INTELIGENCIA ARTIFICIAL
PRÁCTICA 3: PLANIFICACIÓN

Miguel Ángel Merino
Pol Renau
Carla Lara

11-6-2018
2017/2018 - Q2

Índice general

Capítulo 1

Introducción

En esta práctica desarrollada durante las últimas dos semanas del cuatrimestre se nos plantea un problema a resolver mediante el desarrollo de agentes Planificadores con FastForward. Este es un planificador que permite ejecutar planes definidos en lenguaje PDDL.

Para ello se dividen los problemas de PDDL en dos conjuntos:

- **Definición del dominio:** En este encontramos presentes los operadores, predicados, tipos,... cuáles serán los encargados de hallar la solución a partir de unos hechos o instancias iniciales. Se interpreta como las diferentes acciones que se pueden realizar en un entorno concreto.
- **Definición de una instancia del problema:** En ella se encuentran presentes las diversas constantes y hechos que se usarán para llegar a una solución definida en la misma instancia como objetivo a alcanzar.

Esta técnica no intenta encontrar la solución más óptima a un problema dado, sino que pretende encontrar una buena solución en relación al tiempo y recursos a invertir. Además permiten secuenciar unas acciones a efectos de llegar a un objetivo, lo que los convierte en un tipo de problema de síntesis.

Otra característica importante de los planificadores es que pueden separar en *Dominio* y *Problema*. Esto permite que se puedan resolver numerosos problemas distintos sin la necesidad de cambiar el dominio. El resultado final es la generación de un plan que alcanza el objetivo definido en el problema.

Capítulo 2

Descripción del problema

El problema a tratar en esta práctica consiste en generar planes de viajes de la agencia *Al fin del mundo y más allá*, mediante restricciones impuestas por el usuario y otras definidas en la base de datos. Para ello disponemos de un conjunto de ciudades considerablemente grande. Supondremos siempre que partimos de la ciudad de origen y el viaje no se acaba hasta que se llega a este. Además, en un mismo viaje, no se podrán visitar dos ciudades iguales, es decir, no se podrá volver a una ciudad por la que ya has pasado.

Cada una de las ciudades contiene cierta información relevante para la planificación del viaje:

- **Alojamientos:** Cantidad de diversos alojamientos con diversos precios distintos disponibles en la ciudad a visitar. Actualmente se identifican con un número que va del 1 a la cantidad de alojamientos de esa ciudad.
- **Identificador:** Identifica la ciudad mediante un valor entre 0 i la cantidad de ciudades a visitar + 1. La ciudad 0 es la origen.
- **Mínimo de días por ciudad:** Cantidad de días mínimos asignados a la ciudad en concreto.
- **Máximo de días por ciudad:** Cantidad de días máximos asignados a la ciudad en concreto.
- **Interés:** Valor que oscila entre el 1 y el 3 siendo el primero el que indica mayor interés por la ciudad y el último el que indica menor interés.

Además de estos rasgos que poseen las ciudades, hay otros rasgos generales importantes para la resolución de las diversas extensiones propuestas:

- **Vuelos:** Vuelo que va de una ciudad origen a la ciudad destino. Se identifican por ambas el conjunto de los identificadores de las distintas ciudades a visitar. Cada vuelo posee un precio en concreto.
- **Precio total:** Precio calculado de forma dinámica a medida que se van asignando ciudades, hoteles, y vuelos al viaje.

- **Interés total:** Suma de los intereses de las ciudades por las que se planifica el viaje.
- **Mínimo de días por viaje:** Cantidad de días mínimos que quiere pasar el usuario de viaje.
- **Mínimo de ciudades a visitar:** Cantidad de ciudades mínimas por las que quiere pasar el usuario a lo largo de su viaje.
- **Días totales:** Suma dinámicamente los días que se invierten en el viaje, estando este número entre el mínimo y el máximo definidos anteriormente.
- **Ciudad Origen:** Definida con el identificador 0, es la que se define como la ciudad en la cual se empieza el viaje y se finaliza.

Una vez definidos y decididos todos estos datos se nos pide asignar diversos vuelos a diversas ciudades pasando la noche en un alojamiento, en función de un conjunto de criterios iniciales a los cuales se les irán añadiendo más especificaciones a lo largo de la práctica.

Para el **Caso Base** simplemente se pide que, dado un número mínimo de ciudades a visitar, se planifique un viaje en el cual se asigne un vuelo para viajar entre ciudades y un hotel por cada ciudad, cumpliendo el número mínimo de ciudades establecido.

Para la **Primera extensión** se nos pide ampliar el caso base añadiendo una nueva restricción, número mínimo y máximo de días por ciudad. Dado esto y la cantidad de días mínimos definido en el caso anterior, se quiere planificar un viaje que cumpla esas condiciones y nos diga cuantos días se pasa en cada ciudad.

Para la **Segunda extensión** se nos pide ampliar la primera extensión añadiendo el interés por ciudad. La nueva restricción es que se maximice éste interés. El objetivo es la planificación de un viaje donde quede cumplida esta restricción junto con la de los apartados anteriores.

Para la **Tercera extensión** se nos pide ampliar la primera extensión añadiendo precios a los vuelos y a los alojamientos. La nueva restricción es minimizar el precio total del viaje pero que no salga de los márgenes de precio mínimo y máximo que se deben definir. El objetivo es la planificación de un viaje donde se cumplan todas las condiciones.

Para la **Cuarta extensión** se nos pide ampliar la primera extensión junto con la mezcla de las extensiones dos y tres. En este caso se deben optimizar el interés de las ciudades visitadas y el precio total del recorrido utilizando ponderaciones.

Capítulo 3

Dominio

Con el objetivo de exponer las distintas características del dominio de manera clara y adecuada, así como explicar la necesidad de los predicados existentes, analizaremos extensión por extensión para ir entendiéndolo de manera progresiva y no repetir explicaciones iguales.

3.1. Nivel Básico

Dado que en éste caso lo que queremos es indicar el hotel en que alojarse y los vuelos que tomar dado un conjunto de ciudades disponibles, hoteles correspondientes a cada ciudad y vuelos entre las mismas, el dominio estará definido tal y como se explica a continuación.

3.1.1. Objetos y estado inicial

En esta primera sección se representarán las cosas del mundo que nos interesan para resolver el problema y el estado en el que se encuentran.

Como etiquetas simplemente necesitaremos un conjunto de **ciudades**, **alojamientos** y **vuelos**, donde de alguna manera deberemos indicar qué ciudad es la de partida y destino (en nuestro caso **Co**) para posteriormente poder usarla en la sección de objetivos del planificador. En la definición de las etiquetas usaremos **Types** para poder asociar las etiquetas a los tipos anteriormente mencionados y así ahorrarnos predicados y comprobación de los mismos en la precondición de las acciones.

En la definición del estado inicial estableceremos las siguientes pautas para representar el estado en el que nos encontramos inicialmente:

- Definiremos las etiquetas del apartado de Objetos de tipo ciudad a dicho tipo.
- Procedimiento análogo al anterior para las etiquetas correspondientes a alojamientos.

- Proceso también análogo a los anteriores para definir las etiquetas correspondientes a tipo vuelo.
- Definiremos **Allotjamentde** cómo la asociación entre una alojamiento y una ciudad que usaremos para representar que un determinado alojamiento se encuentra en una determinada ciudad.
- También definiremos de una manera similar **Vuelode** asociando un vuelo con dos ciudades para representar el origen y destino de un vuelo.
- Usaremos los fluentes para definir **visitades** que contendrá el número de ciudades visitadas que lógicamente en el estado inicial es 0.
- De la misma manera, definiremos **minim-ciu** como el número mínimo de ciudades a visitar (tal y como nos pide el enunciado) y que marcaremos al valor que nos interese dado el conjunto de objetos de un problema.
- Finalmente definiremos **en** que nos servirá para establecer la ciudad en la que el planificador se encuentra en cada momento, siendo lógicamente en el estado inicial la ciudad de origen la usada.

3.1.2. Objetivo

Esta sección nos servirá para establecer las cosas que queremos que sean ciertas para finalizar el programa. En este caso, para solucionar el problema de nivel básico y en base a los objetos y estado inicial definidos anteriormente, lo que querremos será que fundamentalmente se cumplan dos cosas:

- Que el número de ciudades visitadas (visitades) sea mayor o igual al número mínimo de ciudades a visitar (minim-ciu).
- Que la ciudad en la que nos encontremos sea la misma ciudad de origen, ya que significará que habremos viajado por las ciudades y habremos vuelto al origen.

3.1.3. Funciones

En esta sección se definirán aquellas variables a las que les daremos un uso numérico mediante el uso de fluentes, de manera que no serán simples etiquetas como las demás. Ésto es lo que nos permite darle valores numéricos a las variables del estado inicial, cómo hemos hecho anteriormente. En este primer nivel básico, lo que situaremos en este apartado de funciones será:

- Desearemos que **visitades** pueda almacenar un número que será la cantidad de ciudades visitadas.
- Desearemos también que **minim-ciu** pueda ser un valor numérico que como ya hemos dicho antes será el número mínimo de ciudades a visitar.

3.1.4. Predicados

Para finalizar, en esta última sección expondremos los predicados, que serán las distintas propiedades que querremos que los objetos cumplan. En ellas tendremos:

- Que la ciudad en la que nos encontramos a cada momento (en) sea un objeto ciudad.
- Que la asociación de un vuelo a su origen y destino (Vuelode) relacione un objeto de tipo vuelo con dos de tipo ciudad.
- Que la asociación de un alojamiento a su ciudad (Allotjamentde) relacione un objeto de tipo alojamiento con otro de tipo ciudad.
- Que (Allotjaten) sea un objeto de tipo ciudad que nos servirá para representar el hecho de que ya se haya alojado en dicha ciudad.

3.2. Extensión 1

El incremento funcional que comporta esta primera extensión será indicar el número mínimo y máximo de días a estar en las ciudades, así como el mínimo número de días que tendrá que tener el recorrido completo. En base a ello, partiendo ya de lo que se ha expuesto para el nivel básico, se añadirá al dominio lo siguiente:

3.2.1. Objetos y estado inicial

Dado que lo que se añade son principalmente restricciones numéricas a un entorno ya existente, no se añadirán etiquetas nuevas ya que no necesitaremos más entidades que formen parte del entorno.

Sin embargo, en la definición del estado inicial sí que se añadirán distintos elementos que nos permitirán representar las restricciones numéricas, dichos elementos los siguientes *fluentes*:

- Definiremos los días que se pasan en una ciudad, para cada ciudad, con **dies-perciutat** que lógicamente inicialmente será 0 para todas.
- Definiremos también el máximo y el mínimo de días a estar en una ciudad con **minim-d-c** y **maxim-d-c** y les asociaremos el valor que queramos para ellos en esa ejecución.
- También definiremos el mínimo total de días que durará el recorrido entero del viaje con **minim-d** y al cual le pondremos el valor que queramos para esa ejecución también.

- Finalmente definiremos el número de días que llevamos de días que llevamos de viaje con **diastotal** y que lógicamente en el estado inicial deberá ser 0.

3.2.2. Objetivo

El objetivo que se deberá cumplir será igual al del nivel básico simplemente añadiendo que el número de días total que dura el viaje (diastotal) deberá ser mayor o igual que el mínimo de días que debe durar el viaje (minim-d).

3.2.3. Funciones

En éste caso, viendo que todos los elementos que hemos definido nuevos en el estado inicial son usan fluentes, necesitaremos declarar todas esas 5 nuevas variables en ésta función para que el planificador entienda que van a ser valores numéricos y no simples etiquetas como los demás.

3.2.4. Predicados

Como ya hemos explicado anteriormente, lo único que debemos hacer es tratar las restricciones numéricas, por lo que no estamos añadiendo ni modificando relaciones entre objetos y consecuentemente no se necesitarán añadir predicados nuevos respecto a los existentes en la versión del nivel básico.

3.3. Extensión 2

Como se ha explicado en la descripción del problema, esta segunda extensión consistirá en añadir interés a las ciudades y tratar de maximizarlo. En base a ello, los cambios realizados respecto a la extensión anterior serán los descritos a continuación.

3.3.1. Objetos y estado inicial

Dado que lo que queremos hacer es añadir un determinado interés a las ciudades, los objetos del entorno seguirán siendo los mismos por lo que las etiquetas definidas en la sección de objetos permanecerán iguales.

En relación a la definición del estado inicial, a parte de lo hecho hasta ahora, se deberán añadir unas nuevas definiciones para poder tratar el interés de las ciudades, las nuevas definiciones serán definidas mediante el uso de fluentes de la siguiente manera:

- Definiremos el interés total del viaje acumulado hasta el momento con **interestotal** que como es lógico inicializamos a 0 dado que en el estado inicial del problema no se ha visitado ninguna ciudad y consecuentemente no se ha acumulado interés.
- Definiremos también el interés asociado a cada ciudad mediante **interes** , asociando un valor a cada ciudad correspondiente.

3.3.2. Objetivo

Dado que los objetivos como tal no varían en relación a los de la extensión previa, no será necesario añadir nada nuevo a esta sección porque simplemente el planificador deberá maximizar el interés pero sin tener ningún interés específico ni ningún objetivo como tal.

3.3.3. Funciones

De manera análoga a lo realizado en ésta sección con la extensión 1, solo tendremos que declarar las dos variables que hemos explicitado como nuevas en el anterior apartado del estado inicial.

3.3.4. Predicados

En este caso tampoco estamos añadiendo ni modificando relaciones entre objetos por lo que tampoco será necesario añadir nuevos predicados en esta sección, restando simplemente los que teníamos en el caso anterior (recordar que se siguen explicitando los Types para ahorrar declaración de predicados y comprobación futura de los mismos).

3.4. Extensión 3

En esta extensión cabe remarcar que no partimos de la extensión anterior sinó que lo hacemos de la extensión número uno. Lo que ampliaremos será que incluiremos precios tanto para los vuelos como para los hoteles y pretendemos minimizar el precio total del viaje.

3.4.1. Predicados

Los objetos del entorno van a seguir siendo los mismos dado que tan solo añadimos precios a los que son de tipo alojamiento o vuelo, por lo que consecuentemente no hará falta añadir nuevos tipos de objetos.

En relación al estado inicial, sí que tendremos que añadir dos tres definiciones para poder definir el precio de un hotel, el precio de un vuelo y finalmente el precio total. Las definiciones exactamente serán las siguientes:

- Definiremos el precio total del viaje mediante **preutotal** que lógicamente inicialmente nos encontraremos que vale 0.
- Tendremos que definir también el precio de cada vuelo mediante **preuVol**, cada precio lo inicializamos a algún valor que sea coherente. Finalmente definiremos el precio de cada alojamiento mediante **preuHotel**, también inicializaremos cada precio con algún valor que sea coherente.

3.4.2. Objetivo

Los objetivos seguirán siendo los mismos porque no estamos añadiendo nuevas restricciones para que el planificador entienda que se deben cumplir. Simplemente tratamos de minimizar ciertos valores pero no se especifican como características del estado final.

3.4.3. Funciones

De manera análoga a los casos anteriores, tan solo tendremos que declarar en este apartado las variables que hemos definido nuevas en el estado inicial del problema y que como queremos que almacenen valor numérico deberán ser declaradas en esta sección.

3.4.4. Predicados

Se añadirá un nuevo predicado llamado **AllotjatHotel** que necesitaremos para saber en qué hotel se ha alojado en cada ciudad para poder hacer el cálculo del precio que es lo que se pretende minimizar en esta extensión.

3.5. Extensión 4

Por la descripción del problema realizada anteriormente, sabemos que la implementación de esta extensión consiste en unir las extensiones dos y tres que recordemos que la primera añadía interés a las ciudades y la segunda añadía precio tanto a los vuelos como a los alojamientos.

Analizando la problemática a resolver, observamos que simplemente uniendo todas las características que hemos añadido en las dos extensiones previas ya tendremos el dominio deseado.

Es decir, no necesitaremos añadir nada extra al dominio, dado que tanto los objetos como estado inicial, o los objetivos o predicados van a necesitar la declaración que hacen ambas extensiones, sin necesidad de añadir nada nuevo al entorno.

Capítulo 4

Operadores

En este capítulo nos centraremos en la explicación detallada de los operadores usados en cada uno de los casos a implementar descritos en los capítulos anteriores. Estos operadores son como las acciones que se pueden realizar para planificar el viaje. Disponen de ciertos parámetros, con los cuales se establece precondiciones sobre el entorno que activará la acción cuando este sea favorable y efectos/postcondiciones de las acciones.

4.1. Caso Base

Se trata de una extensión con pocas restricciones y predicados, con lo que la cantidad de operadores será proporcional. Disponemos de los siguientes.

4.1.1. *allotjar*

Los parámetros que usa son:

- ?c - ciudad
- ?a - alojamiento

Con estos parámetros se establece la precondición siguiente:

```
(and (en ?c)(Allotjamentde ?a ?c) (not(Allotjaten ?c)))
```

Define un entorno en el cual estás presente en la ciudad ?c, existe un alojamiento ?a en esa ciudad, y aun no estas alojado en ningún hotel de la ciudad.

Finalmente, el efecto/postcondición:

```
(and (Allotjaten ?c) (increase (visitades) 1))
```

Implica que el resultado de la acción será un entorno en el cual se ha marcado la ciudad como visitada, y por consiguiente aumenta en en 1 la cantidad de ciudades visitadas. Como último se define que se ha alojado en la ciudad ?c.

Este operador es necesario para hacer el cálculo sobre la cantidad de ciudades que se han visitado, puesto que en este caso, se especifica un mínimo de ciudades, además de ser necesario para marcar esa ciudad como visitada y así no volver a alojarse en ella. De esta manera, esta ciudad solo servirá como enlace entre otras.

4.1.2. *volar*

Los parámetros que usa son:

- ?c1 - ciudad
- ?c2 - ciudad
- ?v - vuelo

Con estos parámetros se establece la precondición:

```
(and (en ?c1)(Vuelode ?v ?c1 ?c2))
```

Define un entorno en el cual estás presente en la ciudad ?c1, existe un vuelo ?v con el trayecto de la ciudad origen ?c1 a la destino ?c2.

Finalmente, el efecto/postcondición:

```
(and(not(en ?c1))(en ?c2))
```

Implica que el resultado de la acción será un entorno en el cual se ha realizado un vuelo de la ciudad ?c1 a la ciudad ?c2 de modo que ya no se está presente en c1 y si se está en c2.

Este operador es necesario para realizar la conexión entre las diferentes ciudades a visitar. Evidentemente la única forma que tenemos de ir de una ciudad a otra es mediante los viajes en avión. Sin este operador no se podría cumplir nunca la restricción de visitar un mínimo de ciudades en concreto puesto que no existiría forma de comunicarse entre las diferentes ciudades.

4.1.3. *volar (2)*

Aunque el nombre sea el mismo que la acción explicada anteriormente, tiene varias diferencias. Pues esta se encarga de hacer que los vuelos sean recíprocos entre

ellos, de modo que si tienes definido un vuelo de ciudad1 a ciudad2, esta acción te permite hacer el vuelo inverso, es decir, la vuelta.

Con lo que se invierten las ciudades de la precondición:

```
(and (en ?c2)(Vuelode ?v ?c1 ?c2))
```

Y también las de la postcondición/efecto:

```
(and (not(en ?c2)) (en ?c1) )
```

4.2. Extension 1

Como se ha comentado anteriormente, esta extensión de trata de una amplificación del caso base, con lo que aparecen dos nuevas acciones y se aprovechan las acciones explicadas en el caso base, con los mismos parámetros, precondiciones y efectos. Las que se rehúsan son: **volar, volar (2)**.

4.2.1. *allotjar*

Los parámetros de la acción se mantienen , igual que la precondición.
Lo único que varía respecto al caso anterior es la postcondición/efecto:

```
(and (Allotjaten ?c) )
```

En este caso solo se marcará que se ha alojado en la ciudad ?c. A diferencia del caso anterior, no se sumará uno a la cantidad de ciudades visitadas, con lo que se identifica como visitada (posteriormente lo hará otra acción).

4.2.2. *assigna_dies_minims_a*

Los parámetros que usa son:

- ?c - ciudad

Con estos parámetros se establece la precondición:

```
( and (Allotjaten ?c) (= (diesperciutat ?c) 0))
```

Define un entorno en el cual estás presente y alojado en la ciudad ?c, y la cantidad de días asignados para visitar esa ciudad es igual a 0.

Finalmente, el efecto/postcondición:

```
(and (increase (diesperciutat ?c) (minim-d-c))

(increase (diastotal) (minim-d-c))

(increase (visitades) 1))
```

Implica que el resultado de la acción será un entorno en el cual se han aumentado los días asignados para visitar la ciudad ?c, al mismo tiempo que se aumentan los días totales del viaje. El aumento es la cantidad mínima definida de días definidos por ciudad. Además, es en esta acción en la que se marca como visitada la ciudad aumentando en 1 la cantidad de ciudades visitadas.

Este operador es necesario para la asignación de días de viaje, tanto en días por ciudad como en días totales que va a durar el viaje, siempre respetando los mínimos y máximos definidos. Sin esta acción no podríamos identificar una ciudad como visitada ni hacer el cálculo para llegar al mínimo de ciudades a visitar predefinido.

4.2.3. *assigna_un_dia*

Los parámetros que usa son:

- ?c - ciudad

Con estos parámetros se establece la precondición:

```
(and (Allotjaten ?c)(Ciutat ?c)(<(diesperciutat ?c) (maxim-d-c)))
```

Si se observa detenidamente, esta precondición define un entorno exactamente igual que al de la acción explicada anteriormente. Esto es debido a que nos interesa realizar esta acción otra vez. pero queremos cambiar los efectos que va a tener en el entorno.

Finalmente, el efecto/postcondición:

```
(and(increase(diesperciutat ?c)1)(increase(diastotal)1))
```

A primera vista puede parecer que también es la misma postcondición que pero en este caso, que el resultado de la acción, será un entorno en el cual se aumenta en 1 la cantidad de días asignados a una ciudad y la cantidad de días totales del viaje.

Este operador es necesario para varias cosas. Primero, en el caso de que se hayan asignado los días mínimos a todas las ciudades visitadas usando el operador anterior, y aún así, no se llegue a los días mínimos definidos para el viaje, habrá que

ir aumentando de uno en uno hasta cumplir con la restricción. En el caso contrario, esta acción sirve para asegurar la cumplimentación de las restricciones de cotas mínimas totales.

4.3. Extension 2

Se trata de una ampliación de la extensión 1, con lo que las funciones son las mismas pero hay aspectos de ellas que variarán y otras que quedarán intactas. En esta extensión aparece el interés por ciudad con lo que van a aparecer nuevas funciones y parámetros. Las acciones que se rehúsan son: **allotjar, volar, volar (2), assigna_un_dia**.

4.3.1. *assigna_dies_minims_a*

Los parámetros siguen intactos y la precondición no sufre cambios.

Esto deja clara la idea de que se ejecutará la acción en las mismas condiciones que en la extensión 1 pero los efectos sobre el entorno van a variar:

```
(and (increase (diesperciutat ?c) (minim-d-c))

(increase (diastotal) (minim-d-c))

(increase (visitades) 1)

(increase (interestotal) (interes ?c)))
```

Implica que el resultado de la acción será un entorno en el que se ha visitado una ciudad más, han aumentado los días de visita por esa ciudad la cantidad mínima de días asignados por ciudad, se han sumado los días por esa ciudad a los días totales del viaje, y por último se ha sumado el interés de esa ciudad al interés total del viaje.

La necesidad de esta acción es parecida a la que se ha definido en los casos anteriores pero tiene un par de importantes nuevas tareas. Es la encargada de marcar que la ciudad se ha visitado puesto que se ha asignado un día o más para visitarla, además se encarga de sumar los intereses de las ciudades visitadas con tal de maximizarlo y lograr visitar aquellas que tengan un interés mayor.

4.4. Extension 3

Para esta extensión, ampliamos la extensión 1 añadiendo un nuevo parámetro que implica condiciones y restricciones nuevas. Este parámetro es el precio, y su presencia provoca que en todas las acciones se produzcan algunos cambios.

4.4.1. *volar*

Los parámetros y precondiciones siguen exactamente igual.
Llegamos a verificar, pues, que el precio solo afecta a las causas que se provocan en el entorno cuando se ejecuta la acción.

Así queda la postcondición/efecto:

```
(and (not(en ?c1)) (en ?c2) (increase (preutotal)(preuVol ?v)))
```

La innovación en esta postcondición es la suma del precio del vuelo usado para ir de la ciudad ?c1 a la ?c2 en el precio total.

Es importante realizar esta suma puesto que tenemos que cumplir que el tiempo total está entre los valores de mínimo y máximo precio definidos.

4.4.2. *volar (2)*

Como ya se ha comentado en algún caso previo, esta acción de volar es l reversa de la primera. Entonces, como es de esperar, también es importante hacer la suma del precio del vuelo en el precio total del viaje, con lo que la única diferencia que tiene esta respecto a sus anteriores es la adición de esta suma (presente en el anterior.).

4.4.3. *allotjar*

Como las dos acciones explicadas anteriormente, en esta solo se produce un cambio en la postcondición:

```
(and (Allotjaten ?c) (AllotjatHotel ?a) )
```

Aparece una nueva modificación del entorno. Se marca el hotel en el que se hospeda el usuario con tal de tener ese hotel y su precio, para posteriormente, poder incrementar el precio total del viaje.

4.4.4. *assigna_dies_minims_a*

Los parámetros para esta acción son:

- ?c ciudad
- ?h alojamientoHotel

Con estos parámetros se establece la precondición:

```
( and (Allotjaten ?c) (AllotjatHotel ?h)

 (Allotjamentde ?h ?c) ( = (diesperciutat ?c) 0))
```

Aparece aquí el hotel asignado para alojarse durante la estancia en la ciudad ?c. La explicación de la precondicion es la misma que en las otras extensiones, con la variación que aparece el hotel de alojamiento junto con su precio por día.

Finalmente, el efecto/postcondición:

```
(and (increase (diesperciutat ?c) (minim-d-c))

(increase (diastotal) (minim-d-c))

(increase (visitades) 1)

(increase (preutotal) (* (minim-d-c) (preuHotel ?h))
```

Principalmente, respecto a la extensión 1, se le suma a precio total del viaje, el coste total de alojarse en el hotel. Puesto que el precio que obtenemos es por día, se multiplica el precio por la cantidad de días que se han asignado en esa ciudad en esta acción.

Es importante realizar el cálculo del precio total del alojamiento y sumarlo al del viaje, puesto que el objetivo final es minimizar el precio total y que este esté entre un rango de valores.

4.4.5. *assigna_un_dia*

Respecto a esta misma acción en la extensión 1, sufre los mismos cambios que la acción explicada más arriba. La diferencia es que, en vez de sumar el precio del hotel multiplicado por los días, se suma el precio del hotel por una noche, debido a que en esta acción, solo se incrementa en 1 la cantidad de días asignados a la ciudad.

4.5. Extension 4

Para esta extensión, tenemos que hacer una mezcla de las dos anteriores añadiendo ponderaciones. Los cambios producidos respecto el anterior serán pocos, en algunas postcondiciones se añadirán algunos efectos, pero puesto que es una mezcla de dos versiones no muy distantes, pocos serán los cambios.

Aquellas acciones que se quedan exactamente igual que la extensión 3 son: **allotjar, assigna_un_dia, volar, volar (2)**. (Las acciones volar, tienen el mismo nombre pero diferentes efectos, con lo que son diferentes acciones pero con el mismo código que en la extensión 3).

4.5.1. *assigna_dies_minims_a*

El motivo por el cual esta es la única acción que varía en la extensión 4 es porque es la única que interacciona con el interés de la ciudad. Esto es así puesto que es cuando se marca la ciudad como visitada, entonces se incrementa el interés del viaje.

Evidentemente, el interés es algo que solo afecta a la postcondición, pues el objetivo es sumarlo al total para luego maximizarlo:

```
(and (increase (diesperciutat ?c) (minim-d-c)))

(increase (diastotal) (minim-d-c))

(increase (visitades) 1)

(increase (preutotal) (* (minim-d-c) (preuHotel ?h)))

(increase (interestotal) (interes ?c)))
```

Como se ha comentado ya, el objetivo es incrementar el interés total del viaje. Pues est es la única diferencia en respecto a la extensión 3.

Capítulo 5

Juegos de prueba

En este apartado nos centraremos en proporcionar una serie de juegos de prueba por cada una de las diversas extensiones que hemos implementado. Todos los ficheros de problema han sido generados con un generador (hecho en python) que define los valores de forma random. Hay un generador por extensión. Se intenta buscar que se generen casos que pongan a prueba el correcto funcionamiento del programa.

Con tal de facilitar la lectura del documento, los ficheros de entrada y sus correspondientes resultados los pondremos en ficheros .pddl que se adjuntan como material adicional de la práctica.

5.1. Caso base

En esta extensión se nos pide hacer un viaje que pase por lo menos por un número de ciudades mínimas definido. La gracia de nuestro código es que permite que haya ciudades que hagan de puente con otras, es decir, no todos los vuelos pueden ir a todas las ciudades, pero sí se puede pasar por una ciudad, sin visitarla, para ir a otra.

En el **primer caso** pondremos a prueba nuestro código obteniendo una entrada en la que la cantidad de ciudades totales sea igual a la cantidad mínima de ciudades a visitar. Evidentemente el conjunto de aviones y sus correspondientes viajes tiene que formar un grafo conexo no completo para poder probar la validez de que se usan algunas ciudades como puente. El resultado esperado es que encuentre un camino por el cual llegar a todas las ciudades y visitarlas, es decir, alojarse en ellas. El camino tiene que usar ciudades como puente.

Para los datos de entrada tenemos el ***problema0_viatgeAmbEscala.pddl*** y una vez ejecutado, el resultado se encuentra en ***salida_caso_base_ciudades_puente.pddl***. Tal y como era de esperar, se ha encontrado un camino dónde se visitan todas las ciudades usando otras como puente para llegar a ellas.

Para el **segundo caso** queremos probar que si tenemos un grafo completo, no hay la necesidad de hacer escala entre ciudades. De este modo el viaje entre ciudades es más rápido. El resultado esperado es que para ir de una ciudad a otra use el vuelo de la ciudad en la que se encuentra presente, y no haga ninguna clase de escala con otra.

Los datos de la entrada se encuentran presentes en el fichero ***problema0_totsConnectats.pddl*** y una vez ejecutado el resultado se encuentra ***totsConnectats.pddl***. Tal y como era de esperar, los vuelos se hacen de la ciudad visitada (origen) a la ciudad a visitar (destino) sin pasar por por otras ciudades.

5.2. Extensión 1

En esta extensión se nos pide definir un número máximo y mínimo de días por cada ciudad, además de la cantidad de días mínimos que se quiere destinar al viaje. Al ser una ampliación del caso base se mantienen sus restricciones anteriores.

Para el **primer caso** queremos demostrar la correcta funcionalidad del programa. Aprovecharemos que tenemos el generador para la extensión 1 y obtenemos un fichero de problema ***problema1_TrobaSolucioValida.pddl*** con el que esperamos que el programa sea capaz de encontrar una solución, sea cual sea. No importa cuales sean los valores de los parámetros, solo esperamos una solución. Después de la ejecución, el resultado se encuentra en el fichero ***TrobaSolucioValida.pddl***. Tal y como hemos previsto, se ha encontrado una solución.

Para el **segundo caso** queremos poner a prueba el programa. Para ello el fichero del problema ***problema1_MinimNumdiesIncompatible.pddl***, consta de una cantidad fija de ciudades totales, y se asigna una cantidad concreta en el mínimo de días totales a durar el viaje. Esta cantidad tiene que ser superior a la multiplicación entre el número total de ciudades y la cantidad máxima de días que se puede estar en una de ellas. La idea de hacer esta prueba es que no se encuentre ninguna solución. Una vez hecha la ejecución, la salida se encuentra en el fichero ***MinimNumdiesIncompatible.pddl***. Como era de esperar, no ha sido capaz de encontrar una solución y se ha pasado todo el tiempo en un bucle intentando encontrarla.

Para el **tercer caso** encontrado en el fichero ***problema1_MinIgualMax*** disponemos de una cantidad de días máximos y mínimos asignados para cada ciudad igual. A lo que la cantidad de días mínimos asignados para todo el viaje y la cantidad de ciudades totales son lo suficientemente grandes como para que se encuentre solución. Además que el generador que hemos programado no permite que la situación del caso 2 ocurra. El resultado esperado para este caso es que encuentre un plan de viaje

en el cual la cantidad de días asignados para el es exactamente min_dies o mas_dies. Después de la ejecución, la salida se encuentra en el fichero ***MinIgualMax.pddl*** y el resultado es el plan de viaje que esperábamos.

5.3. Extensión 2

Para esta extensión se nos pedía ampliar la extensión 1, con lo que sus restricciones y parámetros se mantienen, y en este caso añadimos el interés por ciudad, el cual queremos maximizar. Pues las pruebas las dedicaremos a probar el funcionamiento de maximizar una variable.

Para el **primer caso** queremos comprobar el funcionamiento que tiene nuestro código cuando se encuentra con que la mayoría de ciudades disponibles tienen el mínimo interés para el usuario. Dado que las funciones de maximizar y de minimizar no optimizan, sino que llegan a una solución buena en un tiempo muy corto, nos encontraremos que aunque haya 2 ciudades con el máximo interés y queremos ir a 3 ciudades, es probable que no elija esas dos de más interés puesto a la manca de optimización de la función. Conocido este problema, no podemos intuir un resultado esperado, aunque lo deseable sería que se encontraran cuantas más ciudades de buen interés, mejor. El fichero en el cual se encuentra la entrada es ***problema2_MoltesCiutatsPocDesitjades.pddl***. Una vez ejecutado se envía la salida al fichero ***MoltesCiutatsPocDesitjades.pddl***. Aun no habiendo fijado una solución esperada, teníamos la idea de que probablemente la maximización no vaya a ser óptima, y el resultado ha sido este, escogiendo sólo una de las 3 ciudades con mayor interés y las demás ciudades a visitar con el menor interés.

Para el **segundo caso** queremos comprobar el funcionamiento del programa pero teniendo una situación contraria a la tratada previamente. Ahora queremos analizar el comportamiento teniendo la gran mayoría de las ciudades con el máximo interés. Para este caso concreto consideramos que si que podemos hacer una estimación del resultado esperado, pues al maximizar el interés de las ciudades intuimos que la gran mayoría de las ciudades a las que viajar serán aquellas con mayor interés, y puede caber la posibilidad de que se cuele alguna con un interés muy malo (aunque haya suficientes de buen interés) debido a la no optimización.

El valor de la entrada se encuentra en el fichero: ***problema2_MoltesCiutatsDeInteres_MinimCiutatsAlt.pddl***, cuyo resultado, una vez ejecutado, se encontrará en el fichero ***MoltesCiutatsDeInteres_MinimCiutatsAlt.pddl***. Si se observan los ficheros, se verá que existe una única ciudad con el menor interés y un mínimo de 10 ciudades a visitar. El resultado es un plan de viaje en el cual, la ciudad menos deseada, aparece, tal y como se había previsto.

5.4. Extensión 3

Para esta extensión se nos pedía que ampliaremos la extensión 1 añadiendo valores de precio por alojamiento y vuelos. Además de tener que definir un precio mínimo y máximo que va a costar el viaje en total. Evidentemente, el que se intentará en este caso es minimizar el precio total, con lo que podríamos encontrarnos con situaciones similares a las vistas en la extensión 2.

Para el **primer caso** nos centraremos en averiguar cómo se comporta nuestro código cuando no se pueden llegar a cumplir las condiciones descritas. Lo pondremos a prueba definiendo un precio máximo que nos aseguraremos de que sea inferior al precio total, jugando con las variables de mínimo de días totales, etc.

El resultado esperado es que el programa no será capaz de encontrar ninguna solución por el motivo de que no se cumplen las restricciones. Los datos de la entrada se encuentran en el fichero: ***problema3_PreuMaximInferior_PreuFinal.pddl***. Una vez ejecutado, los datos de la salida están en el fichero ***PreuMaximInferior_PreuFinal.pddl***. Como bien se ha dicho hace un momento, el programa no es capaz de encontrar una solución.

Para el **segundo caso** vamos a comprobar el correcto funcionamiento de nuestro planificador generando el fichero de entrada con el generador random (escrito en python). La intención de este caso de prueba es verificar que hace lo esperado y que el generador es capaz de realizar un fichero con solución. El fichero generado de forma automática se encuentra en ***problema3_FuncionamentNormal.pddl*** y una vez ejecutado, la salida se encuentra en el fichero ***FuncionamentNormal.pddl***. Evidentemente, se ha generado una entrada con la que se puede llegar a una solución.

5.5. Extensión 4

Para esta última extensión, el objetivo es juntar las extensiones 2 i 3. Para realizarlo, debemos minimizar cada uno de los criterios que sigue cada extensión. Como hemos visto anteriormente, en la extensión 2 utilizamos el (*metric: minimize (interestotal)*) , y en la extension 3 el (*metric: minimize (preutotal)*). La fusión de estos dos obtenemos (***metric: minimize (+ (* ponderacion_precio (preutotal)) (* ponderacion_interes (interestotal))***)". Como podemos intentaremos minimizar más el precio del viaje, o bien el interés general que tenemos en ese viaje, dependiendo de la ponderación de cada uno de ellos. Cada una de estas distintas ponderaciones, la hemos realizado sobre el mismo fichero de problema, para que

dentro de las mismas condiciones, poder apreciar el impacto de cada una de las ponderaciones.

Pese a creer que los resultados que íbamos a obtener serían salidas distintas para cada tipo de ponderación, no hemos obtenido ningún cambio al respecto, es por eso que llegamos a la conclusión de que la función de minimizar de pddl, no es una condición de restricción que se deba de cumplir siempre, sino que es un modo más de intentar guiar al heurístico en el momento de escoger camio. No obstante, este puede no ser el óptimo, y podría no ser ni el que nos lleve a una solución válida.

Adjuntamos en el fichero todos los casos que hemos probado, con sus respectivas salidas, donde podremos apreciar que la salida es la misma con unas diferencias mínimas como el tiempo de ejecución, de 0.00s. a 0.01s.

Capítulo 6

Evolución del tiempo respecto al tamaño de entrada

En este apartado nos centraremos en analizar el crecimiento del tiempo de ejecución en función del aumento de ciudades. Concretamente por cada valor máximo de ciudades diferentes, analizaremos diversos casos variando la cantidad de ciudades mínimas a visitar. De este modo, ampliamos los datos y seremos capaces de observar una tendencia más clara.

Para poder hacer tantos análisis, es necesario tener un fichero distinto para cada cantidad de ciudades totales y mínimas. Estos los hemos creado a partir del generador (escrito en python) de la extensión 4 pero con algunas modificaciones para que todos los ficheros se generen de golpe.

Para analizar mejor los resultados, se han hecho diversos grupos donde se mantienen estables la cantidad de ciudades totales y van aumentando las ciudades mínimas a visitar.
Debido a las limitaciones del *Metric-FF* los ficheros del programa generados tiene que rondar las 200 líneas de código, con lo que nos hemos limitado a legar a las 28 ciudades totales. Aunque al principio parecía un límite sin sentido, gracias a la ejecución de alguna prueba que excede esa cantidad, nos hemos dado cuenta de lo costoso en tiempo y recursos que puede llegar a ser. La gráfica resultante es la siguiente:

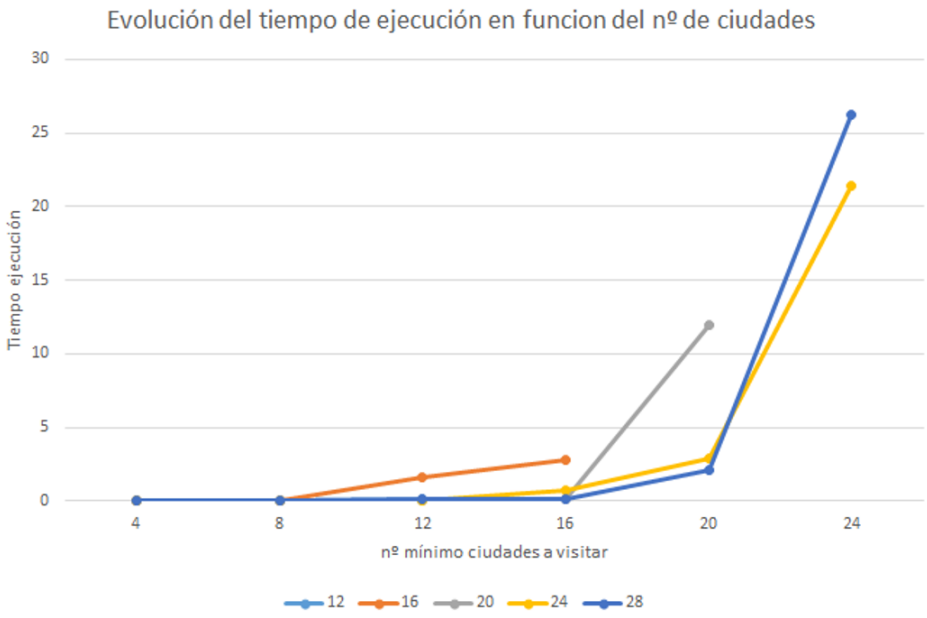

Figura 6.1: Evolución del tiempo de ejecución en función al número de ciudades.

Tal y como se puede observar a simple vista, la gráfica tiene una tendencia creciente para todos los casos que contemplamos. Inicialmente se mantiene en los valores bajos, debido a que la dificultad del problema es relativamente baja. Pero a medida que la cantidad mínima de ciudades a visitar se acerca a el total de ciudades, el aumento del tiempo es considerablemente notable.

Se podría decir que si dispusiéramos de los recursos necesarios, se podrían hacer muestras más grandes y la tendencia a crecer exponencialmente se vería más clara.

Capítulo 7

Conclusiones

En este trabajo, así como en el anterior, hemos tenido que modelar un problema de la vida cotidiana de una forma muy eficaz. Aunque nos hayamos centrado en solo este problema en concreto, la metodología de resolución se puede exportar a infinidad de casos diferentes.

Al resolver esta práctica por casos que se van ampliando y complicado, hemos podido darnos cuenta de la reusabilidad del código, además de resolver diversos problemas con un mismo dominio. Gracias a ello hemos podido comprobar que los planificadores generan planes de forma relativamente eficiente, pese a su facilidad de uso y expresividad.

Por ello consideramos que la realización de esta práctica ha sido muy enriquecedora y útil, tanto como para aprender un nuevo lenguaje de programación, como para los aspectos útiles de usar este.