PRÁCTICA 3

Dominios y problemas de planificación HTN

Óscar David López Arcos, 75571640-B
odlarcos@correo.ugr.es
Grupo A2: Miércoles 17:30-19:30

ÍNDICE

Problema 1

Resolver este problema ha consistido que añadir un nuevo método en la tarea **"transport-person"** para cuando el avión y la persona no se encuentren en la misma ciudad.

```
:precondition (and (at ?p - person ?c1 - city) (at ?a - aircraft ?c2 - city) (different ?c1 ?c2))
:tasks((mover-avion ?a ?c2 ?c1) (board ?p ?a ?c1) (mover-avion ?a ?c1 ?c) (debark ?p ?a ?c ))
```

Si están en ciudades distintas, el avión primero viajará a la ciudad donde se encuentre el pasajero para recogerle y, posteriormente, viajará al destino igual que en el método Case2.

Problema 2

Para solucionar las restricciones de fuel, he seguido los siguientes pasos:

En primer lugar, hay que modificar el predicado derived **"hay-fuel"** para que se corresponda con la realidad. Es decir, el contenido del depósito debe ser mayor que la distancia entre ambas ciudades multiplicada por el consumo a esa velocidad.

```
(:derived
  (hay-fuel ?a - aircraft ?c1 - city ?c2 - city)
  (> (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)) ))
```

En segundo lugar, crear un nuevo método en la tarea **"mover-avion"** que salte cuando no haya suficiente fuel. Este método hará repostar al avión antes de volar.

```
(:method no-fuel-suficiente
    :precondition (not(hay-fuel ?a ?c1 ?c2))
    :tasks( (refuel ?a ?c1) (fly ?a ?c1 ?c2)))
```

Problema 3

En este problema el avión solamente puede gastar un máximo de fuel preestablecido y tiene dos opciones para desplazarse: viajando rápido **(zoom)** o viajando lento **(fly)**.

Otra restricción del problema es que, siempre que se pueda, el avión debe desplazarse lo más rápido posible. Aquí entra en juego el uso de combustible ya que viajar rápido consume bastante más que viajar lento.

En primer lugar el predicado derived "hay-fuel" ha de dividirse en dos: **"hay-fuel-rapido"**, que tendrá en cuenta el consumo para los viajes rápidos y **"hay-fuel-lento"**, para los viajes lentos.

Además se usarán otros dos predicados **"no-excede-fuel-rapido/lento"**, para controlar que la acción a realizar no haga que el total de combustible usado supere el límite establecido. El límite lo he entendido como el total de combustible GASTADO, así que el combustible en el depósito podría superar el límite siempre que se controle no gastar más del preestablecido.

Por último la tarea **"mover-avion"** se modifica para que siga la siguiente estructura:

- Un primer método, **"fuel-suficiente-rapido"** que permita el vuelo rápido si se cumplen las precondiciones "hay-fuel-rapido" y "no-excede-fuel-rapido".
- Después, el método **"no-fuel-suficiente-rapido"**, que compruebe también si "no-excede-fuel-rapido" pero que salte si no se cumple "hay-fuel-rapido" de modo que reposte antes de volar.
- Por último, otros **dos métodos** similares a los anteriores, pero para el vuelo lento en lugar del rápido.

Esta estructura obligará a que, siempre que se pueda, se utilice el viaje rápido. Si no es posible, permitirá mediante "backtracking" elegir el viaje lento, con tal de ahorrar más combustible.

Problema 4

Para este problema ha sido necesario reestructurar completamente el dominio del problema.

El cuerpo principal es la tarea recursiva **“transport-people”**, sin argumentos, que se encargará de que todas las personas lleguen a sus destinos. Será necesario el predicado **(destino persona ciudad)** para conocer dónde transportar cada pasajero.

1. Para controlar las plazas disponibles en cada avión, he creado las funciones **“(pasajeros avion)”** y “**(max_pasajeros avion)** “, que determinarán respectivamente el número de pasajeros en un avión (inicializado a 0) y el máximo que este avión puede transportar. Las **tareas primitivas board** y **debark** se han modificado para que aumenten y disminuyan el contador de pasajeros.

2. La tarea transport-people seguirá la siguiente estructura de métodos:

- **Método Embarcar:** Comprueba que un avión y una persona estén en la misma ciudad, el destino de esa persona sea distinto a la ciudad donde se encuentran y que, o bien el avión esté vacío, o haya otra persona subida que comparta el mismo destino. Si aún quedan plazas disponibles en el avión, la persona podrá embarcar en él. Finalmente, se volverá a llamar a la tarea transport-people de forma recursiva, por si queda alguna acción por realizar.

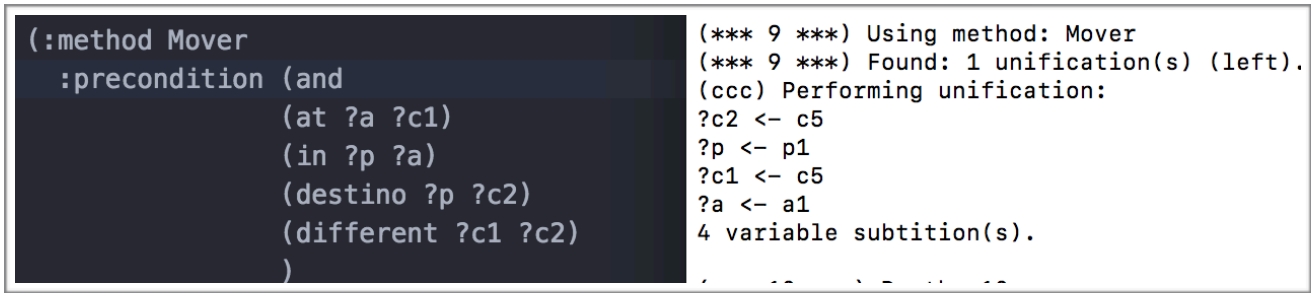
- **Método Mover:** Comprueba si hay una persona dentro de un avión y si esa persona no está en su ciudad de destino. Al estar situado este método después de “Embarcar”, ya se presupone que todos las personas con el mismo destino han embarcado en el avión. Como resultado, se llama a la tarea **mover-avion** (ejercicio anterior) y de nuevo recursivamente a transport-people.

- **Método Embarcar:** Si el avión está en una ciudad, hay un pasajero dentro y esta ciudad es el destino del pasajero, resulta en un desembarco del pasajero. Llamada recursiva a transport-people hasta cumplir el objetivo.

- **Método IrARecoger:** Por último, comprueba si hay una persona en una ciudad distinta de su destino y un avión en una ciudad distinta a donde se encuentre la persona. Esto provocará que el avión se mueva a la ciudad donde se encuentra la persona y se llame recursivamente a esta tarea (lo cual provocará la entrada en el método Embarcar).

- **Método caso-base:** Se disparará cuando no haya ninguna acción a realizar, lo que significa que o bien todos las personas están en su destino o que no se ha encontrado un plan satisfactorio.

En los métodos explicados anteriormente he decidido no utilizar los predicados derived que se nos dieron **(igual x y), (different x y)** y sustituirlos por **(= x y), (not(= x y))**, ya que a veces producían ciertas inconsistencias:



```
(:method Mover
  :precondition (and
                (at ?a ?c1)
                (in ?p ?a)
                (destino ?p ?c2)
                (different ?c1 ?c2)
                )
```

```
(*** 9 ***) Using method: Mover
(*** 9 ***) Found: 1 unification(s) (left).
(ccc) Performing unification:
?c2 <- c5
?p <- p1
?c1 <- c5
?a <- a1
4 variable subtition(s).
```

En este ejemplo puede verse como ambas variables c1 y c2 aceptan la misma ciudad, a pesar de que el predicado different debería impedirlo.

3. Por último, para representar un límite de tiempo para cada avión, he creado las funciones **(tiempo avion)** y **(max_tiempo avion)**. La primera servirá de contador, y aumentará en cada tarea primitiva conforme a la duración de ésta "(increase (tiempo ?a) ?duration)".

Para no superar el máximo de tiempo, he definido cuatro predicados derivados:
DaTiempoAEmbarcar, DaTiempoADesembarcar, DaTiempoAMoverseRapido y DaTiempoA-MoverseLento.

Estos predicados comprueban que, efectivamente, haya tiempo para realizar la acción que indican (el tiempo total más el tiempo de la acción no debe ser superior al límite). Este predicado se llamará en las precondiciones, antes de realizar dicha acción, y si no se verifica no permitirá realizarla.

Experimentación Problema 4

Ejercicio 1

En este ejercicio he mantenido la estructura de los ejercicios con 5 ciudades. He añadido un nuevo pasajero, y he definido las localizaciones de la siguiente forma:

Las personas p1, p2 y p4 se encuentran en la ciudad c2, mientras que el p3 está en la c3. Todos los pasajeros tienen destino c5, y el avión inicialmente está situado en c4.

Para un límite máximo de pasajeros establecido a 3, este es el resultado:

:action (zoom a1 c4 c2) start: 09/06/2007 12:00:00 end: 09/06/2007 20:00:00
:action (board p1 a1 c2) start: 09/06/2007 20:00:00 end: 09/06/2007 21:00:00
:action (board p2 a1 c2) start: 09/06/2007 21:00:00 end: 09/06/2007 22:00:00
:action (board p4 a1 c2) start: 09/06/2007 22:00:00 end: 09/06/2007 23:00:00
:action (refuel a1 c2) start: 09/06/2007 23:00:00 end: 22/06/2007 11:00:00
:action (zoom a1 c2 c5) start: 22/06/2007 11:00:00 end: 22/06/2007 19:00:00
:action (debark p1 a1 c5) start: 22/06/2007 19:00:00 end: 22/06/2007 20:00:00
:action (debark p4 a1 c5) start: 22/06/2007 20:00:00 end: 22/06/2007 21:00:00
:action (debark p2 a1 c5) start: 22/06/2007 21:00:00 end: 22/06/2007 22:00:00
:action (refuel a1 c5) start: 22/06/2007 22:00:00 end: 05/07/2007 10:00:00
:action (zoom a1 c5 c3) start: 05/07/2007 10:00:00 end: 05/07/2007 18:00:00
:action (board p3 a1 c3) start: 05/07/2007 18:00:00 end: 05/07/2007 19:00:00
:action (refuel a1 c3) start: 05/07/2007 19:00:00 end: 18/07/2007 07:00:00
:action (zoom a1 c3 c5) start: 18/07/2007 07:00:00 end: 18/07/2007 15:00:00
:action (debark p3 a1 c5) start: 18/07/2007 15:00:00 end: 18/07/2007 16:00:00

Como se puede ver, el funcionamiento para que es correcto y los 3 pasajeros situados en c2 embarcan simultáneamente para después desembarcar.

Para un límite máximo de 2 pasajeros hay que aumentar el máximo de tiempo del avión para que pueda finalizar. Con un tiempo 15 como el del apartado anterior, el plan se queda a medias:

:action (refuel a1 c4) start: 05/06/2007 08:00:00 end: 09/06/2007 12:00:00
:action (zoom a1 c4 c2) start: 09/06/2007 12:00:00 end: 09/06/2007 20:00:00
:action (board p1 a1 c2) start: 09/06/2007 20:00:00 end: 09/06/2007 21:00:00
:action (board p2 a1 c2) start: 09/06/2007 21:00:00 end: 09/06/2007 22:00:00
:action (refuel a1 c2) start: 09/06/2007 22:00:00 end: 22/06/2007 10:00:00
:action (zoom a1 c2 c5) start: 22/06/2007 10:00:00 end: 22/06/2007 18:00:00
:action (debark p1 a1 c5) start: 22/06/2007 18:00:00 end: 22/06/2007 19:00:00
:action (debark p2 a1 c5) start: 22/06/2007 19:00:00 end: 22/06/2007 20:00:00
:action (refuel a1 c5) start: 22/06/2007 20:00:00 end: 05/07/2007 08:00:00
:action (zoom a1 c5 c2) start: 05/07/2007 08:00:00 end: 05/07/2007 16:00:00
:action (board p4 a1 c2) start: 05/07/2007 16:00:00 end: 05/07/2007 17:00:00
:action (refuel a1 c2) start: 05/07/2007 17:00:00 end: 18/07/2007 05:00:00
:action (zoom a1 c2 c5) start: 18/07/2007 05:00:00 end: 18/07/2007 13:00:00
:action (debark p4 a1 c5) start: 18/07/2007 13:00:00 end: 18/07/2007 14:00:00

No puede apreciarse, el plan obtenido no resuelve el problema, faltaría recoger al pasajero p3. Con un tiempo máximo de 25, sí completa el plan:

```
:action (refuel a1 c4) start: 05/06/2007 08:00:00 end: 09/06/2007 12:00:00
:action (zoom a1 c4 c2) start: 09/06/2007 12:00:00 end: 09/06/2007 20:00:00
:action (board p1 a1 c2) start: 09/06/2007 20:00:00 end: 09/06/2007 21:00:00
:action (board p2 a1 c2) start: 09/06/2007 21:00:00 end: 09/06/2007 22:00:00
:action (refuel a1 c2) start: 09/06/2007 22:00:00 end: 22/06/2007 10:00:00
:action (zoom a1 c2 c5) start: 22/06/2007 10:00:00 end: 22/06/2007 18:00:00
:action (debark p1 a1 c5) start: 22/06/2007 18:00:00 end: 22/06/2007 19:00:00
:action (debark p2 a1 c5) start: 22/06/2007 19:00:00 end: 22/06/2007 20:00:00
:action (refuel a1 c5) start: 22/06/2007 20:00:00 end: 05/07/2007 08:00:00
:action (zoom a1 c5 c2) start: 05/07/2007 08:00:00 end: 05/07/2007 16:00:00
:action (board p4 a1 c2) start: 05/07/2007 16:00:00 end: 05/07/2007 17:00:00
:action (refuel a1 c2) start: 05/07/2007 17:00:00 end: 18/07/2007 05:00:00
:action (zoom a1 c2 c5) start: 18/07/2007 05:00:00 end: 18/07/2007 13:00:00
:action (debark p4 a1 c5) start: 18/07/2007 13:00:00 end: 18/07/2007 14:00:00
:action (refuel a1 c5) start: 18/07/2007 14:00:00 end: 31/07/2007 02:00:00
:action (fly a1 c5 c3) start: 31/07/2007 02:00:00 end: 31/07/2007 17:00:00
:action (board p3 a1 c3) start: 31/07/2007 17:00:00 end: 31/07/2007 18:00:00
```

Ejercicio 2

Para este ejercicio he definido las ciudades y distancias entre los aeropuertos de España, tal y como viene representado en el guión de la práctica. Ahora, algunos de los trayectos requieren más combustible de los que el depósito puede almacenar. Por eso he creado dos predicados derived que informarán de si el avión dispone de un depósito con la capacidad necesaria para abarcar dicho vuelo: **DepositoSuficienteRapido** y **DepositoSuficienteLento**. Cada predicado se llamará en el respectivo método no-fuel-suficiente para cada velocidad. Así, antes de rellenar el depósito para realizar el viaje, comprobará si el viaje es factible.

Definimos un problema con las siguientes características: p1 situado en Granada destino Bilbao, p2 situado en Madrid destino Bilbao, p3 y p4 situados en Huelva con destino Jaén. El avión inicialmente está situado en málaga.

Como puede verse, las distancias son demasiado amplias como para que el avión complete el plan cumpliendo todas las restricciones:

```
:action (fly a1 malaga granada) start: 05/06/2007 08:00:00 end: 06/06/2007 13:00:00
:action (board p1 a1 granada) start: 06/06/2007 13:00:00 end: 06/06/2007 14:00:00
:action (refuel a1 granada) start: 06/06/2007 14:00:00 end: 18/06/2007 18:00:00
:action (fly a1 granada bilbao) start: 18/06/2007 18:00:00 end: 20/06/2007 08:00:00
:action (debark p1 a1 bilbao) start: 20/06/2007 08:00:00 end: 20/06/2007 09:00:00
:action (refuel a1 bilbao) start: 20/06/2007 09:00:00 end: 06/07/2007 03:00:00
:action (fly a1 bilbao madrid) start: 06/07/2007 03:00:00 end: 07/07/2007 19:00:00
:action (board p2 a1 madrid) start: 07/07/2007 19:00:00 end: 07/07/2007 20:00:00
:action (refuel a1 madrid) start: 07/07/2007 20:00:00 end: 24/07/2007 07:00:00
:action (zoom a1 madrid bilbao) start: 24/07/2007 07:00:00 end: 24/07/2007 19:00:00
:action (debark p2 a1 bilbao) start: 24/07/2007 19:00:00 end: 24/07/2007 20:00:00
```

Si aumentamos tanto el límite de fuel como el máximo de tiempo y capacidad del depósito, sí obtenemos una solución:

```
:action (refuel a1 malaga) start: 05/06/2007 08:00:00 end: 09/12/2007 19:00:00
:action (zoom a1 malaga granada) start: 09/12/2007 19:00:00 end: 10/12/2007 10:00:00
:action (board p1 a1 granada) start: 10/12/2007 10:00:00 end: 10/12/2007 11:00:00
:action (zoom a1 granada bilbao) start: 10/12/2007 11:00:00 end: 11/12/2007 06:00:00
:action (debark p1 a1 bilbao) start: 11/12/2007 06:00:00 end: 11/12/2007 07:00:00
:action (zoom a1 bilbao madrid) start: 11/12/2007 07:00:00 end: 12/12/2007 03:00:00
:action (board p2 a1 madrid) start: 12/12/2007 03:00:00 end: 12/12/2007 04:00:00
:action (zoom a1 madrid bilbao) start: 12/12/2007 04:00:00 end: 12/12/2007 16:00:00
:action (debark p2 a1 bilbao) start: 12/12/2007 16:00:00 end: 12/12/2007 17:00:00
:action (zoom a1 bilbao jaen) start: 12/12/2007 17:00:00 end: 14/12/2007 05:00:00
:action (board p3 a1 jaen) start: 14/12/2007 05:00:00 end: 14/12/2007 06:00:00
:action (board p4 a1 jaen) start: 14/12/2007 06:00:00 end: 14/12/2007 07:00:00
:action (refuel a1 jaen) start: 14/12/2007 07:00:00 end: 01/06/2008 12:00:00
:action (zoom a1 jaen huelva) start: 01/06/2008 12:00:00 end: 02/06/2008 14:00:00
:action (debark p4 a1 huelva) start: 02/06/2008 14:00:00 end: 02/06/2008 15:00:00
:action (debark p3 a1 huelva) start: 02/06/2008 15:00:00 end: 02/06/2008 16:00:00
```

Ejercicio 3: Error al utilizar varios aviones

Al trabajar definiendo varios aviones he podido darme cuenta que mi estructura planteada para resolver el problema no es ideal si se utiliza más de un avión, debido al funcionamiento de HTN.

Mi idea era que, por cada método de la regla **transport-people**, comprobaría si algún objeto tipo avión cumplía las precondiciones y, en caso afirmativo, entraría en este método haciendo matching con dicho avión. Sin embargo, el funcionamiento real es distinto: Primero comprueba todos los métodos para un único avión, pasando al siguiente avión sólo si no ha conseguido cumplir las precondiciones para ningún método. El problema de este sistema es que el primer avión definido siempre hará matching con el caso-base, lo que provocará el fin del plan de forma repentina, antes de poder comprobar si hay algún otro avión.

Desafortunadamente no dispongo de tiempo para reestructurar el dominio del problema. Explicaré como he resuelto este ejercicio siguiendo el planteamiento empleado hasta ahora, a sabiendas de que existen mejoras estructuras que conseguirían planes más óptimos:

He definido un nuevo predicado **(ultimoAvion avion),** que inicilializaré con el nombre del último avión definido en el problema. En el caso base comprobaré que se trate de este avión para poder salir de la tarea. De esta forma me aseguro que se exploran las posibilidades de cada avión, aunque no puedo evitar que primero actúe uno y después el otro.

Para la misma representación del ejercicio anterior, donde un único avión no podía cumplir el plan al completo, he obtenido el siguiente plan:

```
:action (fly a1 malaga granada) start: 05/06/2007 08:00:00 end: 06/06/2007 13:00:00
:action (board p1 a1 granada) start: 06/06/2007 13:00:00 end: 06/06/2007 14:00:00
:action (refuel a1 granada) start: 06/06/2007 14:00:00 end: 18/06/2007 18:00:00
:action (fly a1 granada bilbao) start: 18/06/2007 18:00:00 end: 20/06/2007 08:00:00
:action (debark p1 a1 bilbao) start: 20/06/2007 08:00:00 end: 20/06/2007 09:00:00
:action (refuel a1 bilbao) start: 20/06/2007 09:00:00 end: 06/07/2007 03:00:00
:action (fly a1 bilbao madrid) start: 06/07/2007 03:00:00 end: 07/07/2007 19:00:00
:action (board p2 a1 madrid) start: 07/07/2007 19:00:00 end: 07/07/2007 20:00:00
:action (refuel a1 madrid) start: 07/07/2007 20:00:00 end: 24/07/2007 07:00:00
:action (zoom a1 madrid bilbao) start: 24/07/2007 07:00:00 end: 24/07/2007 19:00:00
:action (debark p2 a1 bilbao) start: 24/07/2007 19:00:00 end: 24/07/2007 20:00:00
:action (refuel a2 gibraltar) start: 24/07/2007 20:00:00 end: 06/08/2007 08:00:00
:action (fly a2 gibraltar jaen) start: 06/08/2007 08:00:00 end: 07/08/2007 14:00:00
:action (board p3 a2 jaen) start: 07/08/2007 14:00:00 end: 07/08/2007 15:00:00
:action (board p4 a2 jaen) start: 07/08/2007 15:00:00 end: 07/08/2007 16:00:00
```

Donde el avión 1 acababa, el avión 2 toma el relevo y acaba el plan que el primero empezó.