Memoria Práctica 3:

Dominios y Problemas de planificación HTN

Técnicas de los Sistemas Inteligentes

Christian Vigil Zamora

3º - Grupo 2

Índice

1. PROBLEMA 1

Como se puede comprobar, el dominio inicial no encuentra solución para el problema pues no existe ningún método dentro de la tarea **transport-person** que contemple la posibilidad de que el avión y los pasajeros no se encuentren en la misma ciudad. Es por ello que defino un método más, **Case3**, cuyas precondiciones ahora son que la persona se encuentre en una ciudad y el avión en otra diferente, y que la persona no se encuentre en la ciudad destino. Por lo tanto, las tareas que se realizan en éste método son: **mover-avion** desde la ciudad en la que se encuentra hacia la ciudad en la que está la persona. Una vez allí, se embarca la persona con **board**. Tras el embarque, se desplaza el avión hasta la ciudad destino de la persona con **mover-avion**. Y por último, una vez llegado el avión a la ciudad destino, el pasajero puede desembarcar con **debark**. De ésta forma, conseguimos satisfacer la resolución del problema, ya que el avión va a moverse a las diferentes ciudades en las que se encuentran las personas para llevarlas a su ciudad destino. Código:

```
(:method Case3 ;si no esta en la ciudad destino, y avion y persona no están en la misma ciudad
 :precondition (and (at ?p - person ?c1 - city)
                    (at ?a - aircraft ?c2 - city) (not (= ?c1 ?c2)))

 :tasks (
         (mover-avion ?a ?c2 ?c1)
         (board ?p ?a ?c1)
         (mover-avion ?a ?c1 ?c)
         (debark ?p ?a ?c )))
)
```

2. PROBLEMA 2

En éste problema, debemos considerar la opción de repostar, ¿Por qué?. Si nos fijamos, el fuel inicial del avión en éste caso es 200. Puesto que el avión se encuentra en la ciudad c4 y la persona p1 en la ciudad c1, ese desplazamiento de c4 a c1 suponen 150 de combustible, quedando 50, cantidad innecesaria para seguir con el problema. Es por ello que en la tarea **mover-avion** voy a incluir el método **fuel-insuficiente** para contemplar el suceso comentado anteriormente, que el avión no tenga suficiente combustible. La condición por tanto sería que no haya fuel suficiente y eso se comprueba mediante el predicado derivado **hay-fuel**, utilizado para deducir si hay fuel suficiente para que el avión vuele de una ciudad a otra. He de decir que la precondición sería la negación del predicado **hay-fuel**. Si se cumple la precondición, se reposta mediante la primitiva **refuel** y ya podríamos llevar a cabo el vuelo puesto que el fuel ya es suficiente. Código:

```
(:method fuel-insuficiente ;; este método se escogerá para usar la acción fly siempre que el avión
                           ;; NO tenga fuel para volar desde ?c1 a ?c2, por ello, repostará

 :precondition (not (hay-fuel ?a ?c1 ?c2))
 :tasks (
        (refuel ?a ?c1)
        (fly ?a ?c1 ?c2)
       )
 )
```

3. PROBLEMA 3

En primer lugar, añado al dominio 2 predicados nuevos: **hay-fuel-fast** y **hay-fuel-slow**. Éstos predicados van a ser empleados para poder determinar antes de realizar un vuelo si tenemos la opción de volar rápido o lento. **hay-fuel-fast** comprueba si el combustible que tiene el avión es suficiente para volar rápido de una ciudad a otra. El hecho de poder volar rápido se calcula multiplicando la distancia que hay entre las 2 ciudades por la razón de consumo del avión a velocidad rápida. **hay-fuel-slow** comprueba justo lo mismo pero considerando la razón de consumo del avión a velocidad lenta. Éstas novedades no son suficientes. Lo siguiente que modifiqué fue la tarea **mover-avion**. Dicha tarea deja de tener los métodos comentados en el problema anterior para pasar a tener 4 métodos nuevos. El primero de ellos es el método **fuel-suficiente-fast**, cuyas precondiciones son que haya combustible suficiente para ir de una ciudad a otra a velocidad rápida y que el consumo que va a suponer el viaje no sobrepase el valor de **fuel-limit**. Si se cumplen las condiciones, el viaje se realiza a velocidad rápida mediante la primitiva **zoom**. El segundo método es **fuel-insuficiente-fast**, el hecho de que sea el segundo método es porque debemos priorizar el viaje a velocidad rápida, por tanto en caso de no tener combustible suficiente para viajar rápido, debemos considerar la opción de repostar para poder llevarlo a cabo si es posible. Sus precondiciones por tanto son que NO haya combustible suficiente para viajar rápido y que el consumo que va a suponer el viaje no sobrepase el valor de **fuel-limit**. Si se cumplen, el avión reposta y vuela a velocidad rápida con la primitiva ya comentada. Esos 2 métodos son los que contemplan la opción de ir a velocidad rápida, ahora pasamos a los de velocidad lenta, que funcionan de forma muy similar. **fuel-suficiente-slow** es el método cuyas precondiciones son que haya combustible suficiente para ir de una ciudad a otra a velocidad lenta y que el consumo que va a suponer el viaje no sobrepase el valor de **fuel-limit**. Cumplidas las condiciones, el avión vuela a velocidad lenta con la primitiva **fly**. Por último, el método **fuel-insuficiente-slow** considera la opción de repostar para poder viajar a velocidad lenta. Por lo tanto, sus precondiciones van a ser que NO haya combustible suficiente para viajar a velocidad lenta y que el consumo que va a suponer el viaje no sobrepase el valor de **fuel-limit**. Si se cumplen, el avión reposta y vuela a velocidad lenta. De ésta forma conseguimos solventar las dificultades de éste problema, priorizando además en todo momento la opción de viajar a velocidad rápida. Código:

```
(hay-fuel-slow ?a ?c1 ?c2)
(hay-fuel-fast ?a ?c1 ?c2)
```

```
(:derived

  (hay-fuel-fast ?a - aircraft ?c1 - city ?c2 - city)
  (> (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a))))

(:derived

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

```
(:task mover-avion
 :parameters (?a - aircraft ?c1 - city ?c2 -city)
 (:method fuel-suficiente-fast
  :precondition (and (hay-fuel-fast ?a ?c1 ?c2)
                (> (fuel-limit)(+ (total-fuel-used) (* (distance ?c1 ?c2) (fast-burn ?a)))))
  :tasks (
          (zoom ?a ?c1 ?c2)
         )
  )
 (:method fuel-insuficiente-fast
  :precondition (and (not (hay-fuel-fast ?a ?c1 ?c2))
                (> (fuel-limit)(+ (total-fuel-used) (* (distance ?c1 ?c2) (fast-burn ?a)))))
  :tasks (
          (refuel ?a ?c1)
          (zoom ?a ?c1 ?c2)
         )
  )
 (:method fuel-suficiente-slow
  :precondition (and (hay-fuel-slow ?a ?c1 ?c2)
                (> (fuel-limit)(+ (total-fuel-used) (* (distance ?c1 ?c2) (slow-burn ?a)))))
  :tasks (
          (fly ?a ?c1 ?c2)
         )
  )
 (:method fuel-insuficiente-slow
  :precondition (and (not (hay-fuel-slow ?a ?c1 ?c2))
                (> (fuel-limit)(+ (total-fuel-used) (* (distance ?c1 ?c2) (slow-burn ?a)))))
  :tasks (
          (refuel ?a ?c1)
          (fly ?a ?c1 ?c2)
         )
  )
 )
```

4. PROBLEMA 4

1. Para adecuar el dominio a la posibilidad de representar que cada avión tiene una capacidad máxima de pasajeros y que cada que se embarca/desembarca el número de pasajeros se incrementa/decrementa en 1 he definido las siguientes funciones:

```
(capacidad-pasajeros ?a - aircraft)
(pasajeros ?a - aircraft)
```

la primera de ellas para delimitar la capacidad máxima de cada avión y la segunda para controlar el número de pasajeros a bordo de cada avión. También he modificado las primitivas **board** y **debark** añadiendo las nuevas características:

```
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration (boarding-time))
 :condition (and  (at ?p ?c)
                  (at ?a ?c)
                  (< (pasajeros ?a) (capacidad-pasajeros ?a)))
 :effect (and  (not (at ?p ?c))
               (in ?p ?a)
               (increase (pasajeros ?a) 1))
)
```

```
(:durative-action debark
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration (debarking-time))
 :condition (and (in ?p ?a)
                 (at ?a ?c))
 :effect (and  (not (in ?p ?a))
               (at ?p ?c)
               (decrease (pasajeros ?a) 1))
)
```

En la primitiva **board** he añadido la condición de que el número de pasajeros a bordo del avión antes de embarcar de nuevo sea menor que la capacidad máxima de dicho avión, además de añadir el efecto de que el número de pasajeros incremente en 1 cuando se realice un embarque. En la primitiva **debark** la única novedad ha sido añadir el efecto de decrementar el número de pasajeros en 1 cada vez que se realiza un desembarque.

2. En primer lugar he definido el predicado aconsejado en la pista: **destino ?x - person ?y - city** el cuál es usado para informar al planificador del destino de cada persona. Para poder representar la posibilidad de que varios pasajeros puedan embarcar o desembarcar de un avión he definido 2 tareas compuestas recursivas tal y como recomienda el guión. La primera de ellas ha sido **board-persons**:

```
(:task board-persons
  :parameters()

  (:method recursive
    :precondition (and  (at ?p ?c)
                        (at ?a ?c)
                        (< (pasajeros ?a) (capacidad-pasajeros ?a))
                        (not (destino ?p ?c)))
    :tasks ( (board ?p ?a ?c)
             (board-persons))
  )
  (:method caso-base
    :precondition ()
    :tasks ()
  )
)
```

La tarea está compuesta de un primer método recursivo en el que si se cumplen las precondiciones para poder embarcar pasajeros, mediante la acción que ya teníamos **board** montamos al pasajero y volvemos a llamar a la tarea para decidir si se monta la siguiente persona en caso de haberla. El segundo método sería sin más, el método caso base de la recursión. La segunda tarea compuesta ha sido **debark-persons**:

```
(:task debark-persons
  :parameters()

  (:method recursive
    :precondition (and (in ?p ?a)
                       (at ?a ?c)
                       (destino ?p ?c))
    :tasks ( (debark ?p ?a ?c)
             (debark-persons))
  )
  (:method caso-base
    :precondition ()
    :tasks ()
  )
)
```

Ésta tarea está compuesta nuevamente por un método recursivo el cuál se encarga de desembarcar a un pasajero y se llama así misma de nuevo para decidir si desembarca al siguiente pasajero en caso de haberlo, siempre y cuando se cumplan las precondiciones claro. De igual forma, el segundo método vuelve a ser caso-base.

3. Para representar que hay una duración limitada para los aviones he definido los siguientes predicados:

```
(slow-duration ?a - aircraft ?c1 - city ?c2 - city)
(fast-duration ?a - aircraft ?c1 - city ?c2 - city)
```

cuya finalidad será explicada más adelante. También he definido las siguientes funciones:

```
(duration-limit ?a - aircraft)
(total-duration-used ?a - aircraft)
```

la primera se utiliza para registrar la duración máxima permitida para avión y la segunda función se utiliza para llevar un conteo de la duración ya registrada por cada avión.
Los predicados comentados anteriormente son derivados:

```
;; literal utilizado para deducir si si es posible volar de la ciudad ?c1 a la ?c2
;; a velocidad lenta sin superar el limite de duración del avion
(:derived

  (slow-duration ?a - aircraft ?c1 - city ?c2 - city)
  (> (duration-limit ?a) (+ (/ (distance ?c1 ?c2) (slow-speed ?a)) (total-duration-used ?a))))

;; literal utilizado para deducir si si es posible volar de la ciudad ?c1 a la ?c2
;; a velocidad rapida sin superar el limite de duración del avion
  (:derived

  (fast-duration ?a - aircraft ?c1 - city ?c2 - city)
  (> (duration-limit ?a) (+ (/ (distance ?c1 ?c2) (fast-speed ?a)) (total-duration-used ?a))))
```

Como vemos, el primero de ellos permite deducir si es posible realizar un vuelo a velocidad lenta de la ciudad c1 a la ciudad c2 dada la duración que va a suponer dicho vuelo. La condición para que sea posible es que la duración del vuelo sea menor que el límite de duración máxima establecido para ese avión. El segundo predicado permite deducir justo lo contrario, es decir, si es posible realizar un vuelo a velocidad rápida entre ambas ciudades.
Por lo tanto, para que la nueva característica de la duración de los aviones se pueda complir habría que añadir en la tarea **mover-avion** que ya teníamos de los anteriores problemas, **slow-duration** y **fast-duration** en las precondiciones de los métodos adecuados, es decir,
slow-duration en los métodos para viajar a velocidad lenta y **fast-duration** en los métodos para viajar a velocidad rápida. También habría que añadir un nuevo método en la tarea **transport-person**:

```
(:method Case4 ;si ya estan montados los pasajeros pero no estan en la ciudad destino
 :precondition (and (in ?p - person ?a - aircraft)
                    (at ?a - aircraft ?c1 - city)
                    (not (destino ?p - person ?c1 - city)))

 :tasks (
         (mover-avion ?a ?c1 ?c)
         (debark-persons)
         (board-persons))
 )
```

Éste método básicamente comprueba que si un avión tiene dentro a una persona, y el avión no se encuentra en la ciudad destino del pasajero, realiza las tareas de mover el avión a la ciudad destino del pasajero, desembarcar y embarcar a personas que se encuentren en la ciudad destino, en caso de haberlas. Además de añadir en el método 2 y 3 de esa misma tarea la precondición de que la ciudad en la que se encuentra la persona sea distinta de la ciudad destino.

PROBLEMAS

1. Para éste primer problema, he considerado como viaje entre ciudades vuelo de ida y de vuelta, por tanto la cantidad de fuel limitado establecido ha sido el doble. Para gastar el menor combustible posible lo ideal es que se viaje entre las ciudades cuya distancia sea más corta además de aprovechar el hecho de que si hay personas en las ciudades donde se desembarca, se embarque a personas también y así se aproveche más ese viaje. El número de aviones para éste problema ha sido 2 y el de personas 10.

2. En éste problema se ha intentado ir de ciudad en ciudad cuya distancia sea menor, de esa forma la duración de los planes es más corta. Las condiciones de número de personas y aviones se mantienen con respecto al problema anterior, mientras que su localización y destino cambian.

3. En éste problema hay 22 personas, 3 aviones y diferentes configuraciones de avión, es decir, los aviones tienen distintas velocidades, razones de consumo,etc...