Técnicas de los Sistemas Inteligentes

Práctica 3 – Entrega 2

Curso 2016/2017

Francisco Javier Caracuel Beltrán

Grupo 3

Grado en Ingeniería Informática - CCIA

Índice

1. Ejercicio 1:

El dominio tiene dos métodos para transportar personas:

- Que la persona ya esté en la ciudad destino, donde no se hace nada.
- Que la persona esté en la misma ciudad que el avión y se lleve a su destino.

Falta que la persona esté en una ciudad y el avión en otra. Este método es el nuevo creado en este ejercicio para solucionar el problema. Para añadirlo se crea un tercer método en el *task* "*transport-person*" como sigue:

```
(:method Case3

    :precondition (and
        (at ?p - person ?c1 - city)
        (at ?a - aircraft ?c2 - city)
        (different ?c1 ?c2)
    )

    :tasks (
        (mover-avion ?a ?c2 ?c1)
        (transport-person ?p ?c)
    )

)
```

Se indica que una persona está en la ciudad *c1*, que el avión está en la ciudad *c2* y que la ciudad *c1* y *c2* no deben ser la misma. De esta manera ya coincide con el estado inicial del problema 1.

Las tareas que se deben realizar son mover el avión desde la ciudad en la que está el avión hasta la ciudad en la que está la persona y después realizar todo el proceso de embarcar, volar y desembarcar. Como los tres últimos movimientos ya están predefinidos en el método 2, se llama recursivamente al *task* "*transport-person*".

2. Ejercicio 2:

Ahora la capacidad de fuel es 300 y la carga inicial es 200. La distancia entre las distintas ciudades puede ser 100 o 150, por lo que seguro que como máximo puede viajar a dos ciudades antes del repostaje (pudiéndose quedar justo a 0 el fuel restante).

En el *task* "*mover-avion*" se encuentra definida la tarea *fuel-suficiente*, que comprueba con la precondición *hay-fuel* que pueda volar de una ciudad a otra. Este literal derivado *hay-fuel* comprueba que el fuel que se tiene en el depósito sea mayor que 1, pero no es correcto. Se ha modificado el literal derivado *hay-fuel*, para que compruebe que el fuel restante sea mayor que lo que consume al volar entre las dos ciudades a una velocidad lenta (ya que aún no se indica que vuele rápido).

Cuando no tiene fuel suficiente, es necesario repostar. Se añade un método en el *task* "*mover-avion"* que reposta el avión hasta su capacidad máxima. El método se llama *no-fuel-suficiente* y el resultado es:

```
(:method no-fuel-suficiente

    :precondition (and
        (not (hay-fuel ?a ?c1 ?c2))
        (different ?c1 ?c2)
    )

    :tasks(
        (refuel ?a ?c1)
        (mover-avion ?a ?c1 ?c2)
    )

)
```

En este método se comprueba que haya fuel para realizar el viaje y que las dos ciudades no sean la misma. En caso de no haber fuel, se reposta el avión en la ciudad que se encuentra hasta la capacidad máxima y se vuelve a llamar recursivamente a "*mover-avion*" para que se encargue del proceso de volar.

También se ha modificado el método "*fuel-suficiente*", indicando que las dos ciudades no pueden ser la misma.

3. Ejercicio 3:

En este ejercicio no se puede sobrepasar un consumo máximo de fuel. En el problema está inicializado el valor *fuel-limit*, pero no está definido en el dominio, por lo que se crea una *función* nueva con este nombre.

Se debe controlar que el fuel que se va a gastar para ir a un destino, junto con el que ya se ha consumido, no supere el máximo de fuel que se tiene disponible en total. Para esto se crean dos literales derivados, que calculen si ese límite se va a superar en un viaje. Se crea uno para la velocidad lenta y otro para la velocidad rápida, llamándose *available-fuel-slow* y *available-fuel-fast*. Estos literales derivados necesitan de un predicado, por lo que se crean sus dos predicados correspondientes.

Como las acciones *fly* y *zoom* ya tienen implementado el incremento de fuel consumido en ese viaje concreto, no es necesario hacerlo en el dominio.

En la tarea *mover-avion* ahora se tiene en cuenta la distinta velocidad que puede tener el avión, por lo que se crean dos métodos nuevos de volar y repostaje teniendo en cuenta la velocidad rápida. Estos métodos se llaman *fuel-suficiente-fast* y *no-fuel-suficiente-fast*. También se han renombrado los métodos existentes a *fuel-suficiente-slow* y *no-fuel-suficiente-slow*.

Dependiendo de si el método es *fast* o *slow*, en sus respectivas precondiciones se comprueban los literales derivados creados anteriormente, *hay-fuel-fast* o *hay-fuel-slow.* Las tareas que se ejecuta en cada método son triviales. Si se está en el método *fast*, la tarea será *zoom*. En cambio, si el método es *slow*, la tarea será *fly*.

Para indicar la prioridad de realizar vuelos rápidos, se implementan los métodos rápidos antes que los lentos. De este modo el planificador elegirá los vuelos rápidos para el avión.

Cuando ya se han terminado los tres ejercicios, se ha encontrado que no funcionaban los dos primeros, por lo que, partiendo del dominio completamente terminado, se ha vuelto a añadir el método *fuel-suficiente* del task *"mover-avion"* que se tenía en el ejercicio 2. Para que los tres ejercicios funcionen con el mismo dominio, es necesario reordenar los métodos de este *task*. Tras reordenarlos correctamente, los tres problemas funcionan (aparentemente) correctamente.

4. Ejercicio 4

Este ejercicio requiere que se puedan embarcar y desembarcar a varios pasajeros a la vez, estableciendo un límite de pasajeros en cada avión. Para que pueda ocurrir esto, se ha creado el predicado *destination ?p – person ?c – city*, con el que hay que indicar cuál es el destino del pasajero. Otras dos funciones son necesarias: *num-passenger ?a - aircraft* y *capacity-passenger ?a - aircraft*, que indica el número de pasajeros actuales y la capacidad máxima de pasajeros de un avión concreto.

Cuando están definidos los elementos anteriores, se debe modificar el archivo con las primitivas, permitiendo embarcar a un pasajero siempre que haya capacidad en el avión. Cuando se embarca un pasajero, se incrementa en 1 el número de pasajeros del avión. Cuando se desembarca un pasajero, se disminuye en 1 el número de pasajeros del avión.

La tarea *transport-person* debe cambiar un poco para permitir la posibilidad de embarcar a varios a la vez. La idea es que cuando llegue a una ciudad en la que hay varios pasajeros, sea cual sea el destino (excepto que su destino sea la propia ciudad) se suban en el avión. Para hacer esto se crea una nueva tarea *board-all* que recibe como parámetro la ciudad destino. Esta tarea tiene dos métodos. El primero embarca en el avión que se encuentra en la ciudad al pasajero y vuelve a llamar recursivamente a *board-all*. Se crea un segundo método vacío, para que cuando ya no exista la opción de embarcar a más pasajeros termine esa tarea.

El avión recoge a todos los pasajeros de una ciudad y va moviéndose ciudad por ciudad, dejando a cada uno en su destino correspondiente, como si fuera un autobús con sus paradas. El problema es que puede quedar limitado por el consumo máximo de fuel o por la duración máxima de los vuelos y no poder volar más, lo que haría que los pasajeros no pudieran llegar a su destino. Para solucionar este problema, como último método (para que se ejecute si no queda otra opción) se establece que el pasajero se baje del avión y se vuelve a llamar recursivamente a *transport-person*, para que de cualquier otra manera (con otro avión), el pasajero pueda llegar a su destino.

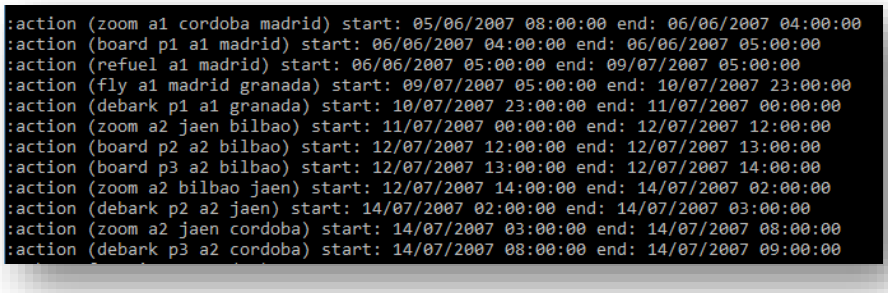
Otro requisito es que no pueden volar más de un tiempo máximo establecido. Para implementar este hecho, se crean las funciones *duration ?a – aircracf* y *max-duration ?a – aircraft*, que indican las horas de vuelo de un avión y la cantidad máxima que puede realizar.

Para que funcione, es necesario modificar en el archivo de las primitivas las acciones *fly* y *zoom*, aumentando las horas de vuelo dependiendo del viaje que realicen. También se modifican las precondiciones de los métodos del dominio para que no vuele si va a superar las horas de vuelo máximas permitidas. Esto se consigue creando dos predicados: *available-duration-fast ?a - aircraft ?c1 - city ?c2 – city* y *available-duration-slow ?a - aircraft ?c1 - city ?c2 – city*, con sus correspondientes literales derivados que controlan que no se exceda de las horas máximas.

Para terminar el ejercicio, lo ideal sería que varios aviones empezaran simultáneamente, para acortar el tiempo que se tarda en transportar a todos los pasajeros, pero no se ha conseguido. Como se puede ver en los problemas de prueba, cuando un avión ya no puede seguir volando por cualquier restricción, comienza otro, pero no se ha conseguido que comiencen en el mismo momento.

Se entregan cuatro problemas para este dominio:

1. Existen tres pasajeros. El primero se encuentra en Madrid, los dos restantes en Bilbao. El destino para cada uno, respectivamente, es Granada, Jaén, Córdoba. Existen dos aviones. El primero está muy limitado y solo puede hacer uno o dos vuelos (dependiendo de la distancia) y el segundo tiene capacidad suficiente para hacer todos los vuelos que desee.
   El resultado es:



2. Existen quince pasajeros, todos salen de Granada y su destino es Córdoba. El límite de pasajeros del primer avión es cinco y el límite del segundo diez. El resultado es:

```
:action (zoom a1 cordoba granada) start: 05/06/2007 08:00:00 end: 05/06/2007 16:00:00
:action (board p1 a1 granada) start: 05/06/2007 16:00:00 end: 05/06/2007 17:00:00
:action (board p15 a1 granada) start: 05/06/2007 17:00:00 end: 05/06/2007 18:00:00
:action (board p14 a1 granada) start: 05/06/2007 18:00:00 end: 05/06/2007 19:00:00
:action (board p13 a1 granada) start: 05/06/2007 19:00:00 end: 05/06/2007 20:00:00
:action (board p12 a1 granada) start: 05/06/2007 20:00:00 end: 05/06/2007 21:00:00
:action (zoom a1 granada cordoba) start: 05/06/2007 21:00:00 end: 06/06/2007 05:00:00
:action (debark p1 a1 cordoba) start: 06/06/2007 05:00:00 end: 06/06/2007 06:00:00
:action (zoom a2 jaen granada) start: 06/06/2007 06:00:00 end: 06/06/2007 11:00:00
:action (board p11 a2 granada) start: 06/06/2007 11:00:00 end: 06/06/2007 12:00:00
:action (board p10 a2 granada) start: 06/06/2007 12:00:00 end: 06/06/2007 13:00:00
:action (board p9 a2 granada) start: 06/06/2007 13:00:00 end: 06/06/2007 14:00:00
:action (board p8 a2 granada) start: 06/06/2007 14:00:00 end: 06/06/2007 15:00:00
:action (board p7 a2 granada) start: 06/06/2007 15:00:00 end: 06/06/2007 16:00:00
:action (board p6 a2 granada) start: 06/06/2007 16:00:00 end: 06/06/2007 17:00:00
:action (board p5 a2 granada) start: 06/06/2007 17:00:00 end: 06/06/2007 18:00:00
:action (board p4 a2 granada) start: 06/06/2007 18:00:00 end: 06/06/2007 19:00:00
:action (board p3 a2 granada) start: 06/06/2007 19:00:00 end: 06/06/2007 20:00:00
:action (board p2 a2 granada) start: 06/06/2007 20:00:00 end: 06/06/2007 21:00:00
:action (refuel a2 granada) start: 06/06/2007 21:00:00 end: 05/10/2007 03:00:00
:action (zoom a2 granada cordoba) start: 05/10/2007 03:00:00 end: 05/10/2007 11:00:00
:action (debark p2 a2 cordoba) start: 05/10/2007 11:00:00 end: 05/10/2007 12:00:00
:action (debark p3 a2 cordoba) start: 05/10/2007 12:00:00 end: 05/10/2007 13:00:00
:action (debark p4 a2 cordoba) start: 05/10/2007 13:00:00 end: 05/10/2007 14:00:00
:action (debark p5 a2 cordoba) start: 05/10/2007 14:00:00 end: 05/10/2007 15:00:00
:action (debark p6 a2 cordoba) start: 05/10/2007 15:00:00 end: 05/10/2007 16:00:00
:action (debark p7 a2 cordoba) start: 05/10/2007 16:00:00 end: 05/10/2007 17:00:00
:action (debark p8 a2 cordoba) start: 05/10/2007 17:00:00 end: 05/10/2007 18:00:00
:action (debark p9 a2 cordoba) start: 05/10/2007 18:00:00 end: 05/10/2007 19:00:00
:action (debark p10 a2 cordoba) start: 05/10/2007 19:00:00 end: 05/10/2007 20:00:00
:action (debark p11 a2 cordoba) start: 05/10/2007 20:00:00 end: 05/10/2007 21:00:00
:action (debark p12 a1 cordoba) start: 05/10/2007 21:00:00 end: 05/10/2007 22:00:00
:action (debark p13 a1 cordoba) start: 05/10/2007 22:00:00 end: 05/10/2007 23:00:00
:action (debark p14 a1 cordoba) start: 05/10/2007 23:00:00 end: 06/10/2007 00:00:00
:action (debark p15 a1 cordoba) start: 06/10/2007 00:00:00 end: 06/10/2007 01:00:00
```

   En los dos problemas restantes no se muestran los resultados para no extender este documento más de lo permitido.

3. El origen de los quince pasajeros sigue siendo Granada, pero en este caso, cada uno vuela a un destino diferente (hay varios que coinciden).

4. Cada pasajero de los quince tiene un origen diferente (hay varios que coinciden) y su destino sigue siendo diferente entre ellos.

Se ha intentado implementar estos mismos problemas con veinte pasajeros, pero en este caso, el planificador no ha conseguido establecer un plan válido en ocho horas de ejecución.