, formalmente se describe como $\pi = \langle A, O, B, C, F \rangle$ [29] donde:

- A es el conjunto de acciones del plan, las cuales pueden ser operadores total o parcialmente instanciados.
- O es el conjunto de relaciones de orden de la forma $a_i \prec a_j$ que existen entre las acciones de A.
- B es el conjunto de restricciones de ligadura, este elemento solo existirá si el planificador trabaja internamente con acciones parcialmente instanciadas.
- C es el conjunto de enlaces causales de la forma $a_i \xrightarrow{f} a_j$ que existen entre las acciones de A.
- F es el conjunto de tareas pendientes de resolver en el plan π; F incluye precondiciones de acciones que aún no están resueltas, es decir, para las que no se ha establecido un enlace causal, amenazas y parámetros de acciones que no tienen un valor definido (este último elemento solo existirá si el planificador trabaja internamente con acciones parcialmente instanciadas).

Un plan $\pi = \langle A, O, B, C, F \rangle$ es solución del problema $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ si se cumplen las siguientes condiciones:

- No quedan tareas pendientes de resolver, es decir, $F = \emptyset$.
- O y B son conjuntos consistentes.

Proceso de planificación POP

El proceso de planificación POP realiza una búsqueda en un árbol cuyos nodos son planes parciales. El punto de partida es un plan parcial $\pi_{raíz}$ donde

- $\pi_{raíz} =< A_{raíz}, O_{raíz}, B_{raíz}, C_{raíz}, F_{raíz} >$
- $A_{raíz} = \{a_0, a_g\}$ contiene dos acciones, la inicial $a_0 = \mathcal{I}$ correspondiente al estado inicial y la final $a_g = \mathcal{G}$ correspondiente al estado objetivo.
- El conjunto de relaciones de orden $O_{raíz}$ es simplemente $a_0 \prec a_g$.
- Los conjuntos $B_{raíz}$ y $C_{raíz}$ están vacíos.
- El conjunto $F_{raíz}$ contiene los objetivos del problema: $F_{raíz} = \mathcal{G}$

Por tanto, el nodo raíz del árbol es: $\pi_{raíz} =< \{a_0, a_g\}, \{a_0 \prec a_g\}, \emptyset, \emptyset, \mathcal{G} >$.

Cuando se escoge un nodo π_i del árbol para ser expandido, se selecciona uno de los fallos de su conjunto F_i y se genera un nodo sucesor por cada una de las formas de resolver el fallo. El conjunto F_i, como se ha comentado, contendría todas las precondiciones pendientes de resolver de π_i, las amenazas detectadas en π_i, y las variables de las acciones que no tienen un valor concreto asignado. Dado que la mayoría de planificadores trabajan con acciones totalmente instanciadas, en lo sucesivo asumiremos que todas las variables de las acciones que se insertan en un plan tienen un valor y que $B = \emptyset$ en todos los nodos del árbol. Cada posible forma de resolver un fallo de F_i del nodo π_i, genera un nodo hijo de π_i del siguiente modo:

- La precondición f de la acción acción a_j del plan π_i ($f \in pre(a_j)$) se puede resolver de dos formas bien añadiendo un enlace causal con una acción que existe en el plan o bien insertando una nueva acción al plan. Si existe una acción a_i en el plan π_i que tiene como uno de sus efectos la precondición $f \in eff(a_i)$ la resolución de la precondición se puede realizar añadiendo el enlace causal $a_i \xrightarrow{f} a_j$.

$$\left.\begin{array}{l} f \in pre(a_j) \mid a_j \in \pi_i \\ \exists a_i \in \pi_i \mid f \in eff(a_i) \end{array}\right\} \rightarrow a_i \xrightarrow{f} a_j \tag{2.1}$$

  Una forma alternativa de resolución consiste en añadir una nueva acción a_k tal que $f \in eff(a_k)$ y establecer el enlace causal $a_k \xrightarrow{f} a_j$ de forma análoga a 2.1.

- Las amenazas de un plan π_i pueden surgir en dos situaciones:
  - Cuando se añade un enlace causal en π_i, es necesario revisar si alguna de las acciones del plan amenaza al nuevo enlace causal.
  - Cuando se añade una nueva acción en π_i, es necesario revisar si la nueva acción amenaza un enlace causal existente en el plan.

  El proceso de planificación detecta las amenazas que se producen en π_i y selecciona una de ellas para su resolución mediante una nueva relación de orden y generando un nodo sucesor por cada posible forma de resolución: promoción o democión.

Heurística en POP

En el proceso de planificación POP existen dos puntos clave de selección:

- Hay que seleccionar primero el *nodo del árbol a expandir*, es decir, el plan π_i.
- Una vez seleccionado el plan π_i hay que seleccionar un *fallo a resolver* de F_i.

La selección que se use determinará en gran medida el tiempo de cómputo y la calidad del plan resultante del planificador. Por este motivo, resulta crítico el disponer de una buena heurística que ayude a realizar las elecciones correctas para dirigir la búsqueda hacia la solución.

Una estrategia habitual para seleccionar el nodo a expandir es usar un algoritmo A^* con una función heurística $f(\pi_i) = g(\pi_i) + h(\pi_i)$. Donde $g(\pi_i)$ representa el coste del plan π_i y $h(\pi_i)$ es la estimación del coste para llegar a tener un plan solución, cuyo valor real se denomina $h^*(\pi_i)$. Para la estimación de los costes es habitual utilizar:

- El número de pasos (número de acciones descontando las que se puedan ejecutar en paralelo) que contiene π_i como indicativo de su complejidad, es decir, se puede usar como $g(\pi_i)$.
- La cantidad de precondiciones sin resolver en π_i indica lo que queda hasta alcanzar un plan solución. No se trata de una medida exacta de coste pues el añadir una nueva acción puede resolver varias precondiciones a la vez (por lo que podría sobrestimar $h^*(\pi_i)$) y también suele añadir nuevas precondiciones a resolver. En general el número de precondiciones sin resolver no sobrestima $h^*(\pi_i)$ aunque no está garantizado.
- La cantidad de amenazas que haya en π_i es otro indicativo de la dificultad para llegar desde π_i a un plan solución. No se suele incluir como parte de $h(\pi_i)$ porque es más probable que sobrestime. Aunque sí se suele usar como factor adicional a considerar.

Para seleccionar el fallo a resolver existen diferentes estrategias. Básicamente consisten en elegir una amenaza o una precondición pendiente de resolver.

- Una estrategia de pila (LIFO) consistente en elegir el último fallo añadido a F_i.
- Priorizar los fallos siguiendo el orden:
  1. Amenazas que no se pueden resolver: se ha llegado a un nodo que no se puede expandir y hay que continuar por otra rama del árbol.
  2. Amenazas que solo se pueden resolver de una forma.
  3. Precondiciones.
  4. Amenazas que se pueden resolver de varias formas.

- Elegir el fallo para el que existan menos alternativas de resolución. En este caso hay que calcular el número de alternativas de resolución para cada fallo de F_i lo que suele tener un coste computacional elevado.
- Elegir primero los fallos con una o ninguna forma de resolverse y luego aplicar una estrategia de LIFO. La ventaja de esta estrategia es que, si bien no es siempre fácil saber cuantas alternativas de resolución tiene un fallo, el calcular si se puede resolver de una forma o de varias (o si no tiene solución), sí es sencillo. Esta estrategia se denomina Zero-LIFO o ZLIFO y se puede combinar con la priorización de fallos:
  1. Elegir una amenaza con estrategia LIFO.
  2. Elegir una precondición irresoluble (cero formas de resolución).
  3. Elegir una precondición con una alternativa de resolución.
  4. Elegir una precondición con múltiples alternativas de resolución mediante una estrategia LIFO.

2.2.2. Planificación basada en grafos

Un paradigma diferente para la resolución de problemas de planificación clásica es la utilización de grafos de planificación. La planificación basada en grafos usa los siguientes procesos y conceptos:

- Un **grafo de planificación** sobre un problema de planificación $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ es un grafo dirigido multinivel que alterna, niveles de nodo literal (fluents) y niveles de nodo acción. Un grafo de planificación representa la evolución de un problema de planificación en cada instante de tiempo o nivel del grafo a medida que se aplican acciones de $\mathcal{A}$.
  - En el caso particular de la iteración 0 (la inicial que representa el instante $t = 0$) se tiene: $L_0 = \mathcal{I}$ y $A_0 = \{a | \forall p \in pre(a) : p \in \mathcal{I}\}$. El nivel de nodo literal L_0 contiene los fluents correspondientes al estado inicial. Y el nivel de nodo acción A_0 contiene las acciones que son aplicables en L_0, es decir, cuyas precondiciones se satisfacen con los literales de L_0.
  - En un caso general, L_i es el conjunto de fluents que aparecen tras i iteraciones o puntos de tiempo en el instante $t = i$: $L_i = L_{i-1} \cup \{eff(a) | \forall a \in A_{i-1}\}$.
  - Y se denota como A_i al conjunto de acciones aplicables tras i iteraciones o puntos de tiempo en el instante $t = i$: $A_i = A_{i-1} \cup \{a | \forall p \in pre(a) : p \in L_i\}$
- El índice g indica el primer nivel literal L_g donde se consiguen todos los objetivos: $\mathcal{G} \subseteq L_g$. Al llegar a este punto se dispone de un grafo que podría contener una solución para el problema.
- El índice f indica un **nivel literal del punto fijo** L_f en el que no se han añadido nuevos fluents respecto al nivel anterior L_{f-1} ni es posible añadir nuevas acciones a partir de los fluents de L_f.

- El primer nivel de acción del grafo en el que aparece una acción a, A_i, indica el primer instante en el que a se podría ejecutar. Todas las acciones contenidas en A_i se podrían ejecutar en el instante de tiempo i. Pero esto no significa que se puedan realizar al mismo tiempo. En el caso de dos acciones a_i y a_j que no pueden ocurrir al mismo tiempo se dice que son **acciones mutex** [3]. Dos acciones a_i y a_j se dice que son mutex en un nivel t del grafo en las siguientes situaciones:
  - Dos acciones tienen **efectos inconsistentes** si los efectos a_i y a_j afectan a la misma variable estado asignándole valores diferentes, es decir, los efectos de a_i son inconsistentes con los efectos de a_j.

$$\left.\begin{array}{l}\langle v, d\rangle \in eff(a_i) \\ \langle v, d'\rangle \in eff(a_j)\end{array}\right\} d \neq d' \tag{2.2}$$

  - Se produce una **interferencia** entre dos acciones cuando un efecto de una acción a_i modifica una variable estado que está en una precondición de otra acción a_j asignándole un valor diferente, es decir, un efecto de a_i es inconsistente con una precondición de a_j.

$$\left.\begin{array}{l}\langle v, d\rangle \in eff(a_i) \\ \langle v, d'\rangle \in pre(a_j)\end{array}\right\} d \neq d' \tag{2.3}$$

  - **Necesidades competitivas**: las acciones a_i y a_j tienen precondiciones que son mutex en el nivel $t-1$.
- Al igual que en el caso de las acciones se puede tener **literales (fluents) mutex**. Dos fluents f_i y f_j son mutex en el nivel t si cada acción que consigue f_i en el nivel $t-1$ es mutex con cada acción que consigue f_j en el nivel $t-1$. Es decir todas las formas de conseguir f_i son mutex con las formas de conseguir f_j en $t-1$.

Para problemas sencillos, una aproximación basada en grafos de planificación permite obtener planes solución de forma eficiente. Pero a medida que aumenta la complejidad del problema, el número de mutex se incrementa considerablemente, haciendo que esta aproximación resulte inviable.

El planificador basado en grafos más representativo es **Graphplan** [3] que tiene dos fases diferenciadas:

1. La fase de **expansión** genera un grafo de planificación hasta que se alcanza el nivel L_g en el cual se encuentran todos los objetivos y ninguno de ellos es mutex con otro objetivo.
2. La fase de **extracción** de la solución consiste en realizar una búsqueda regresiva sobre el grafo a partir de los objetivos alcanzados en L_g. Dado un conjunto de objetivos en el nivel t se selecciona un conjunto de acciones en el nivel A_{t-1} que los consigan y no sean mutex entre ellas. Las precondiciones de dichas acciones forman entonces un conjunto de subobjetivos en el nivel $t-1$. Si el conjunto de objetivos en el nivel $t-1$ no se puede conseguir sin que sean mutex entre ellos, se intenta un nuevo conjunto de acciones. Si no se consigue llegar al nivel $t=0$, se vuelve a la fase de expansión para añadir un nuevo nivel literal y un nuevo nivel acción.
3. Si se alcanza un nivel L_f sin haber encontrado un plan válido, el proceso finaliza indicando que no existe solución.

2.2.3. Heurística

Una necesidad común en los procesos de planificación es dirigir la búsqueda hacia la solución de la forma más rápida posible aumentando la eficiencia del proceso [17]. La orientación del proceso de búsqueda se realiza mediante una estimación del coste para llegar al objetivo, esto es, un valor heurístico. El resultado es un planificador heurístico.

Tradicionalmente las heurísticas para problemas de planificación dependían directamente de las características del problema, dando así lugar a heurísticas dependientes del dominio. Las investigaciones de la comunidad de planificación en los últimos años se han centrado en el desarrollo de funciones heurísticas independientes del dominio.

Los desarrollos obtenidos en la aproximación de planificación basada en grafos ha permitido diseñar nuevas heurísticas genéricas e independientes del dominio a la vez que son heurísticas suficientemente informadas para guiar el proceso de forma eficiente. La idea general para diseñar una heurística consiste en formular una versión simplificada (o relajada) del problema de modo que el coste de la resolución del problema simplificado se toma como una estimación del coste de resolver el problema original. Una forma de diseñar una heurística es usar la planificación basada en grafos eliminando las restricciones que imponen los mutex.

Para obtener el problema simplificado o relajado $\mathcal{R} = (\mathcal{A}_\mathcal{R}, \mathcal{I}, \mathcal{G})$ a partir del problema original $\mathcal{P} = (\mathcal{A}, \mathcal{I}, \mathcal{G})$ se genera $\mathcal{A}_\mathcal{R}$ partiendo de $\mathcal{A}$. Para resolver el problema relajado se construye un **grafo de planificación relajado** usando las acciones del conjunto $\mathcal{A}_\mathcal{R}$ y permitiendo que en un nivel literal L_i aparezca una variable estado con valores diferentes como $\{\langle v, d_1\rangle, \langle v, d_2\rangle\} \subseteq L_i$ con $d_1 \neq d_2$. De esta forma desaparecen los mutex, es decir, se eliminan todas las restricciones para la ocurrencia simultánea de dos o más acciones, dando así lugar a una versión relajada del problema.

Las heurísticas extraídas a partir de un grafo de planificación relajado se basan en la estimación del coste individual de alcanzar un literal l partiendo de un estado s ($g_s(l)$) se calcula como:

$$g_s(l) = \begin{cases} 0 & l \in s \\ min_{a \in A(l)}[1 + g_s(pre(a))] & l \notin s \end{cases} \tag{2.4}$$

donde $A(l)$ representa las acciones que generan l y $g_s(pre(a))$ es el coste de conseguir el conjunto de todas las precondiciones de la acción a desde el estado s. El valor se actualiza cada vez que una acción a_i añade el literal l: $g_s(l) = min_{a_i \in A(l)}[g_s(l), 1 + g_s(pre(a_i))]$.

La estimación $g_s(pre(a))$ que representa el coste de conseguir las precondiciones de a ó $g_s(\mathcal{G})$ que representa el coste de conseguir los objetivos del problema lo denominaremos $g_s(C)$, donde C representa un conjunto de literales; este valor se puede calcular de diferentes formas a partir de los costes de conseguir los componentes del conjunto dando lugar a distintas heurísticas:

- La heurística suma h^{add} se calcula como $g_s^{add}(C) = \sum_{l \in C} g_s(l)$. Se trata de una heurística no admisible pero muy informativa.

- La heurística máximo h^{max} el cálculo es $g_s^{max}(C) = max_{l \in C} g_s(l)$. El resultado es una heurística admisible pero poco informativa.

Muchos planificadores heurísticos utilizan un grafo de planificación relajado para estimar el coste del plan. Uno de los planificadores más eficientes es **FF** (*Fast Forward*) [14]. FF calcula una solución (plan relajado) al problema simplificado $\mathcal{R}$ sobre el grafo de planificación relajado. La longitud del plan relajado se usa como estimación de la dificultad del problema. Dicha estimación se usa posteriormente para controlar un proceso de búsqueda local similar al ascenso de colinas (*hill-climbing*). FF combina la búsqueda local con una búsqueda en anchura para evitar mínimos locales. También usa el plan relajado como guía para seleccionar las acciones durante la búsqueda en anchura con el objetivo de reducir el factor de ramificación.

2.3. Planificación temporal

La planificación clásica asume que las acciones son instantáneas y no tienen duración. Pero en la modelización de problemas reales es necesario poder representar que las acciones tienen una duración determinada. Por ejemplo, una acción para cargar un paquete en un camión no dura lo mismo que una acción para conducir un camión de Valencia a Madrid.

La planificación temporal aborda problemas donde las acciones están asociadas a una duración. En el área de planificación, estas acciones se llaman **acciones durativas**. Al introducir el concepto de temporalidad las acciones dejan de ser puntuales y esto también afecta al instante en el que las acciones durativas requieren sus precondiciones o consiguen sus efectos. Aunque existen varios modelos de acciones durativas, la comunidad de planificación ha asumido el llamado modelo temporal no conservativo como un estándar para la definición de problemas de planificación temporal [7].

El modelo temporal no conservativo define los siguientes elementos de una acción durativa:

- **Duración**: es un valor numérico que representa la duración de una acción. Las duraciones se expresan en la definición de los operadores del problema. Si se expresa un valor estático como duración del operador entonces todas las acciones (instanciaciones del operador) tendrán asociada la misma duración. También es posible definir una función o expresión aritmética en términos de los parámetros del operador como duración del operador; en este caso, cuando se instancia el operador y se genera la correspondiente acción se calcula la duración de dicha acción. En este caso se dice que el operador tiene asociada una duración dinámica cuyo valor concreto dependerá del valor de los parámetros en la instanciación del operador.
- **Precondiciones *at start***: es el conjunto de precondiciones que deben cumplirse al inicio de la acción.
- **Precondiciones *over all***: es el conjunto de precondiciones que deben satisfacerse durante todo el intervalo de duración de la acción.

- **Precondiciones *at end***: son precondiciones que deben cumplirse al término de la acción; es decir, una vez haya transcurrido la duración de la acción, momento en el cual es necesario que se satisfagan las precondiciones *at end* para completar la acción.
- **Efectos *at start***: es el conjunto de efectos que se producen al inicio de la acción.
- **Efectos *at end***: es el conjunto de efectos que se producen al término de la acción; es decir, una vez haya transcurrido la duración de la acción, momento en el cual la acción se ha completado.

El diagrama de la figura 2.1 muestra una acción vista temporalmente con sus extremos inicial (I) y final (F) separados en el tiempo por la duración de la acción. Las flechas que llegan a la acción indican en qué momento de la acción se debe cumplir cada tipo de precondición. Y las flechas salientes indican en qué momento se produce cada efecto.

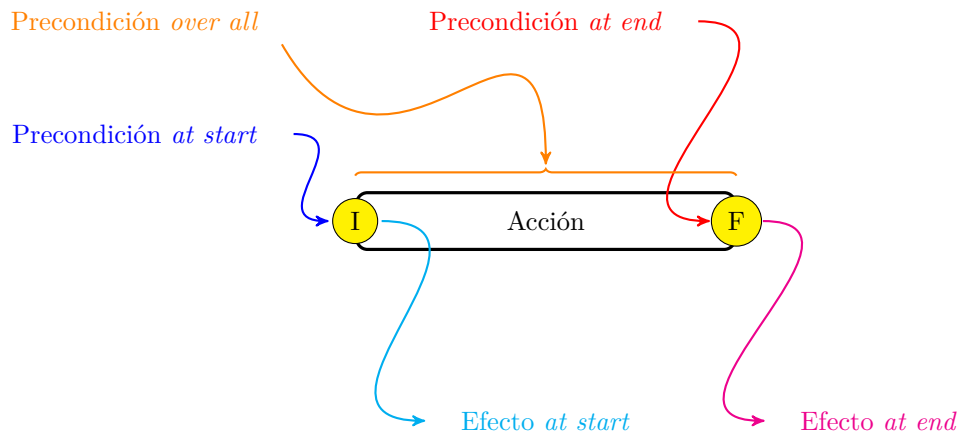

Figura 2.1: Precondiciones y efectos temporales

En un modelo temporal, las acciones dejan de ser puntuales para tener una duración lo que hace que cambie el criterio de optimización y se busque minimizar la duración del plan total. Por ejemplo, supongamos una acción para conducir un camión de una ciudad origen a una ciudad destino. La forma más simple de expresar la duración de la acción es establecer una duración fija. Una opción más flexible es definir la duración con una expresión como $\frac{distancia(ciudad-origen,ciudad-destino)}{velocidad(camión)}$ que devolverá un distinto valor de duración en función de los valores de los parámetros del operador, ciudad-origen, ciudad-destino y camión.

Dada una acción concreta a, diremos que tiene una duración a^{dur} y comienza en el instante a^{ini}, por lo que terminará en $a^{fin} = a^{ini} + a^{dur}$. Para indicar que un evento (como un efecto) sucede inmediatamente después de otro (como una precondición) usamos la constante ϵ. La cantidad de tiempo ϵ es un valor

diferencial muy pequeño comparado con la duración de las acciones de forma que la suma de todos los ϵ de un plan es menor que una unidad de tiempo. La constante ϵ se utiliza para evitar problemas de sincronía entre la ocurrencia de un fluent $\langle v, d\rangle$ que actúa como precondición y un fluent $\langle v, d'\rangle, d \neq d'$ que se genera como efecto en el *mismo instante de tiempo*. Dado que, por definición, no es posible tener en el mismo instante de tiempo dos fluents contradictorios, se emplea el tiempo ϵ para indicar que el efecto se produce un tiempo diferencial ϵ después del cumplimiento de la precondición. Los tiempos correspondientes a cada efecto y precondición son los indicados en la tabla 2.1.

Indicativo temporal	Precondición	Efecto
at start	a^{ini}	$a^{ini} + \epsilon$
over all	Entre a^{ini} y a^{fin}	
at end	a^{fin}	$a^{fin} + \epsilon$

Cuadro 2.1: Tiempo para condiciones y efectos de una acción durativa

Tomando como ejemplo una acción para conducir un camión de una ciudad a otra, el camión debe estar en la ciudad origen en el instante a^{ini} (precondición *at start*) y debe haber un conductor durante todo el tiempo que dura la acción, desde a^{ini} hasta a^{fin} (precondición *over all*). Por otro lado, en cuanto se inicia la acción, el camión deja de estar en la ciudad origen en un instante $a^{ini} + \epsilon$ (efecto *at start*) y alcanzará la ciudad destino al final de la duración de la acción en el instante $a^{fin} + \epsilon$ (efecto *at end*). Si, por ejemplo, fuese necesario pasar un control de aduanas para alcanzar la ciudad destino, ésta sería una precondición *at end*, ya que solo es requerida en el instante a^{fin}. Además se considerará otras dos acciones relacionadas, subir un conductor al camión y pasar por aduanas. A continuación se detallan los componentes necesarios para el ejemplo de conducir un camión de una ciudad a otra objetos, variables estado, fluents y acciones.

- Se usan diferentes objetos: un conductor `d1` y un camión `t1` que realizarán las diferentes acciones. Además se utilizan otros objetos para asignar valores a las variables estado:
  - La localización inicial y final del camión para la acción de conducir: `origen` y `destino` respectivamente.
  - Un objeto auxiliar para indicar que un camión está desplazándose de una localización a otra: `carretera`.
  - Un objeto auxiliar para indicar que un conductor está subiendo a un camión: `subiendo`.
  - Dos objetos auxiliares para indicar si se ha pasado por aduana o no: `pasada` y `pendiente` respectivamente.
  - Otros dos objetos indican si el puesto del conductor de un camión tiene un conductor o está vacío: `ocupado` y `libre` respectivamente.
- Para indicar la localización del camión `t1` se usa la variable estado $localizado-t1$. Dado que el camión `t1` puede estar en la localización origen, destino o en la carretera durante la ejecución del plan aparecerán los siguientes fluents con la variable estado $localizado - t1$:

  - $\langle localizado - t1, origen \rangle$: el camión `t1` está en la localización de origen.
  - $\langle localizado - t1, destino \rangle$: el camión `t1` está en la localización de destino.
  - $\langle localizado - t1, carretera \rangle$: el camión `t1` está circulando por la carretera.
- Existe otra variable estado similar para indicar la localización del conductor `d1`: $localizado - d1$. En este caso el conductor `d1` estará durante la ejecución del plan en la localización origen, subiendo a un camión o en el camión `t1` por lo que aparecerán los siguientes fluents con la variable estado $localizado - d1$:
  - $\langle localizado - d1, origen \rangle$: el conductor `d1` está en la localización de origen.
  - $\langle localizado - d1, subiendo \rangle$: el conductor `d1` está subiendo a un camión.
  - $\langle localizado - d1, t1 \rangle$: el conductor `d1` está en el puesto de conducción del camión `t1`.
- Para saber si el camión `t1` ha pasado o no la aduana se usa la variable estado $aduana - t1$. Esta variable estado solo toma dos valores que indican si se ha pasado o no la aduana formando los fluents:
  - $\langle aduana - t1, pendiente \rangle$: el camión `t1` está pendiente de pasar por aduana.
  - $\langle aduana - t1, pasada \rangle$: el camión `t1` ha pasado la aduana.
- También es necesario saber si el camión `t1` tiene conductor o no lo que se expresa con la variable estado $puesto\_conductor - t1$. Esta variable estado nuevamente tiene dos valores, si tiene conductor o no, por lo que aparecerán los siguientes fluents:
  - $\langle puesto\_conductor - t1, libre \rangle$: no hay conductor en el camión `t1`.
  - $\langle puesto\_conductor - t1, ocupado \rangle$: hay un conductor en el camión `t1`.
- *Camión en origen* consiste en tener el camión `t1` localizado en el punto de origen sin conductor, es decir, los fluents $\langle localizado - t1, origen \rangle$ y $\langle puesto\_conductor - t1, libre \rangle$.
- *Conductor en origen* es un fluent que consiste en tener el conductor `d1` localizado en el punto de origen, es decir, el fluent $\langle localizado - d1, origen \rangle$.
- *Subir* es la acción de subir el conductor al camión. Esta acción necesita que el camión esté *over all* en una misma localización, *at start* que el conductor y el camión estén en el mismo lugar y *at start* que el puesto del conductor esté libre. Su efecto principal es que *at end* se que el conductor esté dentro del camión. Además *at start* el conductor pasa a estar subiendo al camión y el puesto del conductor está ocupado, esto permite evitar que de forma

simultánea múltiples conductores puedan subir a un mismo camión o que un conductor pueda empezar a subir a distintos camiones. Esta acción tiene una duración de $a^{dur}_{subir} = 2$. En este caso se instancia como subir el conductor `d1` al camión `t1` por lo que se tienen las precondiciones y efectos de la tabla 2.2.

Efecto o precondición	Fluent	Relaciones
Precondición *over all*	$\langle localizado - t1, origen \rangle$	
Precondición *at start*	$\langle localizado - d1, origen \rangle$	
Precondición *at start*	$\langle puesto\_conductor - t1, libre \rangle$	
Efecto *at start*	$\langle puesto\_conductor - t1, ocupado \rangle$	
Efecto *at start*	$\langle localizado - d1, subiendo \rangle$	
Efecto *at end*	$\langle localizado - d1, t1 \rangle$	Precondición de conducir

Cuadro 2.2: Precondiciones y efectos de la acción *subir*

- *Conducir* es la acción de conducir un camión de un lugar a otro cuyas precondiciones son *at start* que el camión esté en la localización de origen, que tenga conductor *over all* y que se haya pasado por la aduana *at end*. La acción de conducir tiene dos efectos *at start*, el camión deja de estar en la localización origen y no ha pasado la aduana. Y como efecto *at end* el camión llega a la localización destino. Esta acción tiene una duración de $a^{dur}_{conducir} = 6$. En este caso se instancia como que el conductor `d1` conduzca el camión `t1` de la localización origen a la destino generando las precondiciones y efectos de la tabla 2.3.

Efecto o precondición	Fluent	Relaciones
Precondición *at start*	$\langle localizado - t1, origen \rangle$	
Precondición *over all*	$\langle localizado - d1, t1 \rangle$	Efecto de subir
Precondición *at end*	$\langle aduana - t1, pasada \rangle$	Efecto de aduana
Efecto *at start*	$\langle aduana - t1, pendiente \rangle$	Precondición de aduana
Efecto *at start*	$\langle localizado - t1, carretera \rangle$	Precondición de aduana
Efecto *at end*	$\langle localizado - t1, destino \rangle$	Objetivo

Cuadro 2.3: Precondiciones y efectos de la acción *conducir*

- La acción *aduana* consiste en pasar por aduana. Sus precondiciones *at start* son que el camión esté en carretera y no haya pasado por la aduana. Su efecto *at end* es haber pasado por la aduana. La duración de la acción de pasar por aduana es $a^{dur}_{aduana} = 3$. En el ejemplo, la acción instanciada es pasar por la aduana el camión `t1` conducido por el conductor `d1` por lo que se tienen las precondiciones y efectos de la tabla 2.4.

Efecto o precondición	Fluent	Precondición de
Precondición *at start*	$\langle localizado - t1, carretera \rangle$	Efecto de conducir
Precondición *at start*	$\langle aduana - t1, pendiente \rangle$	Efecto de conducir
Efecto *at end*	$\langle aduana - t1, pasada \rangle$	Precondición de conducir

Cuadro 2.4: Precondiciones y efectos de la acción *aduana*

Se observa que las acciones están relacionadas entre ellas por los diferentes fluents en precondiciones y efectos tal como se detalla a continuación:

- El fluent $\langle localizado - d1, t1 \rangle$ es precondición *over all* de la acción de conducir y es efecto *at end* de la acción de subir. Por este motivo la acción de subir se debe completar antes de empezar la acción de conducir: $a^{fin}_{subir} + \epsilon < a^{ini}_{conducir}$.
- El fluent $\langle aduana - t1, pasada \rangle$ es precondición *at end* de la acción de conducir y es efecto *at end* de la acción de aduana. De esta forma la acción de pasar aduana debe terminar antes que la de conducir: $a^{fin}_{aduana} + \epsilon < a^{fin}_{conducir}$.
- Los fluents $\langle localizado - t1, carretera \rangle$ y $\langle aduana - t1, pendiente \rangle$ son precondición *at start* de la acción de aduana y son efectos *at start* de la acción de conducir. Así que la acción de conducir debe empezar antes que al de aduana: $a^{ini}_{conducir} + \epsilon < a^{ini}_{aduana}$

Estas relaciones forman el sistema de inecuaciones 2.5 en el que se reflejan las relaciones entre los tiempos iniciales y finales de cada acción. Sabiendo que $a^{fin} = a^{ini} + a^{dur}$ se obtiene la segunda parte de la ecuación en la que se ha reducido el número de variables.

$$\left.\begin{array}{l} a^{fin}_{subir} + \epsilon < a^{ini}_{conducir} \\ a^{fin}_{aduana} + \epsilon < a^{fin}_{conducir} \\ a^{ini}_{conducir} + \epsilon < a^{ini}_{aduana} \end{array}\right\} \quad \left.\begin{array}{l} a^{ini}_{subir} + a^{dur}_{subir} + \epsilon < a^{ini}_{conducir} \\ a^{ini}_{aduana} + a^{dur}_{aduana} + \epsilon < a^{ini}_{conducir} + a^{dur}_{conducir} \\ a^{ini}_{conducir} + \epsilon < a^{ini}_{aduana} \end{array}\right\} \tag{2.5}$$

Según la relación de las duraciones de las diferentes acciones, nos encontraremos ante un problema con solución o sin solución. En este caso tiene solución, por ejemplo, la solución que se muestra en que se muestra en la figura 2.2 con los tiempos de la tabla 2.5.

Acción	Tiempo inicial	Duración	Tiempo final
Subir	2	2	4
Conducir	5	6	11
Aduana	7	3	10

Cuadro 2.5: Ejemplo de relaciones temporales: tiempos y duraciones

Este modelo temporal no conservativo permite definir una modelización discreta del tiempo estableciendo las transiciones de estado entre dos puntos del intervalo de duración de la acción, el inicial y el final. Otros modelos temporales

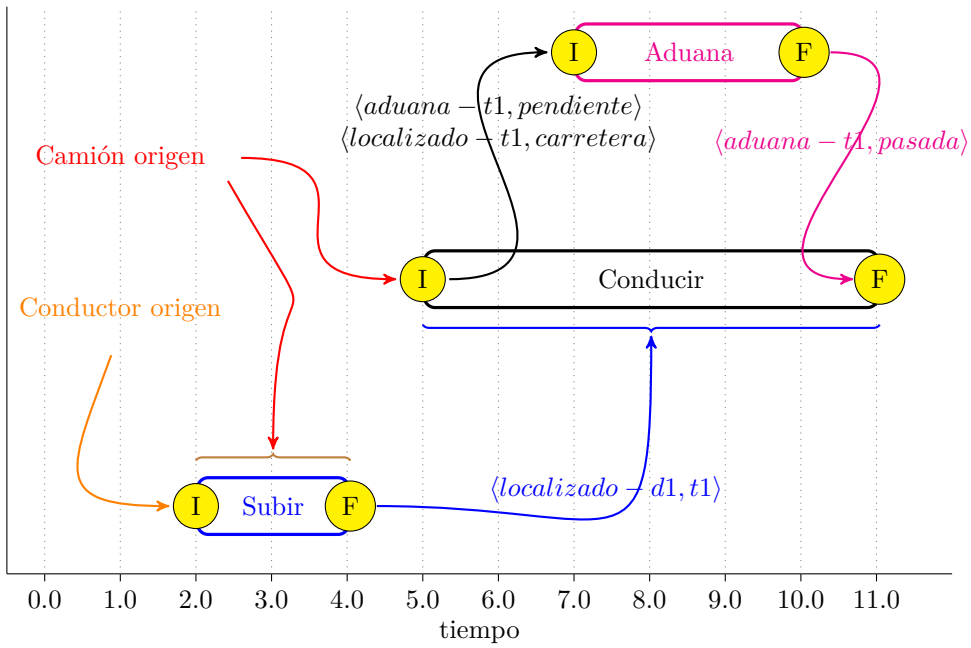

Figura 2.2: Ejemplo de relaciones temporales

más sofisticados permiten expresar precondiciones que solo se tienen que cumplir en una parte del intervalo de la acción durativa [19] o permiten acceder a distintos puntos de tiempo dentro del intervalo de la acción [10]. Sin embargo, los planificadores que utilizan un modelo temporal más flexible y sofisticado suelen tener un peor rendimiento en problemas de gran tamaño.

En el resto de esta sección nos centraremos exclusivamente en planificadores que utilizan el modelo temporal no conservativo, al igual que nuestra aproximación TFLAP. En el estado del arte existen varios planificadores temporales, tres de los más populares y con los que se comparará TFLAP son LPG, TFD y OPTIC.

2.3.1. LPG

LPG [8] es un planificador independiente del dominio que se presentó en la tercera competición de planificación (IPC[1]). LPG es un planificador incremental que genera una secuencia de planes solución cada uno de los cuales mejora la calidad de los planes solución anteriores.

LPG se basa en un método de búsqueda local estocástico y en una variación de la representación basada en grafos (ver sección 2.2.2) denominada grafo de acciones, en una primer versión de LPG, o **grafos de acciones temporales** (*Temporal Action Graphs* o *TA-Graphs*) en una versión mejorada de LPG.

[1] *International Planning Competition* del año 2002, http://icaps-conference.org/index.php/Main/HomePage

Aunque los métodos de búsqueda local no garantizan la obtención de la solución óptima, LPG ha demostrado que es capaz de obtener buenos resultados en la búsqueda de planes. En un *TA-Graph* los nodos acción A_i se etiquetan con valores temporales que indican el tiempo más temprano en que el literal correspondiente es verdadero. Durante el proceso de búsqueda se mantiene un conjunto de restricciones de ordenación que permite manejar las acciones mutex y representar las restricciones temporales implícitas en las relaciones causales del plan.

LPG utiliza una heurística que asigna pesos a los elementos del *TA-Graph*. La evaluación de los *TA-Graphs* se basa en la estimación del número de pasos de búsqueda necesarios para encontrar un plan solución $\mathcal{P}$, la duración estimada de $\mathcal{P}$ y el coste de ejecución (basado en otros criterios distintos del tiempo) que pueda tener $\mathcal{P}$.

Si bien LPG realiza una búsqueda estocástica local que es subóptima e incompleta, el planificador es capaz de encontrar un primer plan solución rápidamente que después mejora de forma incremental. Al realizar una búsqueda estocástica que depende de un valor aleatorio de una semilla, cada ejecución de LPG puede encontrar una solución diferente al mismo problema. LPG obtuvo muy buenos resultados en la tercera IPC en un amplio conjunto de problemas de test tanto en tiempo de computación para la primer solución como en la calidad del plan solución final generado por el proceso de mejora incremental.

2.3.2. TFD

TFD (*Temporal Fast Downward* [5]) es un planificador temporal basado en el planificador heurístico *Fast Downward*. TFD utiliza una búsqueda hacia adelante en un espacio de estados donde cada estado viene etiquetado con un tiempo (*time-stamped states*). El proceso de búsqueda de TFD consiste en añadir una nueva acción al plan o avanzar temporalmente hasta llegar al tiempo final de una acción ya existente. En caso de añadir una nueva acción se aplican los efectos *at start* de la acción, se anota el instante temporal en el que se producirán los efectos finales y se anotan las precondiciones *over all*. En el caso de avanzar hasta el final de una acción ya existente se aplican los efectos *at end* de la acción.

TFD emplea una búsqueda voraz mejorada con una evaluación heurística diferida. Se trata de un proceso integrado de selección de acciones y planificación temporal que comprueba de forma conjunta los valores de las variables estado, efectos y precondiciones.

TFD es un planificador que devuelve buenos resultados en cuanto a la duración de los planes generados (criterio para medir la calidad de los planes temporales) pero suele resolver un bajo porcentaje de la colección de problemas que se emplean en las competiciones de planificación.

2.3.3. OPTIC

OPTIC [2] es un planificador de orden parcial que utiliza un proceso de búsqueda hacia adelante y que ha demostrado un excelente rendimiento en las

competiciones de planificación además de 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 heurística independiente del dominio que es muy eficiente.

OPTIC combina un algoritmo de **búsqueda hacia adelante** con una **estructura de planificador de orden parcial**, donde cada nodo del proceso de búsqueda es un plan parcial $\pi = \langle A, O, B, C, F \rangle$, tal como se ha definido en la sección 2.2.1. Para utilizar una búsqueda hacia adelante, OPTIC utiliza el concepto de ***estado frontera***, que es el estado resultante de la simulación de la ejecución del plan de un nodo del árbol de búsqueda. Dado el nodo de plan parcial a analizar $\pi_a = \langle A_a, O_a, B_a, C_a, F_a \rangle$ se aplican los efectos de las acciones del conjunto A_a siguiendo el orden indicado en el conjunto O_a. OPTIC utiliza el *estado frontera* para dos tareas:

1. Para guiar el proceso de búsqueda seleccionando el nodo a expandir. El *estado frontera* de cada nodo permite usar heurísticas basadas en estados que son más informativas que las heurísticas basadas en planes.
2. En el proceso de expansión de un nodo para generar nodos sucesores. OPTIC utiliza el *estado frontera* para determinar el conjunto de acciones que se pueden añadir al nodo a expandir para generar los nodos sucesores.

Al utilizar el *estado frontera* para determinar el conjunto de acciones que se pueden añadir al nodo a expandir equivale a un proceso de poda que reduce el número de nodos sucesores y el espacio de búsqueda. Adicionalmente facilita la generación de los nodos sucesores acelerando la búsqueda. 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 satisfagan en el *estado frontera* aunque se cumplieran en algún punto intermedio del plan resultando una búsqueda incompleta. Para garantizar la completitud del proceso de búsqueda OPTIC recurre a un mecanismo de *backtracking*.

OPTIC es un planificador eficiente aunque elimina parte de la flexibilidad de la planificación de orden parcial. Se trata de una solución de compromiso entre búsqueda hacia adelante propia de un planificador basado en estados (ver sección 2.2) y el principio de menor compromiso que utilizan los planificadores de orden parcial (ver sección 2.2.1).

OPTIC utiliza un algoritmo de ascenso de colinas similar a FF (véase la sección 2.2.3). Cuando el algoritmo de ascenso de colinas no consigue avanzar OPTIC pasa a un algoritmo voraz de primero el mejor. De esta forma está orientado a alcanzar una solución de forma rápida pero deja en segundo lugar la calidad de la solución.

2.4. Lenguaje de planificación

Para describir los elementos de un problema de planificación se utiliza un lenguaje específico de planificación. Uno de los primeros lenguajes de planificación fue STRIPS (*STandford Research Institute Problem Solver* [6]). A partir

de STRIPS se han desarrollado otros lenguajes, como PDDL (*Planning Domain Definiton Languaje* [9]) el cual se ha convertido en el lenguaje de referencia para la mayoría de planificadores.

PDDL utiliza dos descripciones:

- La descripción de las posibles acciones a realizar así como los tipos de los objetos que intervienen en el problema. Estos elementos se definen en el fichero del **dominio**.
- Junto con la descripción del dominio, se define el problema concreto a resolver. La descripción de un problema requiere especificar los objetos del problema, el estado inicial y los objetivos a conseguir. Todo esto se describe en el fichero del **problema**.

A lo largo del tiempo el lenguaje PDDL se ha ido refinando con sucesivas versiones. En este trabajo se utiliza la versión 3.1 que se comenta a continuación.

2.4.1. PDDL 3.1

Un problema $P = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ se describe a través de los ficheros de dominio y problema mencionados anteriormente.

La descripción del dominio tiene en su encabezado la palabra identificativa **domain** junto al nombre del dominio: `(define (domain <nombre-dominio>))`. En la descripción del dominio se usan las siguientes palabras clave básicas:

- La lista de características del lenguaje a usar se describe en la sección denominada `:requirements`.
- La definición de tipos partiendo del tipo básico `object` se especifica en la sección `types`.
- Para definir valores especiales (constantes) de un tipo determinado se utiliza la sección `constants`.
- La definición de funciones y variables estado se encuentra en la sección `functions`. Tanto las funciones como las variables estado se identifican con un nombre, una serie de parámetros y el tipo de la función o variable estado.
- Los predicados se definen en la sección `predicates`. Cada predicado se identifica con un nombre y una serie de parámetros.
- Cada operador del conjunto $\mathcal{O}$ se define en una sección `action` dedicada para el operador. Cada sección `action` contiene la descripción de un único operador. Se trata del elemento más complejo de la descripción del dominio y se compone de los siguientes elementos:
  - Los parámetros del operador se indican como `parameters`.
  - El conjunto de precondiciones del operador se definen en la sección `precondition` agrupadas dentro de una conjunción `and`. Una precondición sobre un fluent se especifica con el símbolo de igualdad =.

- El conjunto de los efectos del operador se especifican en la sección `effect`, también agrupados dentro de una conjunción `and`. Los efectos relativos a fluents se describen en la sección `effect` con la palabra clave `assign`: una asignación.

El fichero de descripción del problema comienza con el encabezado **problem**: `(define (problem <nombre-problema>))`. Un problema se describe con los siguientes elementos:

- Tras el encabezado se indica el dominio asociado al problema mediante la estructura `(:domain <nombre-dominio>)`.
- Los objetos del problema se definen en la sección `objects`.
- El estado inicial $\mathcal{I}$ se especifica como una lista de fluents y predicados en la sección `init`. Cada variable estado del problema debe tener asignado un valor inicial que constituye un fluent. En el estado inicial solo se describen literales positivos, entendiendo que aquella información que no aparece explícitamente en el estado inicial se considera falsa.
- El estado objetivo $\mathcal{G}$ se describe en la sección `global-goal` que contiene los fluents y literales objetivo dentro de una conjunción `and` para indicar que se deben cumplir todos.
- La asignación de un valor a una variable estado (fluent) se especifica con el símbolo =, tanto en el estado inicial como en el objetivo.

Las variables estado son útiles para describir los atributos del problema que pueden tomar distintos valores durante la resolución del problema. Mientras que los predicados son útiles para modelar propiedades booleanas de las variables. En cualquier caso es posible modelar un dominio de planificación únicamente con fluents o con predicados.

2.4.2. PDDL 3.1 temporal

En esta sección se describen los elementos para la especificación de un dominio y problema temporal mediante el lenguaje PDDL 3.1.

- Los operadores del problema se describen mediante el constructor específico `durative-action`.
- Las precondiciones de un operador se describen en la sección `condition`. Cada una de las precondiciones será `at start`, `over all` o `at end`.

2.4.3. Ejemplo de problema temporal en PDDL 3.1

Sea un problema en el cual hay dos robots (`R1` y `R2`) que se pueden mover por cuatro habitaciones (`H1`, `H2`, `H3` y `H4`) con ciertas interconexiones entre las habitaciones y un pasillo que conecta todas ellas. Inicialmente el robot `R1` está en la habitación `H1` y el robot `R2` está en la habitación `H2`. El objetivo es que `R1` esté en `H2` y `R2` esté en `H4`. La figura 2.3 muestra gráficamente el

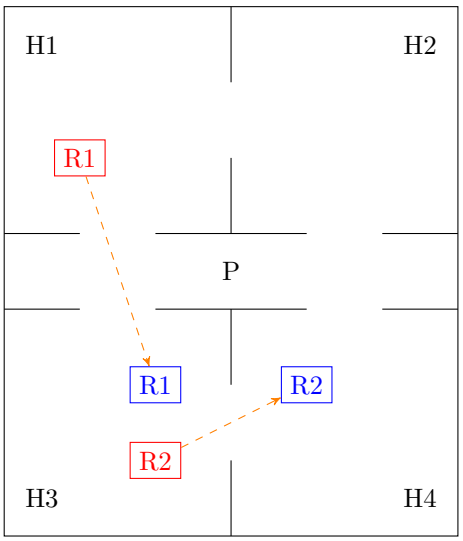

Figura 2.3: Representación gráfica del problema de robots

problema, indicando en color rojo la localización inicial de los robots y en azul la localización objetivo de los robots.

Para describir este dominio es necesario especificar el *objeto* robot, el cual se puede desplazar por un conjunto de habitaciones interconectadas. Un robot solo se puede desplazar de una habitación origen a otra destino si están conectadas y el robot está en la habitación origen, una vez concluida la correspondiente acción de movimiento, el robot pasa a estar en la habitación destino tras el tiempo correspondiente a la duración del movimiento. La descripción del dominio y problema de planificación correspondiente a este ejemplo se muestran respectivamente en las figuras 2.4 y 2.5. El dominio contiene los siguientes elementos:

- Dos tipos de objeto definidos en la sección `types`, habitación y robot.
- Una constante auxiliar de tipo habitación en la sección `constants` para indicar que un robot se está moviendo.
- La información estática de este problema se describe en la sección de predicados: `predicates`. El predicado `(conectadas ?x - hab ?y - hab)` permitirá definir los pares de habitaciones que están conectadas.
- En la sección `functions` se define:
  - La variable estado `(rob_esta ?x - robot) - hab` que indica la localización de un robot.
  - Y la función `(coste_mover ?r - robot)` que indica la duración de los movimientos de un robot.
- Este problema solo tiene un operador, el movimiento de un robot definido como `(:durative-action mover`.

- Sus parámetros son el robot `?r` y las habitaciones origen `?h1` y destino `?h2` del movimiento.
- La duración de la acción se define en la sección `duration` a través de la función `(coste_mover ?r - robot)`. De esta forma cada acción de movimiento puede tener una duración diferente para de cada robot.
- Las precondiciones del operador se definen en la sección `condition`. La precondición `(= (rob_esta ?r) ?h1)` *at start* indica que el robot debe estar en la habitación origen para poder iniciar el movimiento. Además para poder moverse entre dos habitaciones éstas deben estar conectadas lo que se indica con la precondición *over all* `(conectadas ?h1 ?h2)`.
- Los efectos se describen en la sección `effect`. Existe un primer efecto *at start* `(assign (rob_esta ?r) moviendo)` que se produce al comienzo de la acción y hace que el robot deje de estar en la habitación origen. El efecto *at end* `(assign (rob_esta ?r) ?h2)`, hace que el robot llegue a la habitación destino. De esta forma durante la acción de movimiento no se puede iniciar de nuevo un movimiento del mismo robot evitando así la generación de un plan inconsistente.

Partiendo de la descripción de dominio, el problema se describe mediante el código de la figura 2.5 el cual contiene los siguientes elementos:

- Los objetos del problema definidos en la sección `objects`. Tanto los robots `R1` y `R2` de tipo robot. Como las habitaciones `H1`, `H2`, `H3`, `H4` y el pasillo `P` de tipo habitación.
- Las conexiones de las habitaciones se definen en la sección `init` como literales de acuerdo a los predicados definidos en el dominio. Tal como se ha definido el dominio se necesita un literal para cada par de habitaciones conectadas y para cada sentido de conexión permitido. Por ejemplo para poder ir de la habitación `H1` a la `H2` y viceversa se necesitan los literales `(conectadas H1 H2)` y `(conectadas H2 H1)`.
- La localización inicial de los robots se define mediante los fluents de la sección `init (= (rob_esta R1) H1)` y `(= (rob_esta R2) H3)`.
- Los valores de las funciones del coste de movimiento también se definen en la sección `init`.
  - La función `(= (coste_mover R1) 2)` indica que el robot `R1` necesita 2 unidades de tiempo para moverse de una habitación a otra.
  - Mientras que `(= (coste_mover R2) 5)` indica que el robot `R2` necesitará 5 unidades de tiempo.
- La localización objetivo de los robots se define mediante los fluents de `global-goal (= (rob_esta R1) H3)` y `(= (rob_esta R2) H4)`.

```
(define (domain robdemo_dom)
  (:requirements :strips :typing :equality :fluents)
  (:types hab robot - object)

  (:constants
    moviendo - hab   ; Valor auxiliar por usar fluents
  )

  ; Predicados para valores constantes del problema
  (:predicates
    (conectadas ?x - hab   ?y - hab)
  )

  (:functions
    ; Variables estado
    (rob_esta   ?x - robot) - hab
    ; Coste temporal de las acciones
    (coste_mover ?r - robot)
  )


  ; Operador de movimiento
  (:durative-action mover
    :parameters (
      ?r  - robot  ; Robot a mover
      ?h1 - hab    ; Localizacion inicial
      ?h2 - hab    ; Localizacion final
    )
    :duration (= ?duration (coste_mover ?r))
    :condition (and
      (at start (= (rob_esta ?r) ?h1)) ; Inicialmente, el robot en h1
      (over all (conectadas ?h1 ?h2))  ; Las dos habitaciones, conectadas
    )
    :effect (and
      (at start (assign (rob_esta ?r) moviendo))
      (at end   (assign (rob_esta ?r) ?h2))
    )
  )
)
```

Figura 2.4: Especificación de un dominio temporal con el lenguaje PDDL 3.1

```
(define (problem robdemo_prbl)
  (:domain robdemo_dom)
  (:objects
    H1 H2 H3 H4 P - hab
    R1 R2 - robot
  )

  (:init
    ; Valores estaticos: definen el entorno, literales
    (conectadas H1 H2)
    (conectadas H1 P )
    (conectadas H2 P )
    (conectadas H3 H4)
    (conectadas H3 P )
    (conectadas H4 P )
    (conectadas H2 H1)
    (conectadas P  H1)
    (conectadas P  H2)
    (conectadas H4 H3)
    (conectadas P  H3)
    (conectadas P  H4)
    ; Valores que cambian: situacion inicial, fluents
    (= (rob_esta R1) H1)
    (= (rob_esta R2) H3)
    ; Coste de las acciones
    (= (coste_mover R1) 2)
    (= (coste_mover R2) 5)
  )

  (:global-goal (and
    ; Valores que cambian: objetivo, fluents
    (= (rob_esta R1) H3)
    (= (rob_esta R2) H4)
  ))
)
```

Figura 2.5: Especificación de un problema temporal correspondiente al dominio de la figura 2.4

Capítulo 3

Modelo de planificación temporal

En este capítulo se describe el diseño y desarrollo de la versión temporal del planificador de orden parcial FLAP, desarrollado por el grupo GRPS de la UPV. En la primera sección se describen las principales características que diferencian FLAP de un POP clásico y el funcionamiento básico de FLAP. A continuación se describen las modificaciones y extensiones realizadas en FLAP para incluir el tiempo y la gestión temporal del proceso de planificación, dando lugar así a TFLAP (temporal FLAP). En la siguiente sección se describe el proceso de búsqueda de TFLAP y las heurísticas empleadas. Por último se realiza una comparación funcional de TFLAP con OPTIC, el planificador temporal de orden parcial más eficiente que existe actualmente en el estado del arte.

3.1. Planificador FLAP

FLAP es un planificador de orden parcial desarrollado en el subgrupo de investigación GRPS (Grupo de Razonamiento en Planificación y Scheduling) del GTI-IA de la UPV (http://users.dsic.upv.es/grupos/grps/) y que constituye la base de este trabajo [23, 22]. Concretamente, FLAP utiliza un proceso de búsqueda hacia adelante y una heurística basada en estados para guiar la generación del árbol de búsqueda.

Al utilizar un esquema de búsqueda hacia adelante, algunas de las características típicas de POP no se aplican en FLAP. En el primer punto de esta sección se describen las características diferenciadoras de FLAP respecto a un POP clásico. A continuación se explica el proceso de generación del árbol de búsqueda en FLAP y la expansión de un nodo para generar nodos sucesores.

3.1.1. Características diferenciadoras de FLAP

La diferencia básica entre FLAP y un POP clásico radica en que FLAP aplica un proceso de búsqueda hacia adelante (o dirigido por datos), a diferencia de un POP clásico que aplica un proceso de búsqueda hacia atrás o dirigido por objetivos. La búsqueda hacia adelante permite usar heurísticas basadas en estados que son más informativas que las heurísticas POP. A continuación se enumeran las principales diferencias funcionales entre FLAP y un POP clásico tal como se ha definido en la sección 2.2.1.

- FLAP utiliza acciones totalmente instanciadas, por lo cual en ningún momento aparecerán relaciones de ligadura; de este modo, $B = \emptyset$ en todos los nodos del árbol de búsqueda. Antes de comenzar el proceso de planificación, FLAP realiza un proceso de *grounding* mediante el cual se genera el conjunto de acciones $\mathcal{A}$ a partir del conjunto de operadores $\mathcal{O}$ y objetos del problema, así como el conjunto de fluents que se manejarán durante la resolución del problema. Se trata de un proceso común a la mayoría de planificadores que permite aumentar la eficiencia en tiempo de cómputo del proceso de búsqueda.

- Por similitud con un POP tradicional, diremos que la acción ficticia inicial a_0 que representa el estado inicial del problema $\mathcal{I}$ se añade al nodo raíz del árbol de búsqueda. Mientras que la acción ficticia final a_n que representa el estado final del problema $\mathcal{G}$ no se añade al nodo raíz del árbol de búsqueda, dado que no se trata de un proceso de búsqueda dirigido por los objetivos. Cada vez que se añade un nodo al árbol de búsqueda, FLAP se encarga de comprobar si satisface los objetivos del problema.

- La generación de un nodo en el árbol de búsqueda consiste en añadir una nueva acción a_i en el plan parcial del nodo padre siempre y cuando las precondiciones de a_i se soporten con las acciones del plan parcial padre mediante la inserción de los correspondientes enlaces causales y las amenazas sean resolubles generando un conjunto de ordenaciones consistente. Se genera un nodo sucesor por cada combinación de acciones, enlaces causales y ordenaciones.

- Los planes contenidos en los nodos del árbol de búsqueda de FLAP no tienen tareas pendientes de resolver ($F = \emptyset$) dado que las precondiciones de todas las acciones están resueltas y el plan está libre de amenazas.

- FLAP utiliza une heurística basada en estados para guiar el proceso de búsqueda. Dado que cada nodo del árbol de búsqueda representa un plan parcial sin precondiciones no resueltas ni conflictos, FLAP calcula el estado que resulta de simular la ejecución del plan de cada nodo del árbol de búsqueda, denominado *estado frontera* y cuya generación se describe en 3.3.1.

Tal y como se ha comentado, un nodo de FLAP no tiene ligaduras de variables ($B = \emptyset$) ni tareas pendientes de resolver ($F = \emptyset$) por lo que en vez de representar un plan parcial como $\pi = \langle A, O, B, C, F\rangle$, se representa mediante la tupla $\pi = \langle A, O, C\rangle$.

3.1.2. Generación del árbol de búsqueda

El nodo raíz del árbol de búsqueda contiene únicamente la acción inicial ficticia a_0, de modo que el plan del nodo raíz no contiene enlaces causales ni ordenaciones. De esta forma, el nodo raíz se representa como $\pi_0 = \langle A_0, O_0, C_0 \rangle = \langle \{a_0\}, \emptyset, \emptyset \rangle$.

Una iteración del proceso de búsqueda en FLAP es similar a una iteración de un proceso de búsqueda clásico, tal como se muestra en la figura 3.1, y consiste en los siguientes pasos:

1. Se escoge un nodo entre la lista de nodos abiertos del árbol (lista *ABIERTOS*). Inicialmente, *ABIERTOS* solo contendrá el nodo raíz del árbol. El nodo seleccionado de *ABIERTOS* se denomina plan base: $\pi_{base} = \langle A_{base}, O_{base}, C_{base} \rangle$. Dado que los nodos se insertan en *ABIERTOS* según el valor de la función de evaluación detallada en la sección 3.3.3, el proceso de selección de un nodo consiste en extraer el primer nodo de *ABIERTOS*.

2. Se genera un nodo $\pi_{sucesor}$ sucesor de π_{base} por cada combinación de acción que se puede insertar en π_{base}, enlaces causales y relaciones de orden que sean consistentes con las relaciones ya existentes en el plan. Es decir, por cada acción a_i cuyas precondiciones se soportan con las acciones de A_{base} $(\forall f \in pre(a_i), \exists a_j \in A_{base} \mid f \in eff(a_j))$ y si las amenazas que surgen por la introducción de a_i en π_{base} son resolubles de forma consistente, se genera un nodo $\pi_{sucesor}$ sucesor de π_{base}.

3. Se aplica la función de evaluación a cada uno de los nodos $\pi_{sucesor}$ creados y estos se insertan en la lista *ABIERTOS* de acuerdo al valor de la función y se vuelve al primer paso.

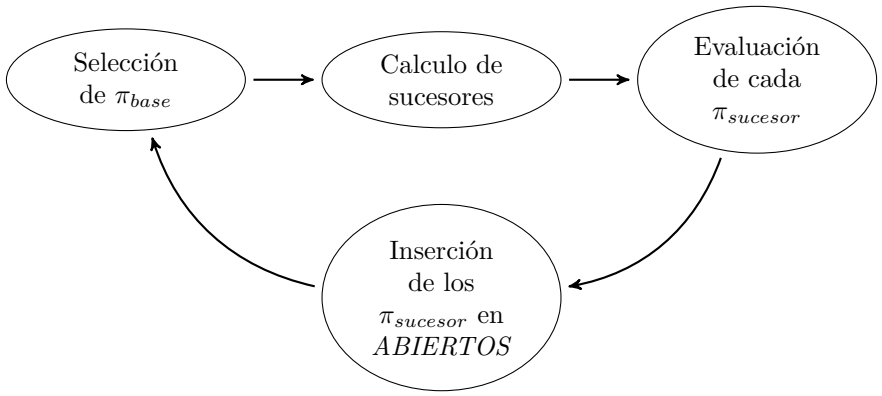

Figura 3.1: Iteración del proceso de búsqueda de FLAP

A continuación se detalla el proceso de generación de un nodo sucesor. Se utilizará el concepto de nodo candidato sucesor para indicar los diferentes pasos que se aplican a los nodos sucesores de π_{base} hasta que estos se convierten en nodos sucesores que se pueden insertar en la lista *ABIERTOS*. Los nodos candidatos sucesor se nombran como el nodo del que se parte con un superíndice que indica las modificaciones que incluye el nodo candidato sucesor, de forma

que el nodo candidato sucesor de π_{base} al que se le ha añadido la acción a_i se nombra $\pi_{base}^{a_i}$.

Sea el plan base seleccionado $\pi_{base} = \langle A_{base}, O_{base}, C_{base} \rangle$. Los pasos del proceso de generación de sucesores se detallan a continuación y se muestran gráficamente en la figura 3.2:

1. Las acciones de A_{base} soportan r de las acciones del problema $a_1...a_k...a_r \in \mathcal{A}$. Nos centramos en la acción a_k, generando un nodo candidato sucesor mediante la inserción de la acción a_k en el conjunto A_{base}. Denominaremos al nodo candidato sucesor generado $\pi_{base}^{a_k}$ de modo que $A_{base}^{a_k} = A_{base} \cup \{a_k\}$

2. Siguiendo con $\pi_{base}^{a_k}$, el siguiente paso consiste en buscar enlaces causales para las precondiciones de a_k. El objetivo es encontrar combinaciones de enlaces causales que soporten todas las precondiciones de a_k. Para cada $p_k \in pre(a_k)$ se busca el conjunto de acciones de A_{base} que soportan p_k; es decir, $\forall p_k \in pre(a_k), \exists a_j \in A_{base}, p_k \in eff(a_j)$. Se buscan las r combinaciones de enlaces causales $c_1...c_l...c_r$ que soportan las precondiciones de a_k y se genera un nodo candidato sucesor por cada una de estas combinaciones. De este modo, para una combinación de enlaces causales particular, c_l, el nodo candidato sucesor se denomina $\pi_{base}^{a_k,c_l}$, el cual también incluirá las relaciones de orden o_l que se derivan de los correspondientes enlaces causales c_l. De este modo tenemos que el conjunto de enlaces causales del nodo $\pi_{base}^{a_k,c_l}$ será $C_{base}^{a_k,c_l} = C_{base} \cup \{c_l\}$ y su conjunto de ordenaciones $O_{base}^{a_k,c_l} = O_{base} \cup \{o_l\}$.

3. En el siguiente paso se escoge un nodo candidato sucesor de los generados en el paso anterior y se resuelven las amenazas que hayan surgido añadiendo las relaciones de orden necesarias. Sea $\pi_{base}^{a_k,c_l}$ el nodo candidato sucesor; asumiendo que existen t combinaciones de relaciones de orden $o_1...o_m...o_t$ que resuelven las amenazas, se generarán t nodos candidatos sucesor, uno por cada posible combinación. Centrándonos en una combinación particular, o_m, se genera $\pi_{base}^{a_k,c_l,o_m}$ cuyo conjunto de relaciones de orden se define como $O_{base}^{a_k,c_l,o_m} = O_{base}^{a_k,c_l} \cup \{o_m\}$.

4. Llegados a este punto se dispone de todos los nodos $\pi_{sucesor}$ que se han generado para todas las acciones aplicables en π_{base} y para cada combinación de enlaces causales y resolución de amenazas de estas acciones aplicables. A continuación se aplica la función de evaluación en cada nodo $\pi_{sucesor}$ para lo cual es necesario convertir el plan contenido en el nodo en un estado, el *estado frontera*, mediante el proceso descrito posteriormente en 3.3.1.

El proceso de búsqueda de FLAP se implementa mediante un algoritmo de búsqueda tradicional. En cada iteración del algoritmo se escoge heurísticamente un nodo a expandir π_{base} de la lista *ABIERTOS*. El nodo π_{base} se expande generando todos sus nodos sucesores inmediatos libres de conflictos. A continuación se escoge un nodo de la lista *ABIERTOS* y se realiza una nueva iteración.

En resumen, se genera un total de u nodos $\pi_{sucesor}$ sucesores de π_{base}, uno por cada combinación de acción, enlaces causales y resolución de amenazas con

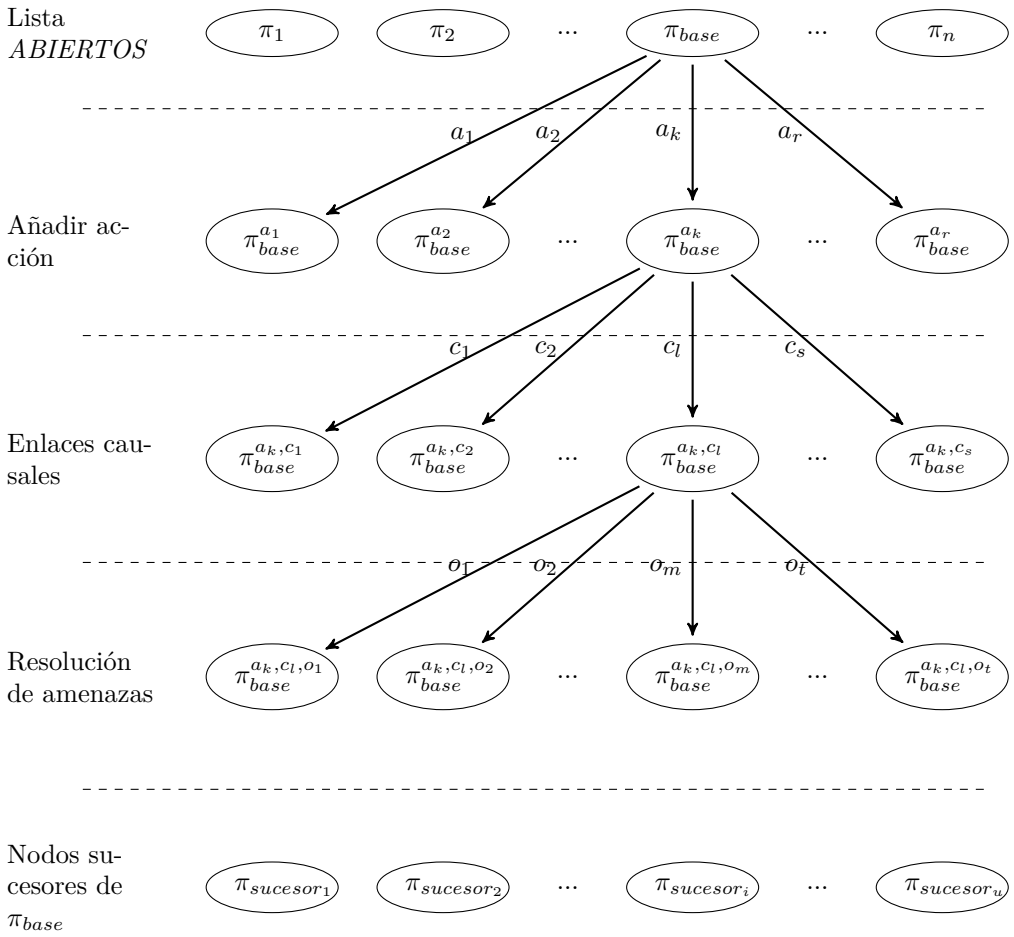

Figura 3.2: Generación de sucesores en FLAP

ordenaciones consistentes. Para cada nodo sucesor $\pi_{sucesor_i}$ se calcula su *estado frontera* y se aplica la función de evaluación, tras lo cual se inserta $\pi_{sucesor_i}$ en la lista de nodos abiertos.

La utilización de una aproximación de orden parcial permite realizar una búsqueda más flexible y alcanzar planes solución de mayor calidad que una búsqueda en un espacio de estados, tal como se ha comentado en la sección 2.2.1. Además, dado que los nodos del árbol de búsqueda de FLAP son planes consistentes (todas las precondiciones de las acciones están resueltas y el plan está libre de amenazas), FLAP puede calcular el *estado frontera* de cada nodo parcial y así aplicar heurísticas basadas en estados que son más informativas que las heurísticas POP.

3.2. Planificador temporal TFLAP

En esta sección se presenta el diseño y funcionamiento de TFLAP. Primeramente se detalla la generación del árbol de búsqueda (3.2.1), el manejo de enlaces causales (3.2.2) y la resolución de amenazas (3.2.3). Posteriormente se comenta la problemática de la consistencia temporal en la planificación temporal.

Como ya se comentó en la sección 2.3, en planificación temporal las acciones tienen asociado un intervalo de duración. A efectos prácticos, esto significa que tanto los enlaces causales como las restricciones de orden se establecen ahora entre los puntos de inicio y final de cada acción. Al utilizar como punto de partida un planificador de orden parcial como FLAP, que ya incorpora todos los mecanismos necesarios para la gestión de la causalidad y restricciones de orden, el cambio principal consiste en aplicar dicho razonamiento a los puntos de inicio y final de las acciones, teniendo en cuenta las restricciones que se establecen entre dichos puntos y las duraciones de las acciones.

Para convertir una acción instantánea en una acción durativa o con duración se introduce el concepto de **punto de tiempo de una acción**. Siguiendo la nomenclatura introducida en 2.3, una acción durativa a se define mediante dos puntos de tiempo, el punto de inicio y el punto final, que nombraremos como a^{ini} y a^{fin} respectivamente y cuya separación temporal será a^{dur}. Ambos puntos se equiparan a la entidad que utiliza FLAP para establecer relaciones de orden, enlaces causales, precondiciones y efectos. De esta forma las relaciones de orden y enlaces causales se refieren a uno de los dos puntos de tiempo de una acción, nunca a la acción globalmente. Igualmente, el procesado de precondiciones y efectos trabaja con los puntos de tiempo de una acción. Además existen pares de puntos especiales, como los que definen la duración o intervalo de ejecución de la acción $\langle a^{ini}, a^{fin} \rangle$, los cuales tienen una serie de consideraciones especiales. Adicionalmente, para generar un plan temporal a cada punto de tiempo p se le asociará un valor numérico (su instanciación temporal): el instante de tiempo en el que se ejecutará el punto de tiempo p dentro del plan y que nombraremos como $t\,(p)$.

- Se establece la relación de orden trivial $a^{ini} \prec a^{fin}$ en todas las acciones.
- Las precondiciones *over all* de la acción a se deben cumplir durante todo el intervalo de tiempo definido por $[a^{ini} \ldots a^{fin}]$.
- La distancia temporal entre a^{ini} y a^{fin} ($[a^{ini} \ldots a^{fin}]$) se corresponde con la duración de la acción a^{dur} que viene determinada por el problema de forma que se establece la relación temporal $t\left(a^{fin}\right) = t\left(a^{ini}\right) + a^{dur}$. Esto tiene implicaciones en el cálculo del instante del tiempo en el que empieza y termina cada acción del plan como se describe posteriormente en 3.2.4.

Al igual que FLAP, en TFLAP se realiza una búsqueda hacia adelante utilizando acciones totalmente instanciadas, por lo que no existen relaciones de ligadura y $B = \emptyset$ en todos los nodos del árbol de búsqueda. En cambio, TFLAP permite que los planes parciales tengan tareas pendientes de resolver debido a las precondiciones *at end*, como se describe en 3.2.1, por lo que puede ocurrir que $F \neq \emptyset$. De esta forma los planes parciales se representan mediante la tupla $\pi = \langle A, O, C, F \rangle$.

TFLAP comienza analizando el dominio y el problema realizando un proceso de *grounding* y extrayendo las acciones ficticias correspondientes al estado inicial (a_0) y al estado objetivo (a_n). En el proceso de *grounding* adicionalmente se analiza si el problema tiene solución y si para alcanzar la solución es necesario dejar precondiciones *at end* abiertas durante la planificación. En este análisis inicial, la información que hay que almacenar de las acciones es la siguiente:

- La duración de la acción.
- El tipo de cada precondición de la acción: *at start*, *over all* o *at end*.
- El tipo de cada efecto de la acción: *at start* o *at end*.
- El fluent asociado a cada precondición y a cada efecto de la acción.

3.2.1. Generación del árbol de búsqueda

El proceso de búsqueda de TFLAP sigue el mismo esquema general representado en la figura 3.1 de la sección 3.1.2. Adicionalmente, las precondiciones *at end* pueden requerir un procesado especial. Cuando se añade una acción a al plan parcial π_{base} se deben soportar sus precondiciones *at start* y *over all*, pues son condiciones que deben satisfacerse al inicio y durante la ejecución de a, respectivamente, con acciones del plan π_{base}. Si se ha detectado en el proceso de *grounding* que el problema se puede resolver sin dejar precondiciones *at end* abiertas también se exige que las precondiciones *at end* de a se soporten con las acciones de π_{base}.

Por otro lado, si se ha detectado que es necesario dejar precondiciones *at end* abiertas para resolver el problema, no se exige el soporte de estas precondiciones cuando se inserta la acción a en π_{base} pues estas precondiciones no son imprescindibles para que comience la ejecución de la acción y se pueden satisfacer posteriormente mediante la inserción de nuevas acciones. De esta forma, los nodos del árbol TFLAP pueden contener una lista de acciones con precondiciones *at end* no resueltas, es decir, $F \neq \emptyset$.

Partiendo de un nodo π_{base} a expandir se realiza el proceso de generación de nodos sucesores. Este proceso consta de dos operaciones: añadir una nueva acción o resolver las precondiciones *at end* de una acción presente en π_{base} en caso de que el nodo contenga acciones con precondiciones *at end* no resueltas.

Añadir una nueva acción es similar a FLAP, se selecciona el plan base π_{base} y se generan los nodos $\pi_{sucesor}$ para todas las acciones aplicables en π_{base} y para cada combinación de enlaces causales y resolución de amenazas de estas acciones que formen un conjunto consistente de relaciones de orden. Se destacan las siguientes particularidades de TFLAP:

- Se añaden dos relaciones de orden: $a_k^{ini} \prec a_k^{fin}$, indicando que el punto final de la acción a insertar, a_k, va siempre después de su punto de inicio; y la relación $a_0 \prec a_k^{ini}$, que indica que la acción a_k se sitúa después del inicio del plan. En el nodo candidato sucesor $\pi_{base}^{a_k}$ se añaden, por tanto, las siguientes relaciones: $O_{base}^{a_k} = O_{base} \cup \left\{ a_0 \prec a_k^{ini}, a_k^{ini} \prec a_k^{fin} \right\}$

- Se resuelven las precondiciones *at start* y *over all* de a_k. También se resuelven las precondiciones *at end* si el problema se puede resolver sin dejar precondiciones *at end* abiertas. Si la acción tiene precondiciones *at end* y para resolver el problema es necesario dejar las precondiciones *at end* abiertas, se anota que la acción no está totalmente resuelta en $F_{base}^{a_k}$.
- El análisis para decidir si se añade la acción ficticia final a_g, lo cual indica que se ha alcanzado la solución, solo se realizará si todas las acciones de A_{base} tienen sus precondiciones *at end* resueltas, es decir se exige $F_{base} = \emptyset$ para determinar si se añade la acción ficticia final a_g a π_{base}.
- Una vez generado $\pi_{sucesor}$, en la fase de evaluación se comprueba si es consistente temporalmente como se indica más adelante en 3.2.4. En caso de no ser consistente, se descarta al tratarse de un nodo no válido y no se añade a la lista *ABIERTOS*.

La otra operación consiste en resolver las precondiciones *at end* pendientes de las acciones de π_{base}. Si una acción a_e de π_{base} contiene precondiciones *at end* sin resolver y existe alguna acción de A_{base} que soportaría dicha precondición entonces se procede a su resolución. Se genera entonces un nodo $\pi_{sucesor}$ por cada combinación de enlaces causales y relaciones de orden que permiten resolver las precondiciones *at end* de las acciones de A_{base}. En el nodo $\pi_{sucesor}$ en el cual se han resuelto todas las precondiciones *at end* de a_e se elimina a_e de $F_{sucesor}$ indicando que no quedan tareas pendiente relacionadas con la acción $a_e \in A_{sucesor}$ en $\pi_{sucesor}$.

3.2.2. Enlaces causales

Cuando se añaden enlaces causales en TFLAP se genera un nodo candidato sucesor de π_{base} por cada forma de soportar las precondiciones de la acción a_k con los efectos de las acciones de A_{base}.

Sea a_k la acción a procesar, la cual tiene una precondición que se soporta con un efecto de la acción $a_j \in A_{base}$. Dependiendo del tipo de precondición de la acción a_k y efecto de la acción a_j se tienen los enlaces causales de la tabla 3.1.

$pre(a_k)$	Efecto *at start*	Efecto *at end*
at start	$a_j^{ini} \xrightarrow{f} a_k^{ini}$	$a_j^{fin} \xrightarrow{f} a_k^{ini}$
at end	$a_j^{ini} \xrightarrow{f} a_k^{fin}$	$a_j^{fin} \xrightarrow{f} a_k^{fin}$
over all	$a_j^{ini} \xrightarrow{f} a_k^{ini}$ $a_j^{ini} \xrightarrow{f} a_k^{fin}$	$a_j^{fin} \xrightarrow{f} a_k^{ini}$ $a_j^{fin} \xrightarrow{f} a_k^{fin}$

Cuadro 3.1: Enlaces causales en TFLAP

Como se ha comentado previamente, si a_k es una nueva acción, solo se analizan sus precondiciones *at start* y *over all* dejando las *at end* como pendientes

de resolver. Por el contrario, si $a_k \in A_{base}$ solo se procesan las precondiciones *at end* de a_k.

En el caso de una precondición *over all* de a_k hay que proteger el fluent durante toda la duración de a_k. Esto se puede lograr mediante el mecanismo ya existente de enlaces causales. Consideremos la siguiente situación genérica:

- Una precondición *over all*: la acción a_k requiere el fluent f, `(over all (f))`.
- El fluent f es un efecto de a_j^{fin}.
- Existe una acción a_i tal que f' es un efecto *at end* de a_i que asigna un valor distinto a la variable estado de f.

De esta forma, hay que ordenar adecuadamente los puntos a_i^{fin} respecto a_k y a_j^{fin}, para que se cumpla la precondición *over all* de a_k. Por lo tanto el objetivo es establecer las restricciones necesarias para que el fluent f esté protegido durante toda la duración de la acción a_k.

Estableciendo dos enlaces causales $a_j^{fin} \xrightarrow{f} a_k^{ini}$ y $a_j^{fin} \xrightarrow{f} a_k^{fin}$ se protege la precondición *over all* de a_k con el fluent f.

- El enlace causal $a_j^{fin} \xrightarrow{f} a_k^{ini}$ implica la introducción de la ordenación $a_j^{fin} \prec a_k^{ini}$ y protege el fluent f desde a_j^{fin} hasta a_k^{ini}. De esta forma se garantiza que el fluent f está presente en a_k^{ini} y a_i^{fin} solo se puede ordenar $a_i^{fin} \prec a_j^{fin}$ o $a_k^{ini} \prec a_i^{fin}$.
- Adicionalmente, debido al enlace causal $a_j^{fin} \xrightarrow{f} a_k^{fin}$ se extiende la protección del fluent f hasta a_k^{fin}. De esta forma a_i^{fin} solo se puede ordenar $a_i^{fin} \prec a_j^{fin}$ o $a_k^{fin} \prec a_i^{fin}$ quedando el fluent f protegido durante toda la duración de la acción a_k.

3.2.3. Resolución de amenazas

Al igual que se ha hecho con los enlaces causales, la resolución de amenazas (introducida en 2.2.1) en TFLAP se debe ajustar a los puntos inicio y final de las acciones. De esta forma en una amenaza están involucrados dos puntos temporales que definen el enlace causal amenazado y un tercer punto temporal que amenaza el enlace causal. Consideremos la siguiente situación:

- El enlace causal $a_j^{fin} \xrightarrow{f} a_k^{ini}$ donde $f = \langle v, d \rangle$.
- Existe una acción a_i tal que f' es un efecto que se genera en a_i^{fin} el cual asigna un valor distinto a la variable estado de f: $f' = \langle v, d' \rangle \in eff(a_i^{fin})$ y $d \neq d'$.

En este caso decimos que el punto a_i^{fin} amenaza al enlace causal $a_j^{fin} \xrightarrow{f} a_k^{ini}$ ya que a_i^{fin} puede situarse entre a_j^{fin} y a_k^{ini}. Dado que el enlace causal $a_j^{fin} \xrightarrow{f} a_k^{ini}$ implica la ordenación $a_j^{fin} \prec a_k^{ini}$, existen dos formas de resolver la amenaza:

1. Resolución de la amenaza por democión: $a_i^{fin} \prec a_j^{fin}$.

2. Resolución de la amenaza por promoción: $a_k^{ini} \prec a_i^{fin}$.

Al igual que en FLAP, en TFLAP una amenaza se resuelve generando un nodo candidato sucesor por cada combinación de ordenaciones que resuelva la amenaza y que resulte, junto a las ordenaciones ya existentes, un conjunto de ordenaciones consistente.

3.2.4. Instanciación y consistencia temporal

En este apartado se explica el proceso para comprobar la consistencia temporal de un nodo candidato sucesor π_j. Tras la resolución de amenazas se tiene un plan parcial π_j libre de amenazas y con un conjunto de ordenaciones O_j consistente. Se denomina consistencia temporal al proceso de comprobar la satisfacción de la restricción temporal $t\left(a^{fin}\right) = t\left(a^{ini}\right) + a^{dur}, \forall a \in A_j$. Para comprobar la consistencia temporal de π_j se realiza el proceso de instanciación temporal de los puntos de tiempo inicio y final de todas las acciones de A_j. La instanciación temporal de un punto de tiempo p consiste en asociar un valor temporal numérico $t\left(p\right)$. En el caso de que sea imposible cumplir la condición de consistencia temporal para alguna acción $a \in A_j$, el nodo candidato sucesor π_j se descartará por ser temporalmente inconsistente.

Como datos de entrada al algoritmo de instanciación temporal aplicado al plan π_j se utiliza: el conjunto de acciones A_j, las ordenaciones entre puntos de acciones O_j y la duración de cada acción $a^{dur}, \forall a \in A_j$. El algoritmo de instanciación temporal se describe en la figura 3.3 y se compone de tres fases: una fase de inicialización, procesado siguiendo el orden topológico de los puntos temporales según O_j y un procesado mediante una cola. El algoritmo utiliza una cola que sirve para almacenar los puntos pendientes de analizar, sobre dicha cola se aplican las funciones: *push* para añadir un elemento al final y *pop* para extraer el elemento del principio. Adicionalmente se utilizan las siguientes funciones:

- *extraerOrdenTopologico* para obtener el orden topológico
- *obtenerAccion* proporciona la acción a la que corresponde un punto temporal
- *esInicio* indica si un punto temporal corresponde al inicio de una acción
- *buscar* proporciona un conjunto de puntos que satisfacen cierta condición

El procesado por orden topológico es una linealización de un grafo de forma que se preservan las ordenaciones entre nodos, esto se consigue mediante una búsqueda recursiva de los nodos adyacentes a cada nodo. El resultado es el orden en que se deben recorrer los nodos de forma que siguiendo el orden topológico un nodo se debe procesar una única vez. Este procesado se podría omitir inicializando la cola con a_0 y el resultado sería el mismo pero con un coste mayor de procesado. Si un mismo punto se analiza demasiadas veces[1], se asume que estamos ante un bucle infinito de actualizaciones, por lo que π_j se considera

[1] Tantas veces como el número de puntos temporales existentes en el plan.

```
// Inicio
cola=∅
foreach a in A_j {
  t(a^ini) = −∞
  t(a^fin) = −∞
}
t(a_0) = 0

// Procesado por orden topologico
foreach p_a in extraerOrdenTopologico(O_j) {
  if p_a ≠ a_0 {
    a = obtenerAccion (p_a)
    if esInicio (p_a) {
      if t(a^fin) < t(a^ini) + a^dur {
        t(a^fin) = t(a^ini) + a^dur
      }
    } else {
      if t(a^fin) > t(a^ini) + a^dur {
        t(a^ini) = t(a^fin) − a^dur
        cola.push (a^ini)
      }
    }
  }
  foreach p_p in buscar(p_a ≺ p_p ∈ O_j) {
    if t(p_p) ≤ t(p_a) {
      t(p_p) = t(p_a) + ϵ
    }
  }
}

// Procesado mediante cola
while cola ≠ ∅ {
  p_a=cola.pop
  a = obtenerAccion (p_a)
  if esInicio (p_a) {
    if t(a^fin) < t(a^ini) + a^dur {
      t(a^fin) = t(a^ini) + a^dur
      cola.push (a^fin)
    }
  } else {
    if t(a^fin) > t(a^ini) + a^dur {
      t(a^ini) = t(a^fin) − a^dur
      cola.push (a^ini)
    }
  }
  foreach p_p in buscar(p_a ≺ p_p ∈ O_j) {
    if t(p_p) ≤ t(p_a) {
      t(p_p) = t(p_a) + ϵ
      cola.push (p_p)
    }
  }
}
```

Figura 3.3: Código del procedimiento de instanciación temporal

inconsistente temporalmente. Adicionalmente la instanciación temporal permite calcular la duración del plan del nodo candidato sucesor π_j (*makespan*) que coincide con el mayor tiempo de los puntos de las acciones de A_j.

A continuación se realiza un análisis algebraico de un ejemplo sencillo de relaciones entre acciones para mostrar un caso simple de inconsistencia temporal. Sea un nodo π_j con dos acciones, además de la acción ficticia inicial: $A_j = \{a_0, a_1, a_2\}$. Consideremos una duración genérica de las acciones: a_1^{dur} y a_2^{dur}. Estas acciones están ordenadas de forma que la acción a_2 empieza y termina durante la ejecución de a_1 tal como se muestra en la figura 3.4. Es decir, según el conjunto de ordenaciones de la ecuación 3.1.

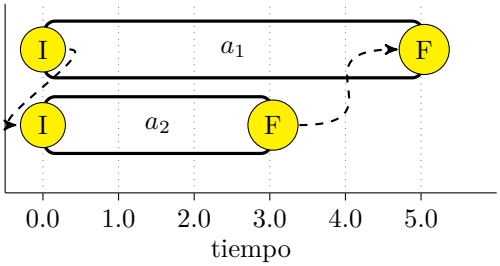

Figura 3.4: Consistencia temporal simple

$$O_j = \left\{ \begin{array}{ll} a_0 \prec a_1^{ini} & a_1^{ini} \prec a_1^{fin} \\ a_1^{ini} \prec a_2^{ini} & a_2^{ini} \prec a_2^{fin} \\ a_2^{fin} \prec a_1^{fin} & \end{array} \right\} \tag{3.1}$$

Se trata de un conjunto de ordenaciones consistentes pero según la duración de las acciones el nodo π_j puede ser un plan temporalmente consistente (como en el caso de la figura 3.4) o inconsistente (como en el caso de la figura 3.5). Obsérvese que las ordenaciones y las acciones de ambas figuras son idénticas, solo cambia la duración de las acciones que hace que en la figura 3.5 no se pueda satisfacer una ordenación (indicada en rojo).

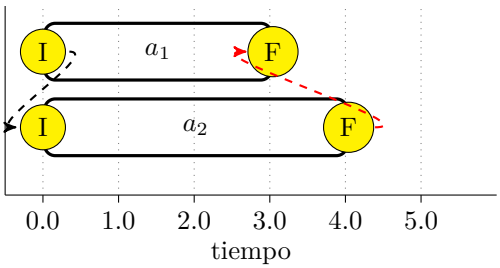

Figura 3.5: Inconsistencia temporal simple

Para el análisis algebraico, se parte de las ordenaciones de O_j y del tiempo de la acción ficticia $t\,(a_0) = 0$. Dado que el objetivo es encontrar una relación entre la duración de las acciones que garantice la consistencia temporal el plan, la duración de las acciones se deja como a_1^{dur} y a_2^{dur}. De esta forma se pueden asignar el resto de tiempos de forma algebraica:

- $a_0 \prec a_1^{ini} \rightarrow t\left(a_1^{ini}\right) \geq t\left(a_0\right) + \epsilon$
- $a_1^{ini} \prec a_2^{ini} \rightarrow t\left(a_2^{ini}\right) \geq t\left(a_1^{ini}\right) + \epsilon$
- $a_2^{fin} \prec a_1^{fin} \rightarrow t\left(a_1^{fin}\right) \geq t\left(a_2^{fin}\right) + \epsilon$
- La acción a_1 dura a_1^{dur}, por lo que $t\left(a_1^{fin}\right) = t\left(a_1^{ini}\right) + a_1^{dur}$
- La acción a_2 dura a_2^{dur}, por lo que $t\left(a_2^{fin}\right) = t\left(a_2^{ini}\right) + a_2^{dur}$

Se trata de un sistema de inecuaciones que se puede simplificar asignando valores a las variables partiendo de $t\left(a_0\right) = 0$ tal como muestra la ecuación 3.2.

$$\left.\begin{array}{l} t\left(a_0\right) = 0 \\ t\left(a_1^{ini}\right) \geq t\left(a_0\right) + \epsilon \\ t\left(a_2^{ini}\right) \geq t\left(a_1^{ini}\right) + \epsilon \\ t\left(a_1^{fin}\right) \geq t\left(a_2^{fin}\right) + \epsilon \\ t\left(a_1^{fin}\right) = t\left(a_1^{ini}\right) + a_1^{dur} \\ t\left(a_2^{fin}\right) = t\left(a_2^{ini}\right) + a_2^{dur} \end{array}\right\} \left.\begin{array}{l} t\left(a_1^{ini}\right) = \epsilon \\ t\left(a_2^{ini}\right) = 2\epsilon \\ t\left(a_1^{fin}\right) \geq t\left(a_2^{fin}\right) + \epsilon \\ t\left(a_1^{fin}\right) = t\left(a_1^{ini}\right) + a_1^{dur} \\ t\left(a_2^{fin}\right) = t\left(a_2^{ini}\right) + a_2^{dur} \end{array}\right\} \tag{3.2}$$

$$\left.\begin{array}{l} t\left(a_1^{fin}\right) \geq t\left(a_2^{fin}\right) + \epsilon \\ t\left(a_1^{fin}\right) = \epsilon + a_1^{dur} \\ t\left(a_2^{fin}\right) = 2\epsilon + a_2^{dur} \end{array}\right\} a_1^{dur} \geq 2\epsilon + a_2^{dur}$$

Finalmente se llega a la relación $a_1^{dur} \geq 2\epsilon + a_2^{dur}$ que es la condición que se debe satisfacer para que exista consistencia temporal en π_j.

3.3. Algoritmo de búsqueda en TFLAP

En esta sección se describen los diferentes componentes del proceso de búsqueda de TFLAP. Se empieza describiendo el *estado frontera*, seguido de la problemática de los nodos repetidos en las búsquedas hacia adelante. A continuación se comenta la heurística utilizada. Por último se describe el lanzamiento de búsquedas en paralelo y la problemática de los *plateau*.

3.3.1. Cálculo del *estado frontera*

El *estado frontera* de un plan π es el estado resultante tras la ejecución de las acciones del plan. Conceptualmente consiste en simular la ejecución de π generando los diferentes estados por los que atraviesa la ejecución del plan. De forma genérica, el cálculo del *estado frontera* de π se realiza partiendo de los fluents del estado inicial y aplicando los efectos de todas las acciones de A (solo las acciones reales, sin incluir la acción ficticia a_0) siguiendo el orden indicado en O.

En TFLAP se adapta el cálculo genérico del *estado frontera* a la estructura de las acciones durativas, utilizando los puntos de inicio y final de las acciones de A siguiendo el orden indicado en O. De esta forma, cuando se procesa a^{ini} se aplican los efectos *at start* de la acción a. Mientras que los efectos *at end* de la acción a se aplicarán al procesar a^{fin}.

3.3.2. Nodos repetidos: memorización

Uno de los inconvenientes de la búsqueda hacia adelante es la generación de múltiples nodos repetidos. Se trata de un problema que aparece en la mayoría de búsquedas hacia adelante y que TFLAP evita usando una técnica de memorización. La memorización permite saber si un nodo candidato sucesor ya se generó previamente en el proceso de búsqueda, en cuyo caso, se descarta antes de añadirlo a la lista *ABIERTOS*.

TFLAP genera nodos sucesores en los que se añades nuevas acciones y nodos sucesores en los que no se añaden acciones (cuando resuelve tareas pendientes debidas precondiciones *at end*). Dado que en un plan temporal las acciones se pueden ordenar de diferentes formas obteniendo planes con *makespan* diferente, dos planes con distinto *makespan* se consideran diferentes.

Por este motivo en TFLAP se considera que dos planes π_1 y π_2 son iguales si tienen el mismo *makespan*, las mismas acciones ($A_1 = A_2$), las mismas tareas pendientes de resolver ($F_1 = F_2$) y tienen el mismo *estado frontera*. De esta forma, dado un nodo candidato sucesor π_c, si se encuentra un plan generado previamente π_p tal que $A_c = A_p$ y $F_c = F_p$ con el mismo *makespan*, TFLAP considera que $\pi_c = \pi_p$, por lo cual π_c es un nodo previamente generado y se descarta.

3.3.3. Evaluación heurística

Para elegir el nodo del árbol a expandir TFLAP utiliza una heurística de tipo A con la función estándar $f(\pi_j) = g(\pi_j) + h(\pi_j)$ [21] para determinar la posición posición del nodo sucesor π_j en la lista *ABIERTOS*. En este trabajo se ha utilizado una función heurística que estima el número de acciones restantes para alcanzar los objetivos; esto es, una heurística que se utiliza habitualmente en problemas no temporales. En cambio, en problemas temporales, el objetivo es minimizar la duración del plan solución y no el número de acciones. No obstante, utilizar el número de acciones como criterio a minimizar puede dar resultados globalmente buenos en planificación temporal. La adaptación de la heurística a un contexto temporal o el desarrollo de heurísticas temporales propias se deja para un futuro desarrollo.

Respecto a la función heurística utilizada hay que destacar que en vez de usar la función estándar de una heurística A, se le da mayor peso al término $h(\pi_j)$. Ese mayor peso aumenta la voracidad del algoritmo por lo que está más orientado a encontrar una solución en poco tiempo que a encontrar la mejor solución. Concretamente, se usa un peso de 5, por lo que queda $f(\pi_j) = g(\pi_j) + 5 * h(\pi_j)$.

Gracias a la aplicación de una búsqueda hacia delante y a la utilización del concepto de *estado frontera*, TFLAP puede emplear heurísticas basadas en estados que son mucho más informativas que las heurísticas clásicas para POP.

- $g(\pi_j)$ es el coste (número de acciones) del plan de orden parcial π_j.
- $h(\pi_j)$ es el número estimado de acciones para llegar a los objetivos desde el *estado frontera* de π_j.

TFLAP puede utilizar diferentes funciones heurísticas para calcular el componente $h(\pi_j)$:

- Heurística h_{DTG} basada en DTGs (*Domain Transition Graph*). Un grafo DTG es un grafo en el que los nodos representan los valores de las variables estado y los arcos representan las transiciones entre los distintos valores de las variables [13]. Cada transición de una variable estado se etiqueta con las condiciones necesarias para que se produzca, como las precondiciones de la acción que genera la transición. La heurística h_{DTG} devuelve el numero de acciones en un plan relajado que permite alcanzar el estado objetivo $\mathcal{G}$ partiendo del *estado frontera* de π_j.

- Otra heurística utilizada en TFLAP es h_{FF}, la heurística del planificador FF. h_{FF}, al igual que h_{DTG}, construye un plan relajado y devuelve el número de acciones del plan relajado pero a diferencia de h_{DTG} el plan relajado se calcula a partir de un Grafo de Planificación Relajado y no de un DTG [14].

- La heurística $h_{LAND}(\pi_j)$ devuelve el número de landmarks que quedan por alcanzar en π_j. Un landmark es un fluent que ha de cumplirse en algún punto de todo plan solución [15, 24]. Hay problemas que por su naturaleza contienen un elevado número de landmarks, pero en otros problemas no se consiguen extraer landmarks. En el análisis inicial del problema se calcula un grafo de landmarks y posteriormente se utiliza esta información para el cálculo de la estimación heurística: como todos los landmarks se deben alcanzar, la distancia a $\mathcal{G}$ se puede estimar mediante los landmarks que todavía no se han conseguido.

TFLAP utiliza una combinación de dos funciones heurísticas al igual que otros planificadores del estado del arte como *FMAP* [25, 26]. Se seleccionan dos funciones heurísticas de modo que los nodos del árbol de búsqueda se ordenan por cada una de las funciones y se selecciona alternativamente de una cola o de otra el nodo a expandir.

La selección de las dos funciones heurísticas en TFLAP se realiza del siguiente modo:

- Si se han podido extraer landmarks del problema se usa h_{FF} en una cola y h_{LAND} en la otra.

- Si el problema no contiene landmarks entonces se usa h_{FF} en una cola y h_{DTG} en la otra.

3.3.4. Procesos de búsqueda paralela

Durante el proceso de búsqueda es posible llegar a un *plateau*. Se trata de una región del espacio de búsqueda donde todos los nodos tienen el mismo valor de la función de evaluación. De este modo el proceso de búsqueda es incapaz de discriminar entre los diferentes nodos y carece de una dirección clara por donde expandir. Es decir, un *plateau* se produce cuando los valores heurísticos no son informativos.

TFLAP considera que se ha llegado a un *plateau* cuando la heurística h_{FF} no ha mejorado en los últimos 50 planes expandidos. Para salir de esta situación, una vez detectada, se lanzan nuevas búsquedas en paralelo.

TFLAP selecciona un nodo de partida π_{origen} y lanza la nueva búsqueda usando el *estado frontera* de π_{origen} como estado inicial de la nueva búsqueda. Esto significa que la nueva búsqueda es un nuevo problema.

Cuando la búsqueda paralela consigue encontrar la solución, devuelve el plan que ha encontrado $\pi_{paralelo}$. En este momento hay que aplicar los resultados de la búsqueda paralela a la búsqueda inicial. Esto significa concatenar π_{origen} con $\pi_{paralelo}$. En el caso de que la búsqueda paralela encuentre un nuevo *plateau*, lanzará una nueva búsqueda paralela.

La forma trivial de aplicar los resultados de la búsqueda paralela consiste en añadir el plan de la búsqueda paralela $\pi_{paralelo}$ al final de π_{origen} de forma que $a_i^{fin} \prec a_j^{ini} \forall a_i \in A_{origen}, a_j \in A_{paralelo}$. Esta aproximación devuelve un resultado subóptimo pues puede haber acciones de $A_{paralelo}$ que podrían ejecutarse antes de que finalizaran todas las acciones de A_{origen}.

Para mejorar el plan resultado de concatenar los planes π_{origen} y $\pi_{paralelo}$, TFLAP utiliza el algoritmo de la figura 3.6 La idea del algoritmo es expandir el nodo π_{origen} añadiendo las acciones de $\pi_{paralelo}$ (en el mismo orden en que se añadieron para llegar a $\pi_{paralelo}$) mediante el proceso de generación de árbol de búsqueda de TFLAP, lo que permite reordenar las acciones de forma óptima. El algoritmo parte de los dos planes a concatenar (π_{origen} y $\pi_{paralelo}$) y que se compone de las siguientes fases:

1. Extracción de las acciones de $\pi_{paralelo}$ en el mismo orden en que se insertaron para la generación del plan $\pi_{paralelo}$ partiendo del *estado frontera* de π_{origen}. Como en $\pi_{paralelo}$ se ha añadido la última acción y en cada uno de sus predecesores se añade una única acción por plan, se recorren los planes desde $\pi_{paralelo}$ siguiendo la línea de planes predecesores y se añade la acción de cada plan parcial a una pila de forma que en la parte superior de la pila quedará la primer acción que se añadió en el proceso de búsqueda paralelo. Este orden garantiza que las precondiciones de cada acción se podrán cumplir pues se ha extraído el orden en que se han añadido las acciones en el proceso de búsqueda paralelo.

2. Generación de los nodos sucesores utilizando de forma consecutiva cada una de las acciones de $\pi_{paralelo}$ partiendo de π_{origen}. En esta fase, se extraen las acciones de la cola, por lo que se empieza por la primer acción se añadió en el proceso de búsqueda paralelo y se termina con la última.

3. En este punto se han generado todas las concatenaciones posibles. El algoritmo finaliza insertando todos los sucesores generados a la lista *ABIERTOS*.

El algoritmo de 3.6 se basa en el uso de diferentes estructuras de datos: listas en las que el orden de los elementos no es relevante y una pila que se debe recorrer en el orden indicado para el correcto funcionamiento del algoritmo.

- Una lista de nodos pendientes de expandir: *expandir*.

```
// Extraer acciones de π_paralelo
acciones=∅
π_actual = π_paralelo
while π_actual ≠null {
  acciones.pushBack (accionNodo (π_actual))
  π_actual = predecesor (π_actual)
}

// Generar nuevos nodos
expandir=π_origen
while acciones≠ ∅ {
  a=acciones.popBack
  sucesores=∅
  while expandir≠ ∅ {
    π_e=expandir.pop
    sucesores.push (expande (π_e, a))
  }
  expandir=sucesores
}

// Finalizar
ABIERTOS.pushHeur (expandir)
```

Figura 3.6: Pseudo-código de concatenación de planes

- Una lista de nodos sucesores generados: *sucesores*.
- Una pila de acciones de $\pi_{paralelo}$ pendientes de añadir a los planes: *acciones*.

El algoritmo de 3.6 utiliza diferentes funciones, tanto para el manejo de las estructuras de datos como para realizar diferentes operaciones relacionadas con la búsqueda:

- *pushBack* para añadir al final de una estructura de datos, se le puede pasar un elemento o $\emptyset$. En el caso de recibir como parámetro $\emptyset$, la función no hace nada.
- *popBack* extrae el elemento final de la estructura de datos
- *push* añade, en cualquier orden, elementos ($\emptyset$, uno o varios) a la estructura de datos.
- *pop* extrae un elemento cualquiera de la estructura de datos.
- *pushHeur* añade elementos a la estructura de datos según el orden indicado por la evaluación heurística descrita en 3.3.3.
- *accionNodo* devuelve la acción que se ha insertado al generar cierto nodo, la acción del proceso descrito en 3.2.1. Es posible que devuelva $\emptyset$ si para generar ese nodo no se hubiera añadido ninguna acción.
- *predecesor* devuelve el nodo predecesor en el árbol de búsqueda.
- *expande* devuelve los nodos sucesores de un plan π_e añadiendo cierta acción. Se trata del proceso de expansión de nodos descrito en 3.2.1.

Tras la ejecución del algoritmo 3.6 se dispone de todas las concatenaciones posibles de π_{origen} con $\pi_{paralelo}$ en *ABIERTOS*, por lo que se continúa con el proceso normal de búsqueda.

3.4. Comparativa de TFLAP con OPTIC

Tanto TFLAP como OPTIC son planificadores temporales de orden parcial que realizan una búsqueda hacia adelante. Por lo cual resulta lógico comparar ambos planificadores tanto a nivel de resultados (como se hace en el capítulo 5) como a nivel conceptual. En esta sección se realiza una comparativa conceptual de TFLAP y OPTIC: primero se analiza el uso del *estado frontera* y a continuación se analiza la heurística.

3.4.1. Uso del *estado frontera*

Tanto OPTIC como TFLAP usan el *estado frontera* como punto de partida de la estimación heurística: estiman el coste de llegar del *estado frontera* de un plan parcial π_j al estado objetivo del problema $\mathcal{G}$. Adicionalmente OPTIC limita las acciones que se pueden añadir a un plan a expandir π_{base} a las acciones cuyas precondiciones están soportadas en el *estado frontera* de π_{base}. En cambio, TFLAP recolecta los efectos de todas las acciones de A_{base} para saber si dispone de todos los fluents de las precondiciones de la acción que se está analizando.

De esta forma OPTIC prueba menos acciones y requiere una menor cantidad de cálculos que TFLAP para decidir si una acción se puede añadir a π_{base}. Al probar menos acciones, la expansión de un nodo genera menos nodos sucesores, por lo que tiende a manejar menos nodos (requiere menos memoria) y la expansión de cada nodo tiende a ser más rápida. Además al añadir una acción en OPTIC no aparecerán amenazas pues basta con añadir la nueva acción al final de π_{base} para resolver todas las amenazas. Lo cual se traduce en que la generación de nodos sucesores es más sencilla (y rápida) en OPTIC que en TFLAP.

Como contrapartida, TFLAP es capaz de estudiar acciones cuyas precondiciones se soporten en el interior del plan π_{base} aunque no en su *estado frontera*. De esta forma TFLAP hace un mayor uso del principio de menor compromiso, requiere un menor uso de backtracking y fomenta el encontrar planes en los que las acciones se ejecuten en paralelo.

A efectos prácticos, esto significa que para para problemas cuya solución es altamente secuencial, OPTIC será más rápido pues básicamente solo se podrán añadir acciones cuyas precondiciones se cumplan en los estados frontera del nodo π_{base} a expandir. Sin embargo, TFLAP tendrá un mejor comportamiento ante relaciones complejas entre acciones.

Tomemos como ejemplo un problema con las siguientes características:

- En el estado inicial se tiene los fluents f_1 y f_2: $\mathcal{I} = \{f_1, f_2\}$. La acción virtual a_0 equivale a $\mathcal{I}$.
- El objetivo es tener los fluents f_4 y f_5: $\mathcal{G} = \{f_4, f_5\}$
- Las acciones de la tabla 3.2.

Acción	Precondiciones	Efectos
a_1	f_1	Convierte f_1 en f_3
a_2	f_2, f_3	Elimina f_3 y convierte f_2 en f_5
a_3	f_2	Añade f_4

Cuadro 3.2: Acciones de ejemplo: *estado frontera* TFLAP respecto OPTIC

Se tiene un nodo con el plan parcial de la figura 3.7, cuyo *estado frontera* es f_5 y que se describe formalmente como:

- $\pi = \langle A, O, C \rangle$
- $A = \{a_0, a_1, a_2\}$
- $C = \{a_0 \xrightarrow{f_1} a_1, a_0 \xrightarrow{f_2} a_2, a_1 \xrightarrow{f_3} a_2\}$
- $O = \{a_0 \prec a_1, a_0 \prec a_2, a_1 \prec a_2\}$

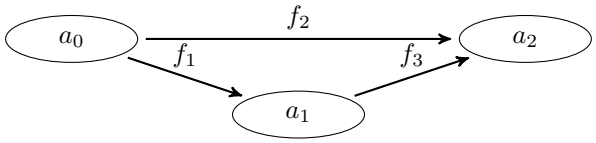

Figura 3.7: Plan parcial a expandir π

El nodo que se está estudiando no es un plan solución, pues no consigue f_4. Además, en su *estado frontera* no se puede añadir ninguna acción, por lo que OPTIC debe volver a un plan parcial previo mediante *backtracking* para explorar otra rama del árbol de búsqueda. En cambio TFLAP consigue insertar la acción a_3 generando el plan parcial π' de la figura 3.8 cuya descripción formal es:

- $\pi' = \langle A', O', C' \rangle$
- $A' = \{a_0, a_1, a_2, a_3\}$
- $C' = \{a_0 \xrightarrow{f_1} a_1, a_0 \xrightarrow{f_2} a_2, a_1 \xrightarrow{f_3} a_2, a_0 \xrightarrow{f_2} a_3\}$
- $O' = \{a_0 \prec a_1, a_0 \prec a_2, a_0 \prec a_3, a_1 \prec a_2, a_3 \prec a_2\}$

En cuyo *estado frontera* ya aparecen los fluents f_4 y f_5 por lo que será un plan solución.

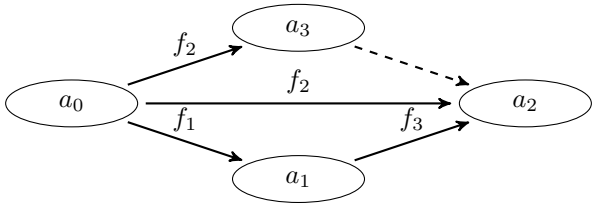

Figura 3.8: Plan parcial expandido por TFLAP π'

3.4.2. Heurística y proceso de búsqueda

La heurística de OPTIC es muy simple basada en grafos relajados, como se ha comentado en la sección 2.3.3, y una búsqueda similar a la del planificador FF. TFLAP usa una combinación de heurísticas como se indica en 3.3.3. Gracias a los puntos intermedios que proporcionan los landmarks la heurística de TFLAP tiene un mejor comportamiento que la de OPTIC para problemas cuya solución requiere una gran cantidad de acciones lo que se traduce en una mayor capacidad para resolver problemas complejos.

En cuanto al proceso de búsqueda, OPTIC usa una combinación de ascenso de colinas y búsqueda voraz. Mientras que TFLAP utiliza una búsqueda de tipo A en la que se da más peso a la estimación heurística $h(\pi)$ que al coste $g(\pi)$ por lo que tiene tendencia voraz.

Capítulo 4

Análisis de TFLAP en problemas concurrentes

En este capítulo se analiza la capacidad de TFLAP para resolver problemas con relaciones complejas, concretamente, concurrencia temporal y un prototipo de aplicación práctica. Primero se analiza la capacidad de TFLAP para resolver problemas con relaciones de concurrencia entre acciones. Se trata de problemas aparentemente sencillos (con pocas acciones y pocos literales) pero que resultan especialmente complejos para los planificadores debidos a las relaciones entre las distintas acciones.

A continuación se presentan diferentes tipos de escenarios reales en los que se puede utilizar un planificador. Por último, se presentan los resultados obtenidos para una versión sencilla de aplicación práctica: un problema de gestión de accidentes de tráfico.

4.1. Problemas básicos de concurrencia

En el trabajo [4] se describe la problemática de la concurrencia y los diferentes tipos de concurrencia temporal entre acciones. Un problema de planificación temporal es concurrente si las acciones de cualquier plan solución no se pueden secuencializar, esto es, situar una acción detrás de otra.

Entre dos acciones a_{1x} y a_{2x}, puede haber distintas relaciones que, a continuación se listan con distintos valores de x. Por una parte está la relación sencilla (relación a: a_{2a} debe comenzar después de que termine a_{1a}) que es equivalente a las relaciones que debe manejar un planificador no temporal y permite una solución secuencial. Pero en un problema de planificación temporal podemos encontrar los tipos de relaciones que denominamos concurrentes:

- Relación b: a_{2b} debe comenzar después de que empiece a_{1b}.
- Relación c: a_{2c} debe terminar después de que empiece a_{1c}.

- Relación d: a_{2d} debe terminar después de que termine a_{1d}.
- Relación e: a_{2e} debe estar dentro de a_{1e}: a_{2e} debe comenzar y acabar entre el inicio y fin de a_{1e}

A continuación se muestran dos ejemplos sencillos. Existen dos acciones a_a y a_b cuyas duraciones son $a_a^{dur} = 4$ y $a_b^{dur} = 2$. La acción a_a genera un efecto R *at start* requerido para que empiece a_b y lo niega *at end*, por lo cual b debe empezar entre a_a^{ini} y a_b^{fin}. La acción a_b genera el efecto objetivo G *at end* y a_a elimina el efecto objetivo *at end*, de esta forma a_b^{fin} debe ejecutarse después de a_a^{fin} para alcanzar el objetivo. La solución es trivial y se muestra en la figura 4.1 en la que se muestra el enlace causal con una línea continua y las ordenaciones con líneas discontinuas. Adicionalmente en la parte superior de cada acción se muestran sus precondiciones y en la parte inferior, sus efectos.

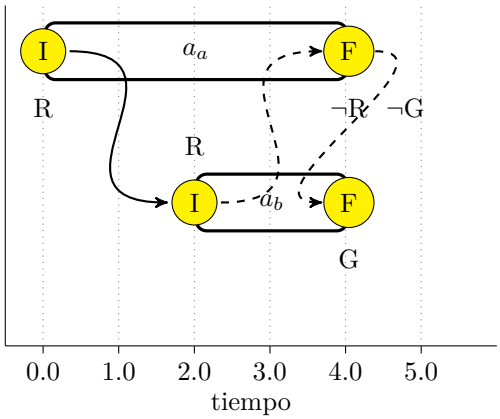

Figura 4.1: Solución del problema concurrente con precondición

Otro ejemplo similar consiste en que ambas acciones no tengan precondiciones y que a_b genere *at end* el efecto $G2$ mientras que a_a genera *at end* el efecto $G1$ y niega el efecto $G2$. En este caso el objetivo es tener $G1$ y $G2$. Ahora el plan solución es el mismo que en 4.1 pero hay un menor número de relaciones, como se muestra en la figura 4.2.

En los problemas de competición, se encuentran algunos dominios que ponen de manifiesto esta problemática. En este trabajo, en la sección 5.4, se prueban dominios con concurrencia: *Matchcellar* y *TurnAndOpen*. También existen problemas ejemplo que muestran esta problemática como los mostrados en las secciones 4.1.1 y 4.1.2.

4.1.1. Concurrencia simple entre acciones: problema *borrower*

En este primer problema con concurrencia hay una persona que necesita una hipoteca para poder comprar una casa. Para conseguir una hipoteca es necesario disponer de un plan de ahorro que se consigue mediante la acción *saveHard*.

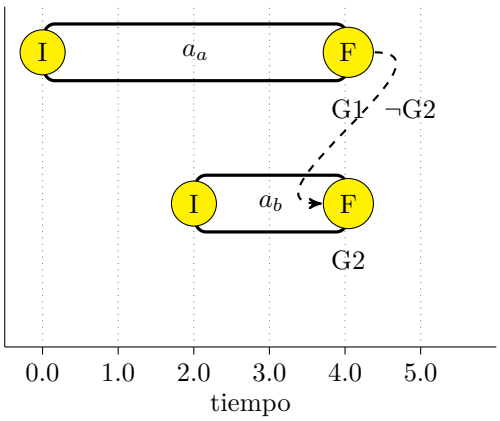

Figura 4.2: Solución del problema concurrente, solo objetivos

Asumiremos que una vez la persona comienza un plan de ahorro, está sujeta a 10 años de dicho plan de ahorro. La hipoteca solo se puede solicitar mientras la persona está aplicando su plan de ahorro. El cliente puede solicitar una hipoteca a 10 (*shortMortgage*) o 12 años (*longMortgage*). Una vez cancelada la hipoteca, el cliente será el propietario de la casa. La hipoteca está representada por la acción *takeMortgage*. Finalmente, se genera una acción para determinar cuando el cliente estará feliz, esto es, libre de deudas y siendo propietario de una casa. La acción *lifeAudit* representa que el cliente sólo será feliz si logra completar su hipoteca durante los años que mantiene el plan de ahorro. La tabla 4.1 muestra las diferentes acciones con sus precondiciones y efectos.

Acción	Precondiciones	Efectos
saveHard	*at start* canSave	Durante la acción: ¬canSave, saving
takeMortgage	*at start* saving	*at end* boughtHouse
lifeAudit	*at start* saving *at end* boughtHouse	*at end* Se consigue el objetivo

Cuadro 4.1: Precondiciones y efectos de las acciones del problema *borrower*

Analizando la tabla 4.1 se observa que las acciones *takeMortgage* y *lifeAudit* deben empezar durante la acción *saveHard*. Además, la acción *lifeAudit* debe terminar después de *takeMortgage*. Es decir, cualquier plan solución debe satisfacer (bien mediante enlaces causales, bien mediante relaciones de orden) las siguientes restricciones:

- $a^{ini}_{saveHard} \prec a^{ini}_{takeMortgage} \prec a^{fin}_{saveHard}$
- $a^{ini}_{saveHard} \prec a^{ini}_{lifeAudit} \prec a^{fin}_{saveHard}$
- $a^{fin}_{takeMortgage} \prec a^{fin}_{lifeAudit}$

La tarea del planificador es generar un plan que permita alcanzar el objetivo en el menor tiempo posible. La solución de este problema mediante TFLAP viene dada por la figura 4.3 que se representa gráficamente en la figura 4.4 en la cual aparecen las acciones, los enlaces causales (mediante líneas continuas) y las ordenaciones (mediante líneas discontinuas).

```
0.001: (saveHard) [10.000]
6.003: (lifeAudit) [4.000]
0.002: (takeMortgage shortmortgage) [10.000]
```

Figura 4.3: Solución del problema *borrower* mediante TFLAP

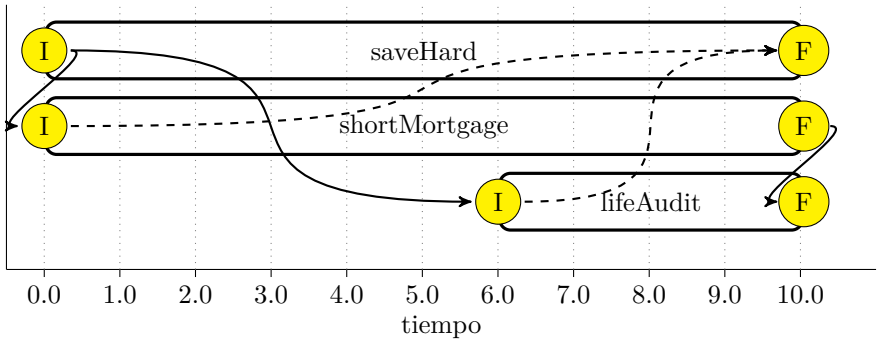

Figura 4.4: Solución del problema *borrower* mediante TFLAP

4.1.2. Acciones interrelacionadas: *interaction*

Es habitual que los planificadores solo analicen una acción a añadir a_a si se cumplen todas las precondiciones de a_a, esto provoca que cierta clase de problemas no se puedan resolver. Se trata de problemas con grupos de acciones interrelacionadas de forma que existe una dependencia circular entre acciones. El ejemplo más simple aparece con dos acciones:

- La acción a_1 tiene como precondición *at end* f_2 y genera *at start* f_1.
- La acción a_2 tiene como precondición *at start* f_1 y genera *at start* f_2.
- La ordenación $a_1^{ini} \prec a_2^{ini} \prec a_1^{fin}$ hace que se cumplan las precondiciones: f_1 en a_2^{ini} y f_2 en a_1^{fin}.
- Pero si no se añade a_1 mientras su precondición sobre f_1 no se cumpla, no se dispondrá del fluent f_2. Y si tampoco se añade a_2 mientras su precondición sobre f_2 no se cumpla, no se dispondrá del fluent f_1. De esta forma no se podría añadir al plan ni a_1 ni a_2

A modo de ejemplo se presenta un problema con cuatro acciones genéricas con ciertas dependencias entre ellas. Las relaciones entre acciones son:

- La acción a_a genera el efecto P en a_a^{ini} y lo niega en a_a^{fin}. Y el efecto P se requiere en a_b^{ini} y a_c^{fin}.
- La acción a_a genera el efecto Q en a_a^{fin} que se requiere en a_d^{ini}.
- La acción a_b genera el efecto R en a_b^{fin} que se requiere en a_c^{ini}.
- La acción a_c genera el efecto S en a_c^{fin} que se requiere en a_d^{ini}.
- La acción a_d genera el efecto T en a_d^{fin} que se requiere en a_b^{fin}.
- Los objetivos se consiguen en a_d^{fin}.

Como se observa se trata de un problema con relativamente pocas acciones que están relacionadas entre ellas. La solución a este problema se puede obtener manualmente y consiste en la ordenación de la ecuación 4.1.

$$A_{inicio} \prec B_{inicio} \prec C_{inicio} \prec C_{fin} \prec A_{fin} \prec D_{inicio} \prec D_{fin} \prec B_{fin} \qquad (4.1)$$

La figura 4.5 presenta el resultado gráficamente, con el objetivo de facilitar el seguimiento de la gráfica se ha seleccionando un color diferente para representar cada fluent (el fluent, los enlaces causales y las ordenaciones). Adicionalmente en la parte superior de cada acción se muestran sus precondiciones y en la parte inferior, sus efectos.

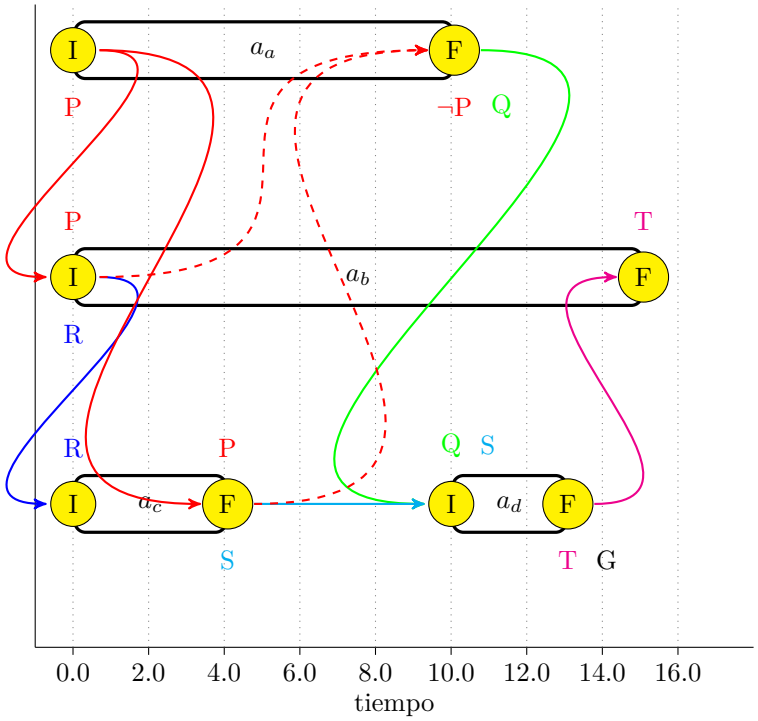

Figura 4.5: Limitación con relaciones

Se observa una serie de enlaces causales que limitan el orden en que se pueden añadir las acciones si se requiere que todas las precondiciones de una acción se cumplan antes de añadir la acción al plan:

- $a_d^{fin} \xrightarrow{T} a_b^{fin}$ hace que no se pueda añadir a_b a no ser que $a_d \in A_{base}$.
- $a_c^{fin} \xrightarrow{S} a_d^{ini}$ hace que no se pueda añadir a_d a no ser que $a_c \in A_{base}$.
- $a_b^{ini} \xrightarrow{R} a_d^{ini}$ hace que no se pueda añadir a_b a no ser que $a_b \in A_{base}$.

Es decir, a_b, a_c y a_d son interdependientes, por lo cual no se puede añadir ninguna acción. Esto supone una limitación en la capacidad del planificador para resolver problemas con interacciones complejas.

Para superar esta limitación, en TFLAP se ha añadido un procesado especial de las precondiciones *at end* como se detalla en la sección 3.2.1. Gracias a este procesado especial TFLAP es capaz de resolver problemas con distintos tipos de interacciones como *interaction*, en cuyo caso proporciona el resultado de la figura 4.6 cuya representación gráfica se muestra en la figura 4.5.

```
0.001: (action_A) [10.000]
0.001: (action_A) [10.000]
0.002: (action_B) [15.000]
0.003: (action_C) [4.000]
10.002: (action_D) [3.000]
```

Figura 4.6: Solución de problema con interacciones

4.1.3. Resultados de concurrencia

En esta sección se presentan los resultados para problemas concurrentes de TFLAP y otros dos planificadores del estado del arte, se trata de los planificadores usados en las comparativas de la sección 5.4. El problema 1 corresponde al problema descrito en la sección 4.1.1 y el problema 2 es el problema descrito en 4.1.2. El resto de problemas son casos simples de relaciones concurrentes (como los introducidos en la sección 4.1) que se describen en el trabajo [4] y que muchos planificadores temporales no son capaces de resolver.

Tal como se describe en la sección 5.1, se comprueba la validez de los planes mediante la herramienta `http://planning.cis.strath.ac.uk/VAL`. El *makespan* y número de acciones del plan solución proporcionado para cada problema se muestra en la tabla 4.2 en la cual se muestra *makespan*/acciones. En la tabla 4.2 se indica como error cuando el planificador genera un plan pero este es erróneo, por ejemplo si no se cumple alguna precondición de alguna acción.

Problema	OPTIC	TFD	**TFLAP**
1	10.002/3	20.021/4	**10.003/3**
2	NoPlan	NoPlan	**15.000/5**
3	4.000/2	4.001/2	**4.001/2**
4	4.000/2	Error	**4.001/2**
5	Error	4.012/3	**4.002/2**
6	Error	4.021/3	**4.002/2**
7	4.001/3	Error	**4.002/2**
8	10.002/5	8.021/7	**5.002/3**

Cuadro 4.2: Resultados del domino concu: *makespan*/acciones

En la tabla 4.3 se muestra el tiempo de ejecución en segundos para cada uno de los problemas. Se trata del tiempo total indicado por el comando *time* de Linux. Debido a que son problemas muy simples, todos los tiempos deberían ser pequeños, aunque no es así en el caso de OPTIC cuando proporciona planes erróneos.

Problema	OPTIC	TFD	**TFLAP**
1	0.00	0.06	**0.13**
2	0.00	0.05	**0.13**
3	0.00	0.06	**0.14**
4	0.11	0.06	**0.13**
5	856.44	0.07	**0.13**
6	1835.23	0.06	**0.14**
7	0.00	0.07	**0.14**
8	0.12	0.08	**0.19**

Cuadro 4.3: Tiempo de cálculo en segundos del dominio concu

Con estos resultados se observa que TFLAP es el único planificador analizado capaz de resolver correctamente problemas con cualquier tipo de concurrencia entre acciones. En la sección 5.4 se muestra el sumario de estos resultados y se analiza el comportamiento de los planificadores en una batería de problemas de otros dos dominios concurrentes.

4.2. Planificación temporal en problemas del mundo real

En esta sección se plantea la aplicabilidad de un planificador como TFLAP a la vida real. Un planificador puede resultar útil para cualquier tipo de tarea que implique seleccionar las acciones a realizar y elegir la secuencia de las acciones. Pero requiere que las acciones se puedan describir de forma suficientemente precisa de forma que las entienda el planificador, por ejemplo con PDDL.

Un gran grupo de tareas son las relacionadas logística y transporte: en estos problemas hay que transportar objetos o personas entre distintas localizaciones mediante diferentes medios de transporte. Ejemplos de este tipo de tareas son el clásico problema del viajante, variantes de *ridesharing* o transporte de mercancías.

Otro grupo de tareas son las que implican el uso de un recurso limitado y con gran demanda. En estos problemas existe algún recurso que se debe utilizar en múltiples ocasiones para resolver el problema. Ejemplos de estos problemas son la organización de contenedores en un puerto, atraque de barcos o la planificación de tareas de equipos remotos como sondas espaciales.

El abanico de posibles aplicaciones es prácticamente ilimitado y se encuentran en múltiples ámbitos. A modo ilustrativo de aplicabilidad, nos centraremos en la gestión de accidentes de tráfico que se detalla en 4.3.

4.3. Gestión de accidentes de tráfico

La gestión de accidentes de tráfico es un problema que requiere una planificación sistemática y coordinación entre equipos para mejorar la atención a las víctimas del accidente. En el caso de los accidentes de tráfico encontramos un continuo aumento de costes, atención mediática y objetivos cada vez más exigentes unido a la necesidad de reducir costes.

De esta forma se plantea el uso de un planificador, TFLAP, como ayuda a la gestión de accidentes de tráfico pues se trata de una tecnología que permite obtener planes de calidad de forma rápida. Es un problema de gran complejidad, donde el espacio de posibles estados y configuraciones del incidente puede ser enorme, y que requiere una solución en tiempo real. Uno de los principales objetivos de la gestión de incidentes en Redes de Tráfico de Carreteras (RTC) es responder a las necesidades del accidente y asignar recursos y equipos de manera segura y eficiente [18].

La codificación de este dominio se ha realizado en el contexto de la red europea *Autonomic Road Transport Support* (ARTS, `http://www.cost-arts.org`) en la que participan académicos e ingenieros de transporte que se enfrentan a este tipo de situaciones cotidianamente. En el diseño del conjunto de requerimientos del problema de planificación de un incidente en una RTC intervinieron varios especialistas en tráfico, participantes en la red europea ARTS y se emplearon varios manuales de gestión de tráfico de incidentes [18, 12, 1]. Particularmente, la codificación del problema se inspiró en el modelo de gestión de accidentes en RTC en el Reino Unido que depende de las *Highways Agency* (HA) y en el que intervienen policía, ambulancia, bomberos, servicios de reparación de averías, etc. [12].

4.3.1. Análisis del problema

Se parte de la definición proporcionada por expertos para realizar una descripción sencilla del problema que posteriormente se puede ampliar incrementando su realismo. Las fases de un incidente son: detección, verificación, respuesta, gestión, recuperación y restauración. En esta primer aproximación se asume que el accidente ha sido detectado siendo necesario el planificar para las siguientes fases.

El escenario de un accidente de tráfico es una red de carreteras que interconecta diferentes poblaciones. Se identifican diferentes proveedores de servicios responsables de diferentes facetas de la resolución de un accidente de tráfico. Cada grupo de proveedores de servicios dispondrá de bases localizadas en las poblaciones y vehículos que se desplazarán al lugar del accidente utilizando la propia red de carreteras. Los proveedores de servicios básicos en un accidente de tráfico son:

- La policía que se encarga de organizar el tráfico y coordinar al resto de equipos.
- El equipo médico que se encarga de atender a los heridos y trasladarlos a los hospitales.

- Los bomberos se encargan de rescatar a personas atrapadas en los vehículos accidentados y de extinguir incendios.
- Las grúas retiran los vehículos accidentados de la carretera y los depositan en garajes de forma que la circulación pueda volver a la normalidad.

Un incidente o problema RTC es una situación en la que se ha producido uno o más accidentes de tráfico en la red de carreteras. Un accidente está localizado en un punto de la red de carreteras y se compone de los siguientes elementos:

- Vehículos accidentados cada uno de los cuales puede estar incendiado o no.
- Personas víctimas del accidente cada una de las cuales puede estar atrapada en un vehículo o haber conseguido salir; adicionalmente, las víctimas pueden estar heridas o ilesas.

Para solucionar el incidente, los diferentes equipos deben actuar de forma coordinada, en unas ocasiones secuencial, en otras simultáneamente. Esto genera diferentes tipos de concurrencia entre acciones. A continuación se describen diferentes tipos de relaciones entre acciones indicando el tipo de precondición involucrada:

- La policía debe haber evaluado un accidente antes de que el resto de equipos puedan empezar a trabajar (precondición *at start*).
- Debe haber un agente de policía controlando la zona y dirigiendo el tráfico mientras otro equipo esté realizando cualquier actuación (precondición *over all*).
- No puede haber personas atrapadas en un vehículo accidentado cuando se termina de cargar en una grúa para transportarlo a un garaje (precondición *at end*).
- Para cargar una persona en una ambulancia es necesario que la persona no esté inicialmente atrapada (precondición *at start*) y que al completar la carga se le hayan prestado los primeros auxilios (precondición *at end*).
- Para iniciar cualquier acción (excepto extinguir el incendio) en un vehículo accidentado este no puede estar incendiado (precondición *at start*).
- Un equipo solo puede realizar una acción a la vez, por lo que al empezar una acción no puede estar realizando otra acción (precondición *at start*).

4.3.2. Resultados para problemas RTC

Los problemas de accidentes de tráfico contienen gran cantidad de información que debe ser coherente entre sí. De esta forma describir los problemas directamente en PDDL no resulta viable, sino que conviene hacerlo de forma automática a partir de ciertos datos básicos. Actualmente existen herramientas que pueden ayudar en este proceso como `https://code.google.com/p/itsimple` [27].

Se resuelven varios problemas de distinta complejidad con distintos planificadores (los mismos que se utilizan en la sección 5.4). A continuación se describen los diferentes problemas y la tabla 4.4 resume las características principales de los problemas indicando el número de elementos que contiene cada problema:

Elemento Problema	1	2	3	4
Ciudades	2	4	4	4
Ambulancias	1	4	4	4
Coches patrulla	1	2	2	2
Brigadas bomberos	1	2	2	2
Grúas	1	2	2	2
Accidentes	1	1	1	2
Vehículos accidentados	1	1	3	5
Víctimas humanas	1	0	3	6

Cuadro 4.4: Cantidad de los principales elementos de problemas RTC

1. Un problema muy simple con dos ciudades, una carretera que los une y un accidente en el que hay un vehículo con una persona atrapada en su interior. En este caso la dotación RTC consiste en un único vehículo de cada tipo.

2. Un problema de prueba de escenario con cuatro ciudades conectadas por la red de carreteras según se muestra en la figura 4.7 y con los equipos RTC:

   - Equipo médico: un hospital y cuatro ambulancias.
   - Policía: dos estaciones y un coche patrulla en cada una.
   - Bomberos: un cuartel de bomberos con dos vehículos.
   - Retirada de vehículos: un garaje y dos grúas.

   En este caso solo hay un vehículo accidentado e incendiado entre las ciudades 1 y 2. No hay víctimas humanas que atender. Este escenario (red de carreteras y equipos RTC) es el que se usa en los siguiente problemas).

3. Un problema con un accidente en el escenario previamente descrito. En este caso hay tres vehículos accidentados, uno de ellos incendiado y tres personas (una ilesa, otra herida y otra herida y atrapada en un vehículo).

4. Un problema de una complejidad intermedia en el que hay dos accidentes en el escenario previamente descrito. Cada accidente tiene dos o tres vehículos accidentados (alguno inicialmente incendiado) y tres personas (algunas inicialmente atrapadas y algunas ilesas).

La figura 4.8 muestra la solución de TFLAP para el primer problema. Dicha solución se compone, de forma resumida, por:

- Desplazar los diferentes vehículos RTC al accidente.
- Preparar la zona por parte de la policía.

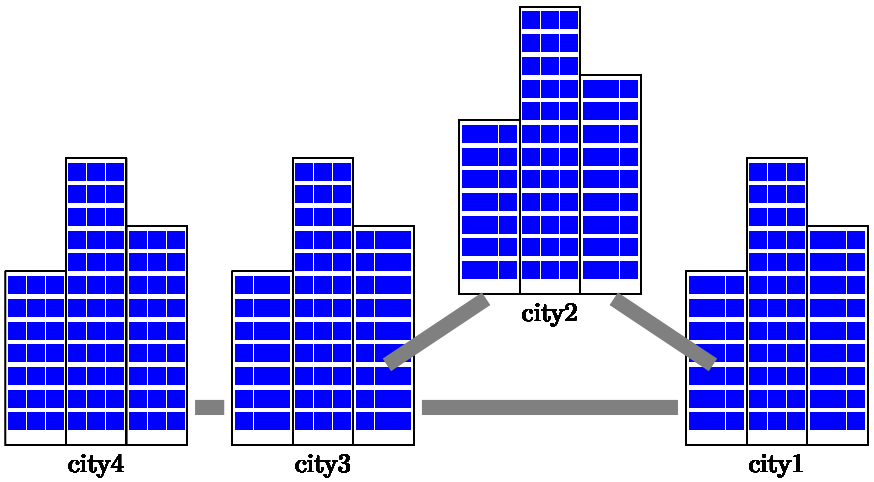

Figura 4.7: Mapa de carreteras

```
0.001000: (MoveRTCvehicle policar1 city1 road2) [0.800000]
0.001000: (MoveRTCvehicle firebri1 city2 road2) [1.500000]
0.802000: (EvaluateAccident policar1 road2) [3.000000]
0.001000: (MoveRTCvehicle tow1 city2 road2) [1.333333]
3.803000: (ControlAccident policar1 road2) [30.000000]
3.804000: (ReleaseHumanVictim firebri1 vic1 viccar1 policar1 road2 num00 num01)
    [3.000000]
6.805000: (LoadCivilCarToTow tow1 viccar1 policar1 road2 num00 num01) [3.000000]
9.806000: (MoveRTCvehicle tow1 road2 city2) [1.333333]
11.140333: (UnloadVictim viccar1 tow1 city2) [1.000000]
12.141333: (EnterCivilCarToGarage viccar1 garage1 city2) [1.000000]
0.001000: (MoveRTCvehicle amb1 city1 road2) [1.090909]
6.805000: (FirstAidHumanVictim amb1 vic1 policar1 road2) [5.000000]
11.806000: (LoadHumanVictimToAmbulance amb1 vic1 policar1 road2) [1.000000]
12.807000: (MoveRTCvehicle amb1 road2 city1) [1.090909]
13.898909: (UnloadVictim vic1 amb1 city1) [1.000000]
14.899909: (HospitalizeHumanVictim vic1 hos1 city1) [1.000000]
```

Figura 4.8: Solución de TFLAP al problema 1 de RTC

- Realizar las acciones correspondientes de cada equipo: liberar personas atrapadas, extinguir incendios, prestar primeros auxilios.
- Trasladar a las víctimas a los centros de atención: hospital para las personas, garaje para los coches accidentados. En este caso particular solo hay un coche accidentado y una persona víctima del accidente.

Las descripciones PDDL determinan las relaciones entre acciones y la duración de cada acción. Se obtienen planes solución cuya duración (*makespan* en minutos con tres dígitos decimales) se muestran en la tabla 4.5 junto al número de acciones (siguiendo el formato *makespan*/acciones).

Problema	OPTIC	TFD	**TFLAP**
1	64.802/20	34.831/17	**33.803/16**
2	51.670/16	51.728/12	**51.674/12**
3	102.338/53	188.075/49	**140.008/52**
4	154.740/98	223.865/94	**132.006/92**

Cuadro 4.5: Resultados del domino RTC: *makespan*/acciones

Para un mismo problema todos los planificadores proporcionan un resultado con una cantidad similar de acciones. Pero a medida que el problema es más complejo aparecen diferencias importantes en el *makespan*. Cabe destacar el caso del problema 1, la solución propuesta por OPTIC incluye una acción innecesaria de que la policía controle la ciudad 1, esto incrementa la duración del plan en 30 minutos.

Las diferencias de *makespan* son muy relevantes en estos problemas, por ejemplo, en el problema más complejo, en la solución propuesta por TFLAP la red de carreteras está despejada (la circulación vuelve a la normalidad) y las víctimas están atendidas en menor tiempo. En los incidentes RTC las diferencias de tiempo, por pequeñas que sean, pueden tener efectos importantes pues ese tiempo puede significar la diferencia entre la vida y la muerte de varias personas.

Estos resultados muestran la viabilidad de utilizar un planificador como TFLAP para resolver problemas reales como los incidentes RTC, aunque se necesita realizar pruebas con múltiples escenarios antes de pensar en la implantación de un sistema de estas características. Posiblemente resulte interesante mejorar la heurística para generar planes buscando el mejor *makespan*.

El tiempo de cálculo en segundos de cada planificador para cada problema se muestra en la tabla 4.6. La forma de medir el tiempo y comprobar los resultados es la que se describe en 5.1: la validez del plan se comprueba mediante la herramienta `http://planning.cis.strath.ac.uk/VAL` y el tiempo es el tiempo total que indica el comando *time* de Linux.

Problema	OPTIC	TFD	**TFLAP**
1	0.08	0.18	**3.73**
2	0.09	0.79	**0.75**
3	1.60	3.82	**68.75**
4	35.39	10.42	**297.28**

Cuadro 4.6: Tiempo de cálculo en segundos del dominio RTC

Con estos resultados de tiempo de ejecución se pone de manifiesto que el tiempo de ejecución de TFLAP es considerablemente mayor que el de OPTIC y TFD. Esta circunstancia se analiza en el capítulo 5.

Capítulo 5

Evaluación experimental de TFLAP

En este capítulo se presentan los resultados experimentales de TFLAP y se comparan con los resultados de diferentes planificadores del estado del arte. En una primer sección se describe cómo se han ejecutado los diferentes planificadores. A continuación se presentan los problemas de competición usados. Después se presentan los resultados obtenidos separando los problemas de tipo general y los problemas concurrentes. Por último se exponen las conclusiones generales de esta batería de pruebas.

5.1. Generación de resultados

Para evaluar los resultados se usa la herramienta `http://planning.cis.strath.ac.uk/VAL` que indica si el plan es correcto y su duración. De esta forma se comprueba de forma automática, independiente del planificador, si el plan proporcionado es correcto y se obtiene la duración del plan.

También se compara el tiempo (segundos) de cálculo usado, concretamente el valor que indica como tiempo total el comando *time* de Linux. Para cada planificador y problema el tiempo de ejecución se limita a media hora (1800 segundos), de forma que si se alcanza dicho tiempo, se aborta la ejecución. A continuación se enumeran los diferentes planificadores usados en la comparativa:

- **TFD** (ver la sección 2.3.2) está basado en el sistema de planificación *Fast Downward* modificado para temporal. TFD se compila directamente para la máquina que lo ejecuta con gcc 4.7.2 y se ejecuta directamente sobre el sistema operativo.
- **YAHSP3** es un planificador del estado del arte, pero que sigue un paradigma más secuencial por lo que tiene un tratamiento pobre de la concurrencia. Al igual que TFD, se compila directamente para la máquina que lo ejecuta con gcc 4.7.2 y se ejecuta directamente sobre el sistema operativo.

- **OPTIC** (ver la sección 2.3.3) es un planificador de orden parcial (ver la sección 2.2.1) similar a TFLAP (ver la sección 3.1). OPTIC se descarga ya compilado y se ejecuta directamente sobre el sistema operativo.
- El planificador desarrollado en este trabajo, **TFLAP**, se compila en una máquina virtual con un compilador de *Delphi* que soporte todas las estructuras del lenguaje usadas en **TFLAP**. En cuanto a la ejecución de TFLAP, debido a que es un programa Windows existen distintas opciones: en una máquina virtual, utilizando *wine*, con un equipo distinto con Windows o con el mismo equipo con arranque dual.

Para las pruebas se han utilizado con un equipo con las siguientes características:

- Sistema operativo: Debian GNU/Linux 7.8 (wheezy) con entorno gráfico KDE 4.
- Procesador: Pentium Dual-Core E5200 a 2.50GHz (5000 bogomips).
- Memoria: cache de 2048 KB, principal de 4GB DDR2 800.

Al no disponer de un equipo con Windows y Linux instalado simultáneamente, TFLAP se ha ejecutado con *wine*. En cualquier caso, todas las alternativas de ejecución de TFLAP van a alterar los tiempos de ejecución, pues como mínimo hay un cambio de sistema operativo. Por otra parte, está la alteración provocada por el uso de un compilador distinto para cada planificador: solo usan el mismo compilador TFD y YAHSP3 Además, al limitar la ejecución a 30 minutos, las modificación de velocidad puede implicar que cambie el número de problemas resueltos. Hay que tener este hecho muy presente en las comparaciones de forma que los resultados de tiempos de ejecución son los menos relevantes.

5.2. Problemas de competición

Como comparativa se va a resolver diferentes problemas de varios dominios de competición con cada uno de los planificadores. Se utilizan diferentes dominios (listados en la tabla 5.1) que se han usado en diferentes ediciones de las competiciones IPC (*International Planning Competition*, `http://icaps-conference.org/index.php/Main/Competitions`) de los últimos años y que se continúan usando en las competiciones actuales. En cada uno de los dominios se analizan 20 problemas con distinto grado de complejidad.

Dominio	Competición en la que aparece	Descripción básica
Depots	IPC 2002	Combina los dominios clásicos de transporte y mundo de bloques (apilar bloques en torres y acceder a ellos)
DriverLog	IPC 2002	Es un dominio de transporte: una empresa de transporte con conductores, camiones y paquetes que hay que llevar de un lugar a otro
ZenoTravel	IPC 2002	Vuelos de pasajeros entre distintas localizaciones
Satellite	IPC 2004	Satélites con instrumentos que realizan medidas de diferentes objetivos
Rovers	IPC 2006	Rovers de exploración planetaria que se desplazan y realizan medidas que envían a la base
Elevators	IPC 2011	Un edificio de varias plantas con ascensores de diferentes características en el que hay pasajeros que desean ir de una planta a otra
Matchcellar	IPC 2011	Cambiar fusibles usando la iluminación de cerillas
OpenStacks	IPC 2011	Un fabricante con ciertos pedidos de producción, cada uno incluye distintos productos a producir
TurnAndOpen	IPC 2011	Robots que se desplazan transportando objetos
Parking	IPC 2014	Modelo de un aparcamiento, similar al mundo de bloques

Cuadro 5.1: Dominios de competición usados para comparar

5.3. Resultados para problemas no concurrentes

En esta sección se presentan los resultados de los planificadores (TFD, OPTIC, YAHSP3 y TFLAP) para diferentes dominios usados en las competiciones en los cuales no exista interacción entre acciones. Es decir, se prueban todos los dominios de la tabla 5.1 excepto *Matchcellar* y *TurnAndOpen*.

Debido a la cantidad de datos, conviene resumirlos en tablas para poder obtener conclusiones. Por una parte se muestra el cubrimiento: el número total de planes resueltos por cada planificador para cada dominio, sin importar su calidad ni el tiempo de cálculo necesario. Adicionalmente se presenta la calidad promedio de los planes proporcionados por cada planificador para los problemas de cada dominio: su duración temporal (*makespan*) y número de acciones. También se presenta el tiempo de cálculo promedio usado por cada planificador para los problemas de cada dominio.

Para promediar (tanto *makespan*, como número de acciones, como tiempo de cálculo) se deben considerar solo los problemas que resuelven todos los planificadores. De esta forma se comparan los planificadores en igualdad de condiciones,

sin penalizar a un planificador que resuelva un problema que los otros no consiguen resolver.

A continuación se presentan los resultados resumidos por dominio. La tabla 5.2 muestra el número de problemas totales y resueltos por cada planificador. De esta forma se obtiene qué planificador resuelve más problemas para cada dominio y en global.

Dominio	Total	OPTIC	TFD	YAHSP3	**TFLAP**
Depots	20	8	4	20	**13**
DriverLog	20	15	6	15	**15**
Elevators	20	3	20	20	**8**
OpenStacks	20	20	20	20	**20**
Parking	20	13	20	20	**20**
Rovers	20	19	20	20	**18**
Satellite	20	15	20	20	**19**
ZenoTravel	20	12	20	20	**20**
Suma	160.000	105.000	130.000	155.000	**133.000**

Cuadro 5.2: Problemas resueltos

La tabla 5.3 muestra la duración del plan (*makespan*) medio de los planes generados por cada planificador y el número de acciones con el formato *makespan*/acciones, lo que da una información de la calidad del plan.

Dominio	OPTIC	TFD	YAHSP3	**TFLAP**
Depots	44.34/19.00	51.44/29.00	61.52/20.00	**49.68/17.67**
DriverLog	89.67/27.50	133.36/29.83	129.27/28.17	**96.18/22.67**
Elevators	595.34/141.33	711.33/127.33	1660.40/374.33	**438.37/99.00**
OpenStacks	191.76/102.90	800.30/102.90	223.42/102.90	**173.13/102.90**
Parking	4.62/19.85	9.62/19.77	5.47/16.85	**10.78/14.00**
Rovers	122.42/32.71	143.46/29.47	113.65/30.29	**121.25/27.71**
Satellite	58.47/41.93	96.00/33.47	98.70/44.20	**94.28/27.33**
ZenoTravel	818.18/28.92	884.36/23.33	752.08/17.67	**667.09/14.83**

Cuadro 5.3: Sumario de resultados: *makespan*/Acciones

Mientras que la tabla 5.4 muestra los tiempos de ejecución en segundos. Nuevamente se promedia para los problemas de cada dominio que resuelven todos los planificadores.

Dominio	OPTIC	TFD	YAHSP3	**TFLAP**
Depots	0.22	46.95	0.08	**0.23**
DriverLog	72.16	1.63	0.03	**2.08**
Elevators	798.50	46.72	1.05	**108.10**
OpenStacks	1.97	24.47	0.07	**25.57**
Parking	2.34	83.69	0.20	**36.88**
Rovers	1.43	0.87	0.01	**1.31**
Satellite	73.80	4.58	0.02	**8.24**
ZenoTravel	48.70	1.17	0.01	**0.28**

Cuadro 5.4: Tiempo de cálculo medio en segundos

Para facilitar la comparación se normalizan los resultados de *makespan*, número de acciones y tiempo de ejecución. La normalización consiste en, dado un dominio, seleccionar el valor mínimo de los planificadores, el valor normalizado es dicho mínimo dividido entre el valor de cada planificador. De esta forma que se asigna un valor de 1 al planificador que proporciona el menor resultado y un valor inferior a medida que los resultados son peores. Esto permite hacer un resumen por planificador promediando su valor normalizado para cada dominio. El valor normalizado se puede interpretar como una relación, de forma que un 1 es el mejor en todos los casos y un 0,9 representa el 90 % del mejor en todos los casos. Como se aprecia en las tablas, es habitual que no haya un planificador que sea el mejor en todos los casos (ninguno tiene un valor de 1).

En cuanto a *makespan* se obtiene la tabla 5.5. Para el número de acciones se tiene la tabla 5.6. Y para tiempo de cálculo, la tabla 5.7

Dominio	OPTIC	TFD	YAHSP3	**TFLAP**
Depots	1.000	0.862	0.721	**0.893**
DriverLog	1.000	0.672	0.694	**0.932**
Elevators	0.736	0.616	0.264	**1.000**
OpenStacks	0.903	0.216	0.775	**1.000**
Parking	1.000	0.480	0.845	**0.429**
Rovers	0.928	0.792	1.000	**0.937**
Satellite	1.000	0.609	0.592	**0.620**
ZenoTravel	0.815	0.754	0.887	**1.000**
Promedio	0.923	0.625	0.722	**0.851**

Cuadro 5.5: *makespan* normalizado

Dominio	OPTIC	TFD	YAHSP3	**TFLAP**
Depots	0.930	0.609	0.884	**1.000**
DriverLog	0.824	0.760	0.805	**1.000**
Elevators	0.700	0.778	0.264	**1.000**
OpenStacks	1.000	1.000	1.000	**1.000**
Parking	0.705	0.708	0.831	**1.000**
Rovers	0.847	0.940	0.915	**1.000**
Satellite	0.652	0.817	0.618	**1.000**
ZenoTravel	0.513	0.636	0.839	**1.000**
Promedio	0.771	0.781	0.770	**1.000**

Cuadro 5.6: Número de acciones normalizado

Dominio	OPTIC	TFD	YAHSP3	**TFLAP**
Depots	0.364	0.002	1.000	**0.348**
DriverLog	0.000	0.018	1.000	**0.014**
Elevators	0.001	0.022	1.000	**0.010**
OpenStacks	0.036	0.003	1.000	**0.003**
Parking	0.085	0.002	1.000	**0.005**
Rovers	0.007	0.011	1.000	**0.008**
Satellite	0.000	0.004	1.000	**0.002**
ZenoTravel	0.000	0.009	1.000	**0.036**
Promedio	0.062	0.009	1.000	**0.053**

Cuadro 5.7: Tiempo de cálculo normalizado

Estos resultados se pueden resumir en la tabla 5.8 en la que se clasifica como 1 al mejor planificador y 4 al peor en cada una de las áreas. Adicionalmente se indica entre paréntesis el resultado de cada planificador: número de problemas resueltos y el valor normalizado para el resto de áreas.

Área	OPTIC	TFD	YAHSP3	TFLAP
Problemas resueltos	4 (105)	3 (130)	1 (155)	2 (133)
Calidad: *makespan*	1 (0.923)	4 (0.625)	3 (0.722)	2 (0.851)
Número de acciones	3 (0.771)	2 (0.781)	4 (0.770)	1 (1.000)
Tiempo de cálculo	2 (0.062)	4 (0.009)	1 (1.000)	3 (0.053)

Cuadro 5.8: Clasificación de los planificadores en dominios no concurrentes

Se observa que YAHSP3 es el planificador que más problemas resuelve, pues resuelve todos los problemas de todos los dominios excepto *DriverLog* en el que también consigue una buena cobertura. El planificador que menos problemas resuelve es OPTIC que solo consigue resolver todos los problemas de *OpenStacks*. TFD consigue resolver todos los problemas de casi todos los dominios, pero tiene un cobertura baja en los dominios *Depots* y *DriverLog*, dos de los dominios más antiguos usados en esta comparativa. TFLAP resuelve un porcentaje alto de

problemas en casi todos los dominios excepto *Elevators*, aunque solo resuelve todos los problemas de tres dominios.

En cuanto a calidad de los planes, OPTIC y TFLAP proporcionan los menores *makespan* medios en cuatro y tres dominios respectivamente. Ningún planificador proporciona planes de mejor (ni peor) calidad que el resto en todos los dominios. En cambio TFLAP es el planificador que proporciona planes con menos acciones en todos los dominios como efecto de su heurística orientada a acciones. A pesar de la heurística orientada a acciones de TFLAP, sus planes son los de mayor calidad para los dominios: *Elevators*, *OpenStacks* y *ZenoTravel*. Y solo OPTIC consigue una mejor clasificación en cuanto a calidad de los planes siendo pequeña la diferencia entre ambos planificadores como se puede ver en los valores promediados de la tabla 5.5. YAHSP3 es claramente el planificador más rápido, tanto globalmente como en cada uno de los dominios.

Cabe destacar que ningún planificador resuelve todos los problemas del dominio *DriverLog*, a pesar de ser uno de los más antiguos y que se están usando los problemas de la primer competición en la que apareció ese dominio.

A modo de resumen, podemos decir que cada planificador tiene un dominio que le resulta especialmente complejo bien para resolver los problemas, bien para proporcionar resultados de calidad. También se puede afirmar que TFD y YAHSP3 son los planificadores que resuelven más problemas, mientras que OPTIC y TFLAP proporcionan planes de mayor calidad. Para estos problemas TFLAP proporciona unos resultados compensados entre calidad de los planes y cantidad de problemas resueltos, lo que se puede considerar un resultado prometedor de cara a su aplicabilidad y resultados en competiciones.

5.4. Resultados para problemas concurrentes

En esta sección se presentan los resultados de los planificadores (TFD, OPTIC y TFLAP) para dominios con concurrencia usados en las competiciones: *Matchcellar* y *TurnAndOpen*. Adicionalmente se incluye un nuevo dominio con problemas básicos de concurrencia, es el dominio concurrente que se describe en la sección 4.1.

En esta sección se ha omitido el planificador YAHSP3 debido a que es un planificador no concurrente el cual, en líneas generales, no es capaz de resolver problemas para los que no exista una solución temporal secuencial. A pesar de eso sí consigue resolver algún problema aislado debido a que la aproximación de YAHSP3 es compatible con dicho problema, pero es habitual que proporcione soluciones erróneas, bien porque el plan no consigue los objetivos, bien porque en alguna acción no se cumplen las precondiciones.

Al igual que se hizo para los dominios no concurrentes, se presentan los resultados resumidos por dominio. La tabla 5.9 muestra el número de problemas totales y resueltos por cada planificador. De esta forma se obtiene qué planificador resuelve más problemas para cada dominio.

Dominio	Total	OPTIC	TFD	**TFLAP**
concurrente	8	5	5	**8**
Matchcellar	20	6	20	**19**
TurnAndOpen	20	9	19	**8**
Suma	48.000	20.000	44.000	**35.000**

Cuadro 5.9: Problemas resueltos

Se promedian los resultados de cada planificador para cada dominio considerando solo los problemas que resuelven todos los planificadores. La tabla 5.10 muestra el tiempo de plan (*makespan*) medio de los planes generados por cada planificador y el número de acciones con el formato *makespan*/acciones, lo que da una información de la calidad del plan.

Dominio	OPTIC	TFD	**TFLAP**
concurrente	8.00/3.33	10.68/4.33	**6.34/2.67**
Matchcellar	22.01/16.50	22.12/16.50	**22.01/16.50**
TurnAndOpen	49.55/127.50	90.51/130.62	**48.80/100.88**

Cuadro 5.10: Sumario de resultados: *makespan*/Acciones

Mientras que la tabla 5.11 muestra los tiempos de ejecución. Nuevamente promedia para los problemas de cada dominio que resuelven todos los planificadores.

Dominio	OPTIC	TFD	**TFLAP**
concurrente	0.04	0.07	**0.15**
Matchcellar	51.51	0.19	**1.63**
TurnAndOpen	51.66	5.12	**40.81**

Cuadro 5.11: Tiempo de cálculo medio en segundos

Al igual que se hizo en la sección 5.3 para los problemas no concurrentes, se normalizan los resultados para poder comparar los planificadores agrupando los resultados de diferentes dominios. En cuanto a *makespan* se obtiene la tabla 5.12. Para el número de acciones se tiene la tabla 5.13. Y para tiempo de cálculo, la tabla 5.14

Dominio	OPTIC	TFD	**TFLAP**
concurrente	0.792	0.594	**1.000**
Matchcellar	1.000	0.995	**1.000**
TurnAndOpen	0.985	0.539	**1.000**
Promedio	0.926	0.709	**1.000**

Cuadro 5.12: *makespan* normalizado

Dominio	OPTIC	TFD	**TFLAP**
concurrente	0.802	0.617	**1.000**
Matchcellar	1.000	1.000	**1.000**
TurnAndOpen	0.791	0.772	**1.000**
Promedio	0.864	0.796	**1.000**

Cuadro 5.13: Número de acciones normalizado

Dominio	OPTIC	TFD	**TFLAP**
concurrente	1.000	0.571	**0.267**
Matchcellar	0.004	1.000	**0.117**
TurnAndOpen	0.099	1.000	**0.125**
Promedio	0.368	0.857	**0.170**

Cuadro 5.14: Tiempo de cálculo normalizado

Nuevamente los resultados se resumen en la tabla de clasificación 5.15 en la que se clasifica como 1 al mejor planificador y 4 al peor en cada una de las áreas. Adicionalmente se indica entre paréntesis el resultado de cada planificador: número de problemas resueltos y el valor normalizado para el resto de áreas.

Área	OPTIC	TFD	TFLAP
Problemas resueltos	3 (20)	1 (44)	2 (35)
Calidad: *makespan*	2 (0.926)	3 (0.709)	1 (1.000)
Número de acciones	2 (0.864)	3 (0.796)	1 (1.000)
Tiempo de cálculo	2 (0.368)	1 (0.857)	3 (0.170)

Cuadro 5.15: Clasificación de los planificadores en dominios concurrentes

En los problemas concurrentes cabe destacar que TFLAP es el único capaz de resolver todos los problemas del dominio concurrente. Mientras que TFD es el que más problemas del dominio *TurnAndOpen* resuelve, lo que hace que sea el planificador que más problemas resuelve de esta batería. Esto se debe no a la complejidad del dominio *TurnAndOpen* sino al tamaño de los problemas de dicho dominio. De hecho TFLAP y OPTIC agotan el tiempo o la memoria antes de encontrar la solución. En el dominio *Matchcellar* se produce una situación similar.

En cambio, los problemas del dominio concurrente son extremadamente simples: se pueden resolver manualmente. Esto significa que no encontrar una solución a uno de estos problemas es muy relevante en cuanto a la capacidad del planificador para abordar diferentes tipos de interacciones. Un planificador incapaz de resolver alguno de los problemas del dominio concurrente, tiene algún tipo de limitación en cuanto a interacción entre acciones.

En cuanto a la calidad de los planes TFLAP es el que mejores resultados proporciona de *makespan* y acciones para el dominio concurrente debido a su

mejor manejo de interacciones. En el dominio *Matchcellar*, debido a su regularidad, todos los planificadores proporcionan planes de idéntica calidad. Por último, en el dominio *TurnAndOpen* TFLAP vuelve a proporcionar los planes de mayor calidad.

En cuanto a tiempo de ejecución, OPTIC y TFLAP son considerablemente más lentos que TFD. Siendo TFLAP más lento que OPTIC.

A modo de resumen, podemos afirmar que TFLAP es el mejor planificador para resolver problemas concurrentes y con interacciones complejas: resuelve problemas con todo tipo de interacciones entre acciones y genera planes de mayor calidad.

5.5. Conclusiones de los resultados

Como conclusión, podemos afirmar que TFLAP es el único planificador probado preparado para enfrentarse a dominios con relaciones complejas entre las acciones. Además proporciona buenos resultados (similares o mejores a la media del resto de planificadores del estado del arte) en cuanto a problemas resueltos y calidad de los planes generados. Mientras que lo habitual es tener una buena cobertura o generar planes de calidad, pero no las dos cosas a la vez como hace TFLAP.

Adicionalmente TFLAP se puede mejorar en cuanto a problemas resueltos y calidad de los planes utilizando mejores heurísticas que le permitan encontrar mejores soluciones expandiendo menos nodos.

Respecto a YAHSP3, que es el mejor planificador en dominios no concurrentes en cuanto a cobertura y velocidad, TFLAP aporta la capacidad de resolver problemas concurrentes y una calidad de resultados superior. Respecto a OPTIC los tiempos de ejecución son similares, pero TFLAP es capaz de resolver un mayor número de problemas. TFD y TFLAP resuelven una cantidad similar de problemas, pero TFLAP lo hace en menor tiempo y con mayor calidad.

TFLAP resulta un planificador lento, al menos al ejecutarse con *wine*. Esta menor velocidad afecta a la cantidad de problemas resueltos (reduciendo la cobertura) debido que en las pruebas realizadas el tiempo de ejecución para resolver cada problema está limitado. Pero no es posible comparar los planificadores en igualdad de condiciones debido a que actualmente TFLAP solo se puede compilar para Windows mientras que el resto de planificadores están compilados para Linux. De esta forma tenemos una nueva propuesta de mejora de TFLAP: que se pueda compilar en Linux para estar en igualdad de condiciones que el resto de planificadores.

Capítulo 6

Conclusiones

Tal como muestran los resultados, TFLAP es un planificador muy competitivo en comparación con otros planificadores del estado del arte, igualando o mejorando sus resultados generales. Como se aprecia en la tabla 5.8 en problemas no concurrentes TFLAP está en las primeras posiciones en cuanto a cobertura y calidad, mientras que ningún otro planificador está en las dos primeras posiciones en ambas facetas simultáneamente. Al analizar los resultados para problemas concurrentes, tal como se muestra en la tabla 5.15, TFLAP es el planificador que proporciona planes de mayor calidad. Adicionalmente TFLAP es el único planificador capaz de resolver todos los tipos de concurrencia. De esta forma, TFLAP iguala o mejora el funcionamiento del resto de planificadores del estado del arte analizados y el mayor beneficio que aporta TFLAP respecto al resto de planificadores analizados es su mayor capacidad para resolver problemas concurrentes.

Adicionalmente, TFLAP se ha aplicado con éxito a un problema del mundo real de gestión de accidentes de tráfico. Por lo que podemos afirmar que se trata de un planificador competitivo para problemas reales.

6.1. Líneas de trabajo futuro

Si bien los resultados de calidad y cobertura son muy positivos, TFLAP es un planificador lento en comparación con los tiempos de ejecución de los otros planificadores. En vistas a futuras líneas de trabajo se han detectado diferentes puntos de mejora:

- Poder compilar y ejecutar TFLAP en el mismo tipo de plataforma que el resto de planificadores, es decir, ordenadores con sistema operativo Linux.
- Heurística orientada a encontrar soluciones de calidad. En este sentido se estudiará diseñar heurísticas que tengan en cuenta el *makespan* del plan así como la dificultad de resolver el problema.

- Modificar la expresividad de TFLAP para que pueda aceptar problemas descritos en PDDL 3.0, es decir, para poder expresar: restricciones temporales, preferencias, *dead-lines*, valores numéricos. Esta mejora permitirá modelizar y resolver muchos más problemas del mundo real con un mayor grado de realismo.

En cuanto a la comparativa con los diferentes planificadores del estado del arte y la participación en competiciones tanto FLAP como TFLAP se encuentran limitados por el uso original de Delphi con un compilador para Windows. Mientras que el resto de planificadores se pueden compilar en Linux y en las competiciones se requiere el código fuente y un script para compilar. Esto supone un problema en las comparativas de los tiempos de ejecución ante la imposibilidad de ejecutar TFLAP en las mismas condiciones que el resto de planificadores, tal y como se comentó en la sección 5.5. En relación con esta cuestión, actualmente se está trabajando para reescribir el código de TFLAP en C++.

Como se comenta en la sección 3.3.3, TFLAP utiliza una heurística orientada a minimizar el número de acciones. Pero guiar un proceso de búsqueda hacia una solución con menor número de acciones no siempre garantiza el plan de menor duración. De esta forma tenemos una línea de trabajo futuro que se debe centrar en mejorar la heurística. El objetivo es doble pues debe encontrar una solución rápido, pero también debe proporcionar una solución de calidad.

Bibliografía

[1] Benesch. Traffic incident management operations guidelines. Iowa Department of Transportation, Federal Highway Administration, 1200 New jersey Avenue, SE, Washington, DC 20590, March 24 2011.

[2] J Benton, Amanda Jane Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 77, pages 2–10, 2012.

[3] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[4] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. When is temporal planning really temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1852–1859, 2007.

[5] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*, 2009.

[6] R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[7] 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.

[8] 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.

[9] 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.

[10] M. Ghallab and H. Laruelle. Representation and control in ixtet, a temporal planner. In *Proceedings 2nd. Int. Conference on AI Planning Systems*, pages 61–67, 1994.

[11] M. Ghallab, D. Nau, and P. Traverso. *Automated Task Planning. Theory and Practice*. Morgan Kaufmann, 2004.

[12] HA. Highways agency network management manual. Highways Agency, Network Services, Network Management Policy Team, City Tower, Piccadilly Plaza, MANCHESTER M1 4BE, 5.10 edition, July 2009.

[13] Malte Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, pages 161–170, 2004.

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

[15] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–287, 2004.

[16] S. Kambhampati. Notes from the ASU planning seminar: Planning methods in artificial intelligence, 2000.

[17] 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.

[18] N. Owens, A. Armstrong, P. Sullivan, C. Mitchell, D. Newton, R. Brewster, , and T. Trego. Traffic incident management handbook. Technical Report Office of Travel Management, Federal Highway Administration, 2000.

[19] J.S. Penberthy and D.S. Weld. Temporal planning with continuous change. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 2:1010–1015, 1994.

[20] M. Peot and D. Smith. Threat-removal strategies for partial-order planning. *Proceedings of the AAAI National Conference*, pages 492–499, 1993.

[21] S. Russell and P. Norving. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[22] Oscar Sapena, Eva Onaindia, and Alejandro Torreño. Forward-chaining planning with a flexible least-commitment strategy. In *International Conference of the Catalan Association for Artificial Intelligence (CCIA)*, pages 41–50, 2013.

[23] Oscar Sapena, Eva Onaindia, and Alejandro Torreño. Flap: Applying least-commitment in forward-chaining planning. *AI Communications*, 28(1):5–20, 2015.

[24] L. Sebastia, E. Onaindia, and E. Marzal. Decomposition of planning problems. *AI Communications*, 19, n.1:49–81, 2006.

[25] Alejandro Torreño, Eva Onaindia, and Oscar Sapena. A flexible coupling approach to multi-agent planning under incomplete information. *Knowl. Inf. Syst.*, 38(1):141–178, 2014.

[26] Alejandro Torreño, Eva Onaindia, and Oscar Sapena. FMAP: distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626, 2014.

[27] Tiago Stegun Vaquero, Victor Romero, Flavio Tonidandel, and José Reinaldo Silva. itsimple 2.0: An integrated tool for designing planning domains. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 336–343, 2007.

[28] D.S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4), 1994.

[29] Q. Yang. *Intelligent Planning. A Decomposition and Abstraction Based Approach*. Spinger-Verlag. Berlin, Heidelberg, 1997.