Planificación Inteligente: Problema Transporte

Jaime Ferrando Huertas, Javier Martínez Bernia

Febrero 2021, MIARFID, UPV

Contents

1 Introducción al problema

En esta memoria se presenta el trabajo realizado para la asignatura Planificación Inteligente. El objetivo de este trabajo es experimentar con diferentes planificadores para encontrar soluciones al dominio propuesto, en este caso un dominio de Transporte. En concreto, se hace uso de los planificadores FF, LPG y Optic. Se experimentará con distintos tipos de dominio (proposicional, temporal y con recursos numéricos) además de aplicar técnicas que utilizan los planificadores como el desarrollo de un árbol de estados POP y la creación de un *Graphplan*.

En la primera sección se hace una presentación del problema que se pretende resolver, seguida de tres secciones que resuelven el problema desde tres puntos de vista distintos: proposicional, temporal y con recursos. A estas secciones le siguen dos más donde se verá el desarrollo de un árbol POP y la creación de un *Graphplan*. Finalmente, hay una sección donde se exponen las conclusiones tras la realización del trabajo.

1.1 Problema Transporte

El problema a resolver es un dominio de transporte, en el que hay que llevar unos paquetes de un sitio a otro. Hay unas ciudades donde podemos encontrar tres tipos de localizaciones: casas, aeropuertos y estaciones. Hay tres tipos de medios de transporte: aviones, trenes y furgonetas. Los aviones se mueven entre aeropuertos de distinta ciudad, los trenes se mueven entre estaciones de cualquier ciudad y las furgonetas se mueven entre las localizaciones de una misma ciudad. En las ciudades hay unos conductores que pueden moverse dentro de ellas y que se necesitan para conducir las furgonetas. Finalmente, en las localizaciones están situados los paquetes, que pueden ser cargados en cualquier medio de transporte para ser movidos. En la figura 1 podemos ver un ejemplo gráfico del problema.

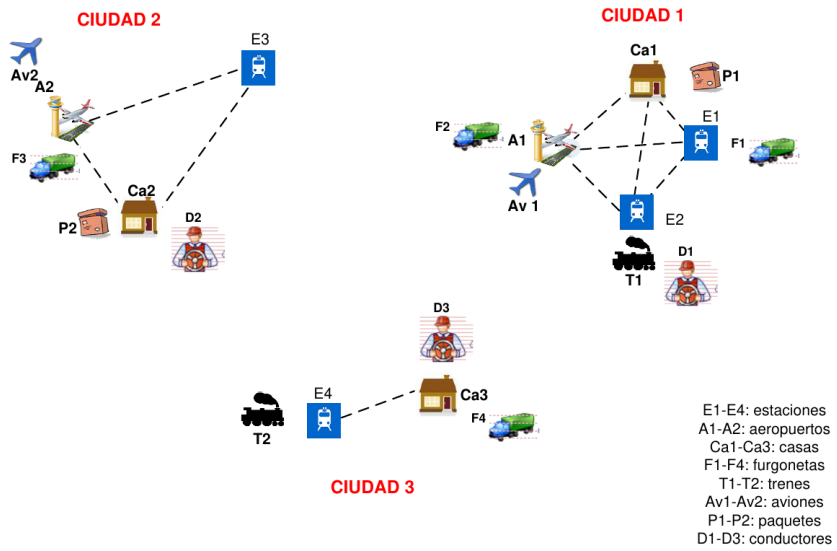

Figure 1: Ejemplo gráfico del dominio de transporte con tres ciudades [1].

2 Dominio Proposicional

En esta sección se presentan las definiciones de los predicados, de los operadores y del problema para el caso de un dominio proposicional.

2.1 Definición de los predicados

- (**in** ?c - ciudad ?x - loc): Modela las localizaciones que existen dentro de una ciudad. De esta manera sabemos qué localizaciones forman parte de una ciudad.
- (**at** ?x - (either paquete vehiculo conductor) ?l - loc): Modela las posiciones de paquetes/vehículos/conductores en las correspondientes localizaciones, de modo que podemos saber en todo momento en qué posición se encuentra cada objeto.
- (**loaded** ?v - vehiculo ?p - paquete): Modela cuando un paquete está cargado en un vehículo.
- (**on** ?c - conductor ?f - furgoneta): Modela cuando un conductor está subido en una furgoneta.

[1] Imagen extraída del enunciado del problema.

- (**empty** ?f - furgoneta): Modela cuando una furgoneta no tiene ningún conductor subido. Esto nos permite saber cuándo una furgoneta puede moverse o no.

2.2 Definición de los operadores

Para el modelado del dominio se ha decidido utilizar un total de 8 operadores. En los siguientes subapartados se explica cada uno en detalle.

2.2.1 *Load*

load ?v - vehiculo ?l - loc ?p - paquete

El operador *load* sirve para cargar un paquete en un vehículo. Para poder cargar un paquete en un vehículo, ambos deben estar en la misma localización. Simplemente se necesita hacer esa comprobación. Los efectos que provoca la acción es que se elimina el predicado **at** del paquete y se añade el predicado **loaded** para modelar que el paquete se ha cargado.

2.2.2 *Unload*

unload ?v - vehiculo ?l - loc ?p - paquete

Este operador hace lo contrario del anterior, descarga un paquete de un vehículo en una posición. En este caso se debe cumplir que el vehículo esté en la posición con la que se instancia el operador y que esté cargado con el paquete. Los efectos que provoca son el borrado del predicado **loaded** y la aserción del predicado **at** del paquete en esa posición.

2.2.3 *Mover_avión*

mover_avion ?a - avion ?o - aeropuerto ?d - aeropuerto ?co - ciudad

?cd - ciudad

El operador *mover_avión* tiene como objetivo el mover un avión desde el aeropuerto de una ciudad a otro aeropuerto de otra ciudad distinta. Para ello, necesitamos comprobar que el avión esté en un aeropuerto de una ciudad origen, y que el aeropuerto al que quiere viajar esté en una ciudad distinta a la ciudad origen. Esta comprobación se realiza viendo si el aeropuerto origen está en la ciudad origen y el aeropuerto destino en la ciudad destino. También se comprueba que estas ciudades sean distintas. Los efectos que provoca son que se borra el literal **at** del avión en el aeropuerto origen y se añade un literal **at** del avión en el aeropuerto destino.

2.2.4 *Mover_tren*

mover_tren ?t - tren ?o - estacion ?d - estacion

Este operador sirve para mover un tren entre dos estaciones. En este caso no hay que hacer ninguna comprobación especial, simplemente se comprueba que las estaciones sean distintas. Los efectos de este operador son el borrado del literal **at** del tren en la estación origen y la aserción del literal **at** del tren en la estación destino.

2.2.5 *Mover_furgoneta*

mover_furgoneta ?f - furgoneta ?c - conductor ?o - loc ?d - loc ?ci - ciudad

Este operador permite mover una furgoneta por una ciudad. Las furgonetas solo se pueden mover por dentro de las ciudades y además necesitan de un conductor. Por eso, las precondiciones son que el la furgoneta esté en la localización origen, que el conductor esté subido en la furgoneta y que las dos localizaciones sean de la misma ciudad. Los efectos son el borrado del literal **at** de la furgoneta en la posición inicial y la aserción del **at** de la furgoneta en la posición final.

2.2.6 *Mover_conductor*

mover_conductor ?c - conductor ?o - loc ?d - loc ?ci - ciudad

El operador *mover_conductor* sirve para mover un conductor (a pie) por una ciudad. En este caso comprobamos que exista el literal **at** del conductor en la posición inicial, ya que si no existe puede ser porque esté en otra posición o que esté subido en una furgoneta, por lo que no se podría mover. Adicionalmente se comprueba que las dos posiciones estén en la misma ciudad. Los efectos son similares a las acciones de mover: se elimina el literal **at** del conductor en la posición inicial y se añade el literal **at** del conductor en la posición final.

2.2.7 *Subir_conductor*

subir_conductor ?f - furgoneta ?c - conductor ?l - loc

Este operador permite subir un conductor a una furgoneta. Para ello, se comprueba que tanto el conductor como la furgoneta estén en la misma posición y que la furgoneta no tenga ningún conductor subido. Esto último se puede saber gracias al predicado **empty**. Este operador provoca que se elimine el literal **empty** de la furgoneta y el **at** del conductor en esa posición. También se añade un literal **on** del conductor en la furgoneta.

2.2.8 *Bajar_conductor*

bajar_conductor ?f - furgoneta ?c - conductor ?l - loc

El operador *bajar_conductor* permite bajar a un conductor de una furgoneta. Se comprueba que la furgoneta esté en la localización con la que se instancia en los parámetros del operador y que el conductor está subido en la furgoneta. Los efectos que provoca son el borrado del literal **on** que indicaba que el conductor estaba subido en la furgoneta, y las aserciones del literal **empty** con la furgoneta y **at** del conductor en esa localización.

2.3 Definición del problema

Una vez tenemos el dominio definido es el turno de definir el problema. Para este primer ejercicio se pedía crear un problema con la situación inicial mostrada en la figura 1, al que nos referiremos de ahora en adelante como problema base. Para ello hemos definido los objetos, los literales y los objetivos en el fichero del problema utilizando los predicados que hemos comentado anteriormente. En este caso, el objetivo es llevar los paquetes P1 y P2 a la casa Ca3.

2.4 Experimentación

Se han lanzado distintos experimentos para comprobar el funcionamiento de los tres planificadores. En concreto se ha ejecutado el planificador FF, LPG con la opción *-timesteps* y OPTIC. Se han comprobado los planes y se ha visto que todos resuelven el ejercicio. En la figura 2 podemos ver un ejemplo de plan sacado del planificador FF.

```
 0: MOVER_CONDUCTOR D1 E2 A1 CIUDAD1
 1: MOVER_CONDUCTOR D2 CA2 A2 CIUDAD2
 2: SUBIR_CONDUCTOR F2 D1 A1
 3: MOVER_FURGONETA F2 D1 A1 CA1 CIUDAD1
 4: LOAD F2 CA1 P1
 5: MOVER_FURGONETA F2 D1 CA1 E2 CIUDAD1
 6: UNLOAD F2 E2 P1
 7: LOAD T1 E2 P1
 8: MOVER_TREN T1 E2 E4
 9: SUBIR_CONDUCTOR F3 D2 A2
10: MOVER_FURGONETA F3 D2 A2 CA2 CIUDAD2
11: LOAD F3 CA2 P2
12: MOVER_FURGONETA F3 D2 CA2 E3 CIUDAD2
13: UNLOAD F3 E3 P2
14: SUBIR_CONDUCTOR F4 D3 CA3
15: UNLOAD T1 E4 P1
16: MOVER_TREN T2 E4 E3
17: LOAD T2 E3 P2
18: MOVER_TREN T2 E3 E4
19: UNLOAD T2 E4 P2
20: MOVER_FURGONETA F4 D3 CA3 E4 CIUDAD3
21: LOAD F4 E4 P1
22: LOAD F4 E4 P2
23: MOVER_FURGONETA F4 D3 E4 CA3 CIUDAD3
24: UNLOAD F4 CA3 P1
25: UNLOAD F4 CA3 P2
```

Figure 2: Ejemplo de plan generado por el planificador FF que resuelve el problema base.

Como podemos observar en la figura 6, el plan comienza moviendo a los conductores D1 y D2 a las localizaciones donde hay furgonetas. A continuación sube a un conductor y con esa furgoneta se va a la posición del paquete P1. Se carga el paquete en la furgoneta y se lleva a una estación, donde se descarga el paquete y se carga en un tren. Después el tren se mueve a la estación E4, que está en la ciudad donde hay que llevar el paquete. Para el otro paquete sucede lo mismo, pero con el otro conductor y en otra ciudad. Cuando los dos paquetes están en E4, se recogen con una furgoneta y se descargan al final en la posición final cumpliendo el objetivo.

Como observación, los planes que sacan no utilizan el operador *mover_avión* ya que no es necesario. No es necesario porque hay que llevar los paquetes a la ciudad 3, la cual no tiene aeropuerto, por lo que, como estamos intentando minimizar el número de acciones, los planificadores prefieren utilizar los trenes antes que los aviónes en este caso.

Planificador	**Acciones**	**Pasos**	**Tiempo (s)**
FF	25	25	<0.01
LPG	25	14	<0.01
LPG -timesteps	27	12	<0.01
OPTIC	30	12	0.08

Table 1: Resultados del dominio proposicional con el problema base.

Como se observa en la tabla 1, si miramos acciones el que menor número de acciones encuentra para un plan es FF, aunque el plan encontrado es totalmente secuencial. En el caso de LPG y Optic, se encuentran planes con más acciones que FF pero con menos pasos, con lo que al final son mejores planes al poder ejecutar acciones en paralelo. Dada esta observación, el planificador que mejor plan ha encontrado para el problema base ha sido LPG con la opción -*timesteps*, con un plan de 27 acciones en 12 pasos. En este caso empata con OPTIC, pero este último tiene mayor número de acciones y mayor tiempo de cómputo. Respecto al tiempo de cómputo, los tiempos son tan bajos que no llegan a mostrarse en algunos planificadores y se muestra 0.00, por lo que se ha indicado en la tabla que son valores menores a 0.01.

2.5 Otras instancias

A continuación, se detallan diferentes instancias de problema, cuya creación se pide en el enunciado:

- **Problema A**: Problema con solamente una ciudad. Esta instancia se ha utilizado para comprobar que los operadores funcionan tal y como se espera.
- **Problema B**: Mismo número de medios de transporte que el problema base, pero con 5 paquetes.
- **Problema C**: Mismos objetos que en el problema B pero diferente objetivo.
- **Problema D**: Problema con 3 trenes, 5 paquetes, 2 aviones, 3 furgonetas y 3 conductores.
- **Problema E**: Mismos objetos que el problema D pero distinto objetivo.

Gracias a estas instancias hemos comprobado que el dominio estaba bien definido, sobre todo con la instancia A, ya que al ser un problema pequeño hemos podido probar cada operador por separado.

Problema	**Planificador**	**Acciones**	**Pasos**	**Tiempo**
B	FF	44	44	<0.01
B	**LPG -timesteps**	44	**12**	2.64
B	OPTIC	45	13	0.14
C	FF	39	39	<0.01
C	LPG -timesteps	42	20	0.02
C	**OPTIC**	43	**13**	0.16
D	FF	44	44	<0.01
D	**LPG -timesteps**	43	**12**	0.06
D	OPTIC	48	12	0.16
E	FF	47	47	0.02
E	LPG -timesteps	47	20	0.02
E	**OPTIC**	44	**11**	0.20

Table 2: Resultados de las distintas instancias en el dominio proposicional.

En la tabla 2 se puede observar un resumen de los experimentos realizados con las nuevas instancias. En general, el planificador OPTIC nos ha conseguido mejores planes.

3 Dominio Temporal

En el segundo ejercicio se nos pide adaptar el problema para tener en cuenta el tiempo en nuestros predicados. La modificación tiene que cumplir con las siguientes restricciones

- **Conductores**: subir y bajar de un medio de transporte, 2 unidades de tiempo.
- **Paquetes**: definir un peso para cada paquete y un tiempo de carga/descarga proporcional al mismo.
- **Aviónes-Aeropuertos**: definir tiempo de recorrido de un avión entre aeropuertos.
- **Trenes-Estaciones**: definir tiempo de recorrido de un tren entre estaciones.
- **Distancia-Ciudades**: definir distancia de los caminos entre ciudades.
- **Conductores-Ciudades**: establecer tiempo de recorrido (caminando) proporcional a la distancia y en función de la velocidad a la que anda el conductor.
- **Furgonetas-Ciudades**: definir tiempo de recorrido de una furgoneta entre ciudades.

Para cumplir con todas estas restricciones hemos hecho distintos cambios al dominio y a los problemas. Se han definido los tiempos de manera dinámica (calculados en función de distancia y velocidad) y se ha intentado dar valores a las nuevas funciones para crear una representación realista del problema.

3.1 Modificaciones sobre el dominio proposicional

3.1.1 Funciones

Para la correcta implementación de las restricciones previamente mencionadas hemos tenido que añadir tres funciones numéricas.

```
(:functions
    (peso ?p - paquete)
    (distancia ?l1 - loc ?l2 - loc)
    (velocidad ?f - (either furgoneta conductor avi n tren))
)
```

Figure 3: Nuevas funciones añadidas, dominio temporal

Dada la distancia entre dos localizaciones y velocidad de vehículo podremos calcular la duración de movimientos de vehículos y con el peso podremos darle una duración proporcional al número de paquetes a las operaciones de **load unload**.

3.1.2 Operadores

Para el primer requisito hemos modificado los operadores de ***bajar_conductor*** y ***subir_conductor*** para incluir una duración fija de 2. Podemos ver un ejemplo de este cambio en *bajar_conductor*:

```
(:durative-action bajar_conductor
    :parameters (?f - furgoneta ?c - conductor ?l - loc)
    :duration (= ?duration 2)
    :condition (and
        (over all (at ?f ?l))
        (at start (on ?c ?f))
    )
    :effect (and
        (at end (not (on ?c ?f)))
        (at end (at ?c ?l))
        (at end (empty ?f))
    )
)
```

Figure 4: Operador bajar conductor, dominio temporal

El segundo requisito ha necesitado la función peso y una modificación a los operadores **load** y **unload**, podemos encontrar un ejemplo de load donde la duración se calcula en función del peso del paquete.

```
(:durative-action load
    :parameters (?v - vehiculo ?l - loc ?p - paquete)
    :duration (= ?duration (/ (peso ?p) 5)) ; Duracion igual a 1/5 del peso
    :condition (and
        (over all (at ?v ?l))
        (at start (at ?p ?l))
    )
    :effect (and
        (at start (not (at ?p ?l)))
        (at end (loaded ?v ?p))
    )
)
```

Figure 5: Operador load, dominio temporal

Para el resto de requisitos temporales — Furgonetas-ciudades — Trenes-Estaciones— Aviónes-Aeropuertos — Mover-conductor — hemos modificado todos los operadores de movimiento para avión, tren, furgoneta y conductor para contar con una duración calculada según la distancia a recorrer y la velocidad del sujeto. Podemos encontrar un ejemplo de *mover_avión*:

```
(:durative-action mover_avion
    :parameters (?a - avi n ?o - aeropuerto ?d - aeropuerto ?co - ciudad ?cd -
     ciudad)
    :duration (= ?duration (/ (distancia ?o ?d) (velocidad ?a)  ))
    :condition (and
        (at start (at ?a ?o))  ; El avion esta en el aeropuerto origen
        (over all (not (= ?co ?cd)))
        (over all (in ?co ?o)) ; Aeropuerto origen en ciudad origen
        (over all (in ?cd ?d)) ; Aeropuerto destino en ciudad destino
    )
    :effect (and
        (at start (not(at ?a ?o)))
        (at end (at ?a ?d))
    )
)
```

Figure 6: Operador mover avión, dominio temporal

Este cambio ha sido propagado al resto de funciones de movimiento.

3.2 Cambios en los problemas

Para los problemas hemos tenido que definir valores para todas nuestras funciones nuevas:

```
; Temporal
(= (peso P1) 10) (= (peso P2) 15) (= (peso P3) 20)
(= (peso P4) 10) (= (peso P5) 15) (= (peso P6) 20)
(= (peso P7) 10) (= (peso P8) 15) (= (peso P9) 20)
(= (peso P10) 20)


; Ciudad 1
(= (distancia Ca1 E1) 50) (= (distancia E1 Ca1) 50)
(= (distancia Ca1 E2) 120) (= (distancia E2 Ca1) 120)
(= (distancia Ca1 A1) 50) (= (distancia A1 Ca1) 50)
(= (distancia A1 E1) 100) (= (distancia E1 A1) 100)
(= (distancia A1 E2) 60) (= (distancia E2 A1) 60)
(= (distancia E2 E1) 50) (= (distancia E1 E2) 50)


; Ciudad 2
(= (distancia A2 E3) 150) (= (distancia E3 A2) 150)
(= (distancia A2 Ca2) 80) (= (distancia Ca2 A2) 80)
(= (distancia Ca2 E3) 150) (= (distancia E3 Ca2) 150)

; Ciudad 3
(= (distancia Ca3 E4) 70) (= (distancia E4 Ca3) 70)

; Velocidades
(= (velocidad F1) 50) (= (velocidad F2) 50)
(= (velocidad F3) 50) (= (velocidad F4) 50)
(= (velocidad D1) 10) (= (velocidad D2) 10)
(= (velocidad D3) 10)

(= (velocidad Av1) 300)
(= (velocidad Av2) 300)
(= (velocidad T1) 100)
(= (velocidad T2) 100)
```

Figure 7: Valores asignados para funciones nuevas en problema temporal

Con estos valores conseguimos poder ejecutar las acciones previamente mencionadas. Se ha decidido crear nuevas instancias distintas a las que se han creado en la sección anterior para poder tener experimentos con distinta talla y así poder realizar una mejor evaluación comparativa. Se han creado cuatro problemas nuevos basados en el base pero con distinto número de paquetes,

concretamente 10, 20, 30 y 40 paquetes. El nombre de los mismos es problema 1, problema 2, problema 3 y problema 4, respectivamente.

3.3 Experimentación

Se han lanzado varios experimentos con los planificadores LPG y OPTIC intentando conseguir el mejor plan posible para las cuatro instancias de problemas que hemos creado minimizando el tiempo. En la figura 8 podemos ver un resumen de las duraciones de los planes, los pasos y el tiempo de cómputo obtenidos en experimentos. Se han puesto los datos solamente de los mejores planes que se han conseguido para cada problema.

Problema	LPG			OPTIC		
	Duración	Pasos	Tiempo Cómputo	Duración	Pasos	Tiempo Cómputo
Base	**36.50**	27	0.02	40.91	31	0.16
1	**41.60**	81	0.34	47.60	75	2.06
2	64.20	127	3.24	**47.60**	111	5.14
3	92.50	176	32.18	**56.90**	167	13.88
4	105.90	230	63.20	**56.90**	203	22.76

Figure 8: Resultados experimentación dominio temporal.

Como podemos ver en la tabla, para el problema base y el problema 1, que son los más pequeños, se obtiene una duración menor con el planificador LPG, aunque las duraciones no distan mucho de las obtenidas con OPTIC. Esto ocurre también con el número de pasos y el tiempo de cómputo. En cambio, si probamos con problemas más grandes, como el 2, 3 y 4, es OPTIC el planificador que consigue mejores tiempos de plan. En concreto, podemos observar una gran diferencia en las duraciones del plan 4, con el que OPTIC consigue un tiempo de aproximadamente la mitad del tiempo que consigue LPG. Respecto al tiempo de cómputo y el número de pasos, se empieza a ver un una diferencia notable entre ambos planificadores al experimentar con estos tres últimos problemas.

Respecto a los resultados obtenidos con OPTIC, podemos observar una peculiaridad en las duraciones. En el caso de los problemas 1 y 2 la duración es la misma. También ocurre con los problemas 3 y 4. Pensamos que esto puede ser debido a la paralelización que hace este planificador, ya que las instancias del problema se han creado añadiendo paquetes a las mismas posiciones que tenían paquetes. De esta manera cada posición del mapa con paquetes tiene el mismo número de paquetes que las demás posiciones con paquetes. Además, los paquetes de una misma posición inicial hay que llevarlos a la misma posición final. Esto hace que el planificador paralelice las acciones de manera que se carguen todos los paquetes juntos, pudiendo obtener una misma duración en problemas de talla distinta.

4 Dominio con recursos numéricos

Para este tercer ejercicio hemos añadido una variable recurso a nuestros vehículos, el combustible. Con esta variable podemos modelar el consumo de cada vehículo y crear una versión del dominio mas realista. Como indica el boletín esta variable es inversamente proporcional al consumo del tiempo definido en el ejercicio segundo, cuanto menos tiempo requiere un movimiento de un vehículo más combustible gasta.

Se han creado dos versiones del dominio, una en la que se puede recargar el recurso y otra en la que no. En nuestros experimentos evaluaremos los problemas para cada dominios minimizando tanto tiempo como combustible usado. Los problemas ha evaluar han sido el base y problema 1, intentamos los problemas 2,3,4 pero no conseguimos resolverlos con los planificadores.

4.1 Combustible como recurso

Nuestro recurso es una variable numérica que representa el valor de combustible actual de cada vehículo. Este combustible debe disminuir a medida que un vehículo realiza viajes y poder ser repostado (solo en uno de nuestros dos dominios), para ello se requieren distintas modificaciones a nuestros dominios y problemas. Hemos realizado distintas modificaciones con el fin de alcanzar una representación realista del combustible en los vehículos.

4.2 Modificaciones sobre el dominio temporal

4.2.1 Funciones adicionales

Para ser capaces de modelar el combustible hemos añadido varias funciones a cada uno de nuestros vehículos:

- **Consumo**: Cuanto combustible consume el vehículo en una unidad de tiempo
- **Combustible**: Combustible actual del vehículo
- **Capacidad**: Capacidad máxima de almacenaje de combustible para el vehículo, esta variable nos permite añadir la opción de recargar combustible

Con estas tres funciones somos capaces de realizar una representación mas real de como funciona un recurso como el combustible en la vida real. Veremos como se usan en los siguientes cambios

4.2.2 Operadores

En cuanto a los operadores, hemos añadido restricciones para que un vehículo solo pueda moverse entre dos puntos si tiene combustible suficiente. También añadimos efectos extra a estas acciones de movimiento, reducimos el combustible

del vehículo y aumentamos una variable global de combustible usado según la cantidad usada. Calculamos el combustible requerido dada la velocidad del vehículo, distancia entre origen-destino y gasto del vehículo. Podemos ver un ejemplo de operador para mover un avión:

```
(:durative-action mover_avion
    :parameters (?a - avion ?o - aeropuerto ?d - aeropuerto ?co - ciudad ?cd -
  ciudad)
    :duration (= ?duration (/ (distancia ?o ?d) (velocidad ?a)))
    :condition (and
        (at start (at ?a ?o)) ; El avion esta en el aeropuerto origen
        (over all (not (= ?co ?cd)))
        (over all (in ?co ?o)) ; Aeropuerto origen en ciudad origen
        (over all (in ?cd ?d)) ; Aeropuerto destino en ciudad destino
        (at start (>= (combustible ?a) (* (/ 1 (/ (distancia ?o ?d) (velocidad
  ?a))) (gasto ?a)))) ; Tenemos suficiente combustible en el avion
    )
    :effect (and
        (at start (not(at ?a ?o)))
        (at end (at ?a ?d))
        (at end (increase
                combustibletotal
                (* (/ 1 (/ (distancia ?o ?d) (velocidad ?a))) (gasto ?a)))) ;
 Aumentamos la variable de combustibel total
        (at end (decrease
                (combustible ?a)
                (* (/ 1 (/ (distancia ?o ?d) (velocidad ?a))) (gasto ?a)))) ;
 Restamos el combustible usado al avion
    )
)
```

Figure 9: Operador mover-avión.

En el caso del dominio con recarga de combustible hemos añadido un operador extra:

```
(:durative-action refuel-avi n
    :parameters (?a - avi n ?o - aeropuerto ?c - ciudad)
    :duration (= ?duration 0.3)
    :condition (and (at start ( > (capacidad ?a) (combustible ?a)))
        (over all (at ?a ?o)) ; El avi n esta en el aeropuerto origen
        (over all (in ?c ?o))
    )
    :effect (at end (assign (combustible ?a) (capacidad ?a)))
)
```

Figure 10: Operador refuel-avión.

4.3 Cambios en los problemas

Para los problemas hemos tenido que definir valores para todas nuestras funciones nuevas:

```
;Combustible
(= (combustibletotal) 0)

;Aviones
(= (combustible Av1) 20)
(= (combustible Av2) 10)
;Trenes
(= (combustible T1) 40)
(= (combustible T2) 20)
;Furgonetas
(= (combustible F1) 75)
(= (combustible F2) 75)
(= (combustible F3) 75)
(= (combustible F4) 75)

;Capacidad
(= (capacidad Av1) 1000)
(= (capacidad Av2) 1000)
(= (capacidad T1) 500)
(= (capacidad T2) 500)
(= (capacidad F1) 250)
(= (capacidad F2) 250)
(= (capacidad F3) 250)
(= (capacidad F4) 250)

; ;Gasto
(= (gasto Av1) 5)
(= (gasto Av2) 5)
(= (gasto T1) 10)
(= (gasto T2) 10)
(= (gasto F1) 14)
(= (gasto F2) 14)
(= (gasto F3) 14)
(= (gasto F4) 14)
```

Figure 11: Valores definidos para las funciones nuevas.

Como hemos implementado el combustible para todos los vehículos se ha tenido que definir una gran cantidad de valores. Esto también nos ha generado problemas que veremos más en adelante ya que al ser tantos valores nos fue bastante complicado encontrar una buena combinatoria que permitiera a los problemas resolverse tanto sin recargar como recargando pero que si hacia uso de la recarga fuera beneficioso.

4.4 Experimentación

Hemos realizado ocho experimentos distintos, 2 problemas (base y problema 1) con dos versiones cada uno (minimizando tiempo o combustible) probados en todos los dominios (con recarga de combustible o sin ella). Los resultados son los siguientes:

			LPG			OPTIC		
Problema	Dominio	A Minimizar	Resultado métrica	Pasos	Tiempo Cómputo	Resultado métrica	Pasos	Tiempo Cómputo
Base	Combustible	Tiempo	**34.6**	30	0.02	39.00	30	0.22
Base	Combustible	Combustible	**88.28**	25	0.02	165.33	30	0.22
Base	Combustible-recarga	Tiempo	**34.6**	27	0.02	39.00	30	0.22
Base	Combustible-recarga	Combustible	**88.28**	25	0.02	165.33	30	0.22
1	Combustible	Tiempo	**50.40**	86	12.08	73.00	90	124.68
1	Combustible	Combustible	**124.77**	74	0.82	285.08	90	100.42
1	Combustible-recarga	Tiempo	**39.90**	79	9.42	45.20	80	25.16
1	Combustible-recarga	Combustible	**126.20**	73	9.06	233.83	80	33.20

Figure 12: Resultados experimentación dominio numérico

Encontramos mejores resultados en la métrica a minimizar con el planificador LPG en todos los casos para problema base y 1 ya sea con dominio sin recargar o minimizando tiempo. Creemos que la victoria de LPG sobre OPTIC se debe mayoritariamente al tratarse de problemas de talla pequeña, algo que hemos visto también suceder en los primeros experimentos del dominio temporal. El mayor valor de esta tabla debería ser como en para el problema 1 tanto LPG como OPTIC consiguen mejorar su plan temporal gracias al uso de dominio con recarga de combustible, LPG consigue bajar de 50 a 39.90 unidades de tiempo y LPG baja de 73 a 45.20. Esto se debe a que encuentran planes donde recargan el combustible de los vehículos con mas gasto y velocidad.

5 Desarrollo parcial de un árbol POP

En esta sección se desarrolla un plan de orden parcial. Se ha escogido un solo objetivo a partir del problema base que se ha visto en el dominio proposicional. En este caso, se ha elegido el objetivo **(at P1 Ca3)**. Para la selección del *flaw* en cada nodo del árbol, se ha optado por una elección aleatoria. A continuación, se detalla el desarrollo del árbol.

Partimos del plan 1, donde solo tenemos el estado inicial y final. En este plan podemos ver el objetivo que hemos propuesto encima del estado final. Lo podemos observar en la figura 13.

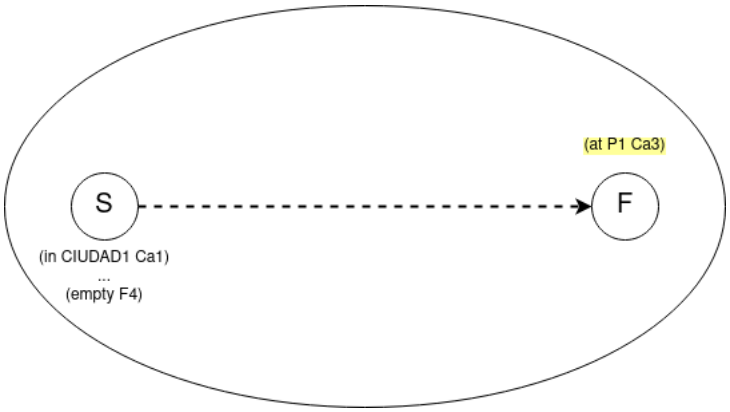

Figure 13: Plan 1 del árbol POP.

A partir de este primer plan, se decide resolver el literal del objetivo final, que se resuelve con la acción de *unload*, la cual descarga el paquete en Ca3. En la figura 14 podemos ver el estado del árbol al que se llega.

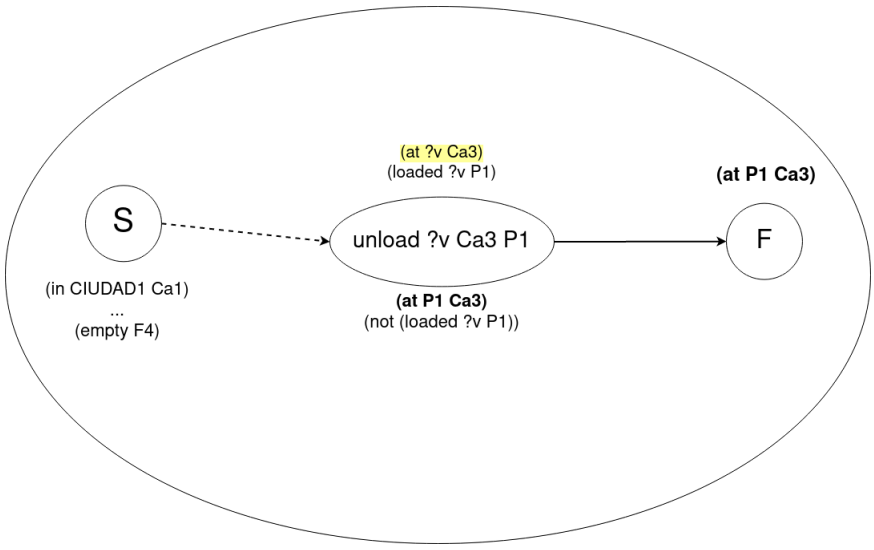

Figure 14: Plan 2 del árbol POP.

Una vez estamos en el plan 2 del árbol POP, tenemos tres flaws: los literales **(at ?v Ca3)** y **(loaded ?v P1)**, y la instanciación de la variable **?v**. En este caso se ha decidido resolver el literal **(at ?v Ca3)**. Este literal se resuelve con acciones de mover algún vehículo de una localización a Ca3. Como el único vehículo que se puede mover a una Casa es una furgoneta, este literal solo puede resolverse con la acción *mover_furgoneta*. De este modo, se llega al siguiente

nodo del árbol, que podemos observar en la figura 15.

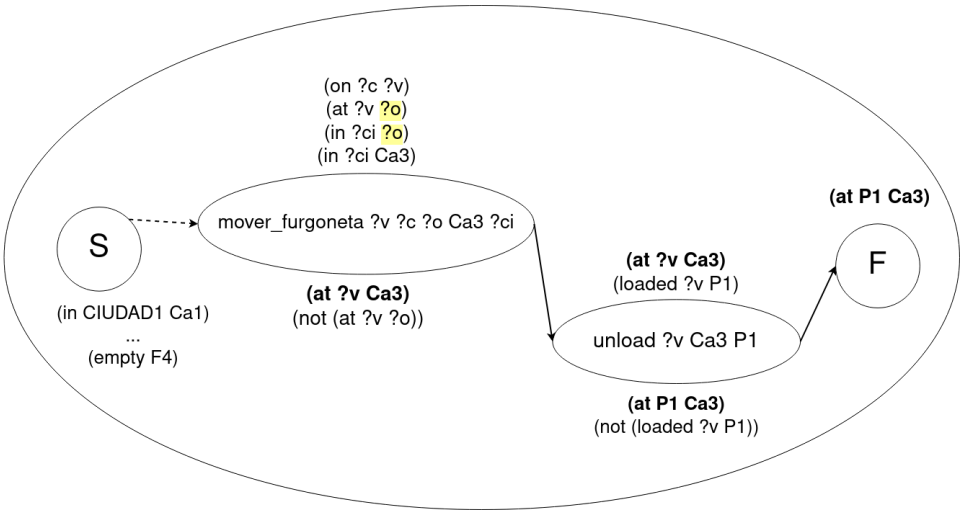

Figure 15: Plan 3 del árbol POP.

Una vez estamos en el plan 3, se añaden bastantes *flaws*. Como objetivos abiertos tendríamos los literales **(on ?c ?v)**, **(at ?v ?o)**, **(in ?ci ?o)**, **(in ?ci Ca3)** y **(loaded ?v P1)**. También tendríamos como variables a instanciar las siguientes: **?c ?v ?o** y **?ci**. En este caso se ha decidido instanciar la variable **?o**, la cual ha instanciarse con una localización. Este *flaw* solo puede resolverse de una manera, ya que solo existe una localización conectada con la casa Ca3, y es la estación E4, por lo que la variable **?o** debe instanciarse con E4. De este modo, el árbol de planificación se expandiría un nodo, que podemos ver en la siguiente figura.

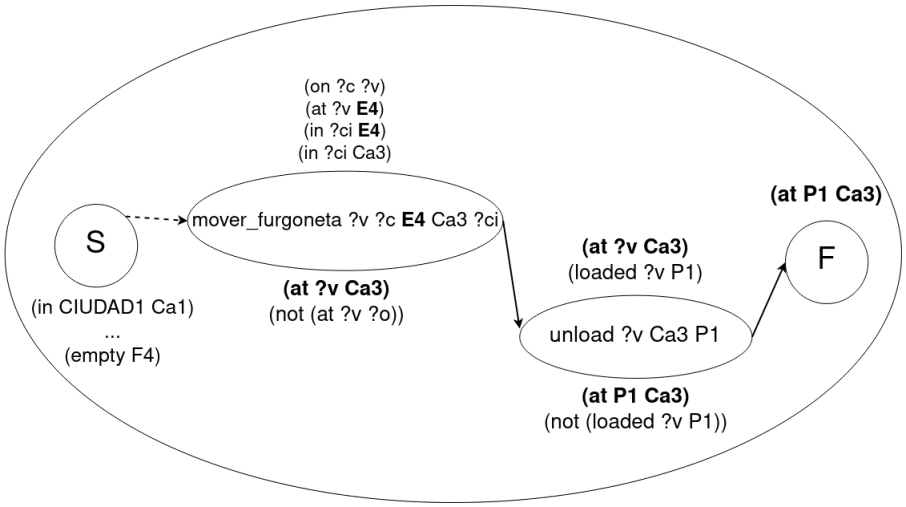

Figure 16: Plan 4 del árbol POP.

Como hemos visto, nuestros planes dentro del árbol se expanden solamente con un nodo, ya que solo hay una forma posible de resolver cada *flaw*. Esto se debe a que la ciudad 3 a la que hay que llevar un paquete solamente se puede acceder con una furgoneta y desde una única localización, por lo que para resolver el objetivo los últimos pasos tienden a ser secuenciales. Probablemente, empezarán a haber amenazas cuando haya que mover vehículos entre distintas ciudades.

En la siguiente figura podemos ver un resumen de los nodos del árbol POP que hemos generado. Se puede observar que no ramifica en ningún momento. Esto ocurre porque no hay más de una forma de resolver los *flaws* que se han elegido.

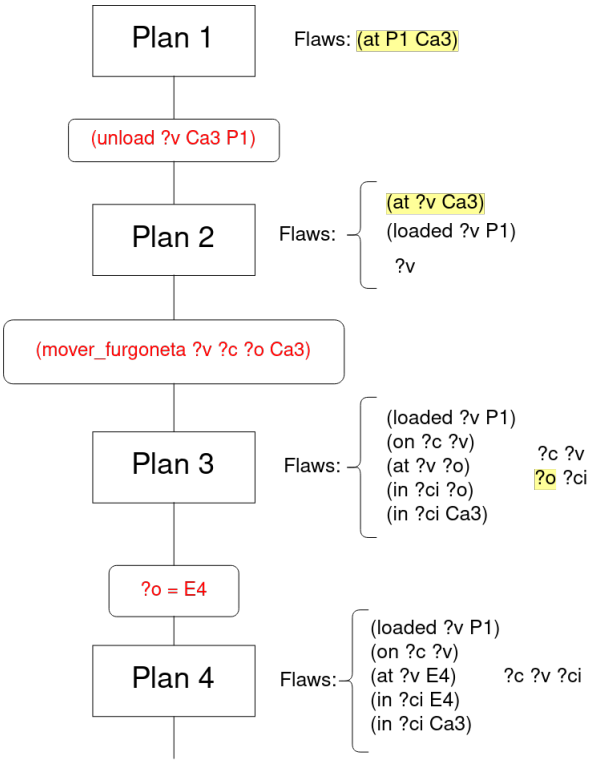

Figure 17: Árbol POP generado.

6 Graphplan

En esta sección se desarrolla un grafo de planificación relajado. Este grafo sirve para calcular unas heurísticas que den información al planificador para que pueda dirigir la búsqueda, ya que a veces el factor de ramificación es muy alto en este tipo de búsquedas. Estas heurísticas son muy fáciles de extraer ya que es un grafo multietapa donde el número de capas hasta llegar al objetivo es un límite inferior del número de acciones del plan.

Se ha calculado un grafo de planificación para resolver el problema base mencionado en el apartado del dominio proposicional. Una vez desarrollado el grafo, el cual se puede encontrar adjunto en un fichero en la entrega de este trabajo, se muestran a continuación el cálculo de las heurísticas h_max y h_sum.

- $h\_max(G) = max(9, 9) = 9$
- $h\_sum(G) = 9 + 9 = 18$

A continuación, se muestra el plan relajado que se ha extraído del grafo.

```
Nivel 1: mover_conductor(D1 E2 E1)
         mover_conductor(D2 Ca2 A2)
Nivel 2: subir_conductor(F3 D2 A2)
         subir_conductor(F1 D1 E1)
Nivel 3: mover_furgoneta(F3 D2 A2 Ca2)
         mover_furgoneta(F1 D1 E1 Ca1)
Nivel 4: load(F3 Ca2 P2)
         load(F1 Ca1 P1)
Nivel 5: (unload F3 E3 P2)
         (unload F1 E2 P1)
Nivel 6: (load T2 E3 P2)
         (load T1 E2 P1)
Nivel 7: (unload T1 E4 P1)
         (unload T2 E4 P2)
Nivel 8: (load F4 E4 P1)
         (load F4 E4 P2)
Nivel 9: (unload F4 Ca3 P1)
         (unload F4 Ca3 P2)
```

Figure 18: Plan relajado extraído del grafo de planificación relajado.

Dado el plan relajado, su heurística quedaría de la siguiente manera:

- $plan\_relajado(G) = 18$

Este plan se puede extraer en tiempo polinómico y sin necesidad de hacer *backtracking* ya que el grafo de planificación que hemos calculado es relajado, por lo que no tiene efectos negados y no hay que calcular mutex. Una vez se consigue un literal por primera vez se cumple para siempre, y la extracción del plan es directa.

La heurística que consideramos más informada para nuestro problema es h_max, ya que tenemos dos acciones en cada paso, muchas de las cuales se podrían paralelizar también en un problema no relajado, al ser acciones en ciudades distintas que no se anulan sus precondiciones recíprocamente. Además, esta heurística es admisible, ya que no sobrestima el coste del plan. De hecho, para este problema con el planificador LPG hemos obtenido un total de 12 pasos, lo cual se acerca al valor de 9 obtenido con esta heurística y está más lejos de los valores obtenidos con las demás.

7 Conclusiones

En primer lugar, se ha realizado una codificación PDDL del problema. Esta parte no ha resultado demasiado dificultosa, ya que teníamos una ligera experiencia con este lenguaje y el dominio proposicional no nos ha resultado compli-

cado de codificar. Los dominios temporal y con recursos si que han necesitado más esfuerzo, al no tener demasiada experiencia.

Por la parte de experimentación, hemos aplicado los conceptos aprendidos durante las clases de teoría para poder entender en cierta manera el funcionamiento de los distintos planificadores. En esta parte experimental ha sido donde más hemos aprendido, ya que hemos tenido que modificar varias veces el código por dificultades con algún planificador.

En segundo lugar, los ejercicios del árbol de estados POP y el *Graphplan* nos han hecho aplicar conceptos explicados en clase y nos han ayudado a entender el mundo de los planificadores, ya que nos han dado una imagen de cómo funcionan por dentro a la hora de planificar.

Finalmente, la realización de este trabajo nos ha dado una buena base en cuanto al tema de la planificación inteligente. Todo el proceso de desarrollo y experimentación nos han ido dando conocimiento y experiencia para llegar a entender cómo se trabaja para abordar tareas de planificación inteligente.