Técnicas de los Sistemas Inteligentes

Universidad de Granada

Práctica 3

Planificación HTN

Ignacio Vellido Expósito

Grupo Lunes

Ejercicio 1

El dominio proporcionado no resuelve puesto que no sabe qué hacer cuando el avión no se encuentra en la misma ciudad que la persona. Por tanto, se añade un nuevo caso para esta situación en concreto, donde ordenamos llevar el avión hasta la ciudad del pasajero para posteriormente trasladarlo al destino.

```
(:method Case3 ; Si el avión no está en la misma ciudad, traerlo primero
  :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)
         )
)
```

Ejercicio 2

En este problema al contar con poca capacidad de fuel el avión necesita repostar entre viajes. Para solucionarlo añadimos otro método a *mover-avion*.

```
(:task mover-avion
  :parameters (?a - aircraft ?c1 - city ?c2 -city)
  (:method fuel-suficiente  ;; este método se escogerá para usar la acción fly siempre que el avión tenga fuel para
                            ;; volar desde ?c1 a ?c2
                            ;; si no hay fuel suficiente el método no se aplicará y la descomposición de esta tarea
                            ;; se intentará hacer con otro método. Cuando se agotan todos los métodos posibles, la
                            ;; descomponsición de la tarea mover-avión "fallará".
                            ;; En consecuencia HTNP hará backtracking y escogerá otra posible vía para descomponer
                            ;; la tarea mover-avion (por ejemplo, escogiendo otra instanciación para la variable ?a)
   :precondition (hay-fuel ?a ?c1 ?c2)
   :tasks (
            (fly ?a ?c1 ?c2)
          )
  )

  (:method sin-fuel ; No podemos movernos sin repostar previamente
   :precondition (not (hay-fuel ?a ?c1 ?c2))
   :tasks (
            (refuel ?a ?c1)
            (fly ?a ?c1 ?c2)
          )
  )
)
```

Ejercicio 3

Duplicamos el método de *mover-avion* de forma que podamos diferenciar entre las acciones de *fly* y *zoom*. Se ordenan para que se priorice el vuelo rápido.

Para esto es necesario añadir funciones que nos calculen si el combustible es suficiente y no se sobrepasa el límite.

```
; Si no se pasa el consumo del límite
(:derived
  (no-sobrepasado-fuel-rapido ?a - aircraft ?c1 - city ?c2 - city)
  (< (+ (total-fuel-used) (* (distance ?c1 ?c2) (fast-burn ?a))) (fuel-limit))
)

(:derived
  (no-sobrepasado-fuel-lento ?a - aircraft ?c1 - city ?c2 - city)
  (< (+ (total-fuel-used) (* (distance ?c1 ?c2) (slow-burn ?a))) (fuel-limit))
)

; Para comprobar si hay suficiente fuel
(:derived
  (hay-fuel-lento ?a - aircraft ?c1 - city ?c2 - city)
  (>= (fuel ?a) (/ (distance ?c1 ?c2) (slow-burn ?a)) )
)

(:derived
  (hay-fuel-rapido ?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 rapido-no-refuel ; Al estar primero priorizamos el volar deprisa
    :precondition (and
                    (hay-fuel-rapido ?a ?c1 ?c2) ; Hay fuel para el vuelo
                    (no-sobrepasado-fuel-rapido ?a ?c1 ?c2) ; No sobrepasa el límite
                  )
    :tasks (
             (zoom ?a ?c1 ?c2)
           )
  )

  (:method rapido-refuel ; Probamos repostando
    :precondition (and
                    (not (hay-fuel-rapido ?a ?c1 ?c2))
                    (no-sobrepasado-fuel-rapido ?a ?c1 ?c2)
                  )
    :tasks (
             (refuel ?a ?c1)
             (zoom ?a ?c1 ?c2)
           )
  )

  (:method lento-no-refuel
    :precondition (and
                    (hay-fuel-lento ?a ?c1 ?c2)
                    (no-sobrepasado-fuel-lento ?a ?c1 ?c2)
                  )
    :tasks (
             (fly ?a ?c1 ?c2)
           )
  )

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

Ejercicio 4

Apartado 1

Guardamos dos funciones adicionales para recordar el número actual y máximo de pasajeros (*num-pasajeros* y *max*-pasajeros) y modificamos las acciones primitivas:

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

Apartado 2

Ponemos dos tareas que recursivamente van embarcando y desembarcando a todos los pasajeros que van al mismo destino que el avión. Modificamos las anteriores para que llamen a estas en vez de a *board* y *debark*.

```
(:task embarcar-pasajeros ; Embarca a todos los pasajeros en ?c1 con destino ?c2
  :parameters (?a - aircraft ?c1 - city ?c2 - city)
  (:method Case1
    :precondition (and
                    (at ?p - person ?c1 - city)
                    (at ?a - aircraft ?c1 - city)
                    (destino ?p - person ?c2 - city)
                  )
    :tasks (
             (board ?p ?a ?c1)
             (embarcar-pasajeros ?a ?c1 ?c2)
           )
  )

  (:method Case2 ; Si no hay nadie no hace nada
    :precondition ()
    :tasks ()
  )
)
```

```
(:task desembarcar-pasajeros ; Desembarca a todos los pasajeros con destino ?c
  :parameters (?a - aircraft ?c - city)
  (:method Case1
    :precondition (and
                    (in ?p - person ?a - aircraft)
                    (at ?a - aircraft ?c - city)
                    (destino ?p - person ?c - city)
                  )
    :tasks (
             (debark ?p ?a ?c)
             (desembarcar-pasajeros ?a ?c)
           )
  )

  (:method Case2 ; Si no hay nadie no hace nada
    :precondition ()
    :tasks ()
  )
)
```

Para comprobarlo añadimos un problema (*problemaEmbarqueMultiple.pddl*) con dos personas en la misma ciudad y con el mismo destino.

```
:action (board p1 a1 sevilla) start: 05/06/2007 08:00:00 end: 05/06/2007 09:00:00
:action (board p2 a1 sevilla) start: 05/06/2007 09:00:00 end: 05/06/2007 10:00:00
:action (fly a1 sevilla almeria) start: 05/06/2007 10:00:00 end: 07/06/2007 03:00:00
:action (debark p1 a1 almeria) start: 07/06/2007 03:00:00 end: 07/06/2007 04:00:00
:action (debark p2 a1 almeria) start: 07/06/2007 04:00:00 end: 07/06/2007 05:00:00
```

Apartado 3

Se considera una nueva restricción en la que el avión no puede pasar más de un cierto tiempo en el aire, para cumplirla se realiza lo siguiente:

- Se añaden dos *derived* para hacer la comprobación, y una función para almacenar el tiempo máximo de vuelo (*max-tiempo-vuelo*).
- Se modifica la tarea *mover-avion* para que cumpla con la restricción (añadiendo a la precondición).
- Se incluye un predicado *es-ciudad* para evitar que el planificador comprueba con objetos de otros tipos.
- Se duplica la tarea *mover-avion* y luego se le añade un nuevo método para incluir la opción de hacer escala.

Para verificar el apartado se incluye un nuevo problema (*problemaEscala.pddl*) donde se le da combustible suficiente al avión, aunque un tiempo pequeño de vuelo, obligándolo a hacer múltiples escalas para el trayecto.

```
:action (board p1 a1 sevilla) start: 05/06/2007 08:00:00 end: 05/06/2007 09:00:00
:action (zoom a1 sevilla cadiz) start: 05/06/2007 09:00:00 end: 05/06/2007 15:00:00
:action (zoom a1 cadiz cordoba) start: 05/06/2007 15:00:00 end: 06/06/2007 04:00:00
:action (fly a1 cordoba granada) start: 06/06/2007 04:00:00 end: 06/06/2007 20:00:00
:action (refuel a1 granada) start: 06/06/2007 20:00:00 end: 16/06/2007 14:00:00
:action (fly a1 granada almeria) start: 16/06/2007 14:00:00 end: 17/06/2007 06:00:00
:action (debark p1 a1 almeria) start: 17/06/2007 06:00:00 end: 17/06/2007 07:00:00
```

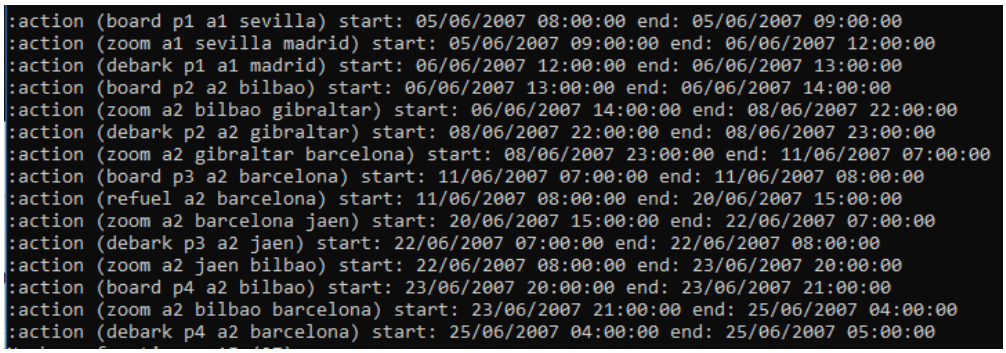
```
; Para comprobar si el avión puede hacer el vuelo en ese tiempo, en caso de que no hará escala
(:derived
  (hay-tiempo-lento ?a - aircraft ?c1 - city ?c2 - city)
  (>= (max-tiempo-vuelo ?a) (/ (distance ?c1 ?c2) (slow-speed ?a)) )
)

(:derived
  (hay-tiempo-rapido ?a - aircraft ?c1 - city ?c2 - city)
  (>= (max-tiempo-vuelo ?a) (/ (distance ?c1 ?c2) (fast-speed ?a)) )
)
```

```
(:method escala ; No puede con ninguno, probar otra ciudad
  :precondition (and
                  (es-ciudad ?c3 - city)
                  (not (= ?c2 ?c3))
                  (not (= ?c1 ?c3))
                )
  :tasks (
           (hacer-escala ?a ?c1 ?c3)
           (mover-avion ?a ?c3 ?c2)
         )
)
```

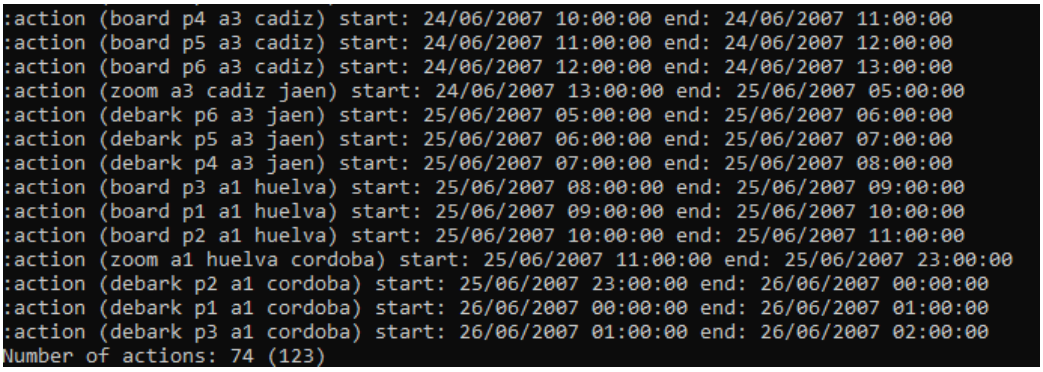
La tarea *hacer-escala* funciona de la misma forma que *mover-avion*, pero no cuenta con el método recursivo *escala* para evitar un bucle infinito.

También se añaden los siguientes problemas:

- *problemaFuel.pddl*: Con varios aviones y varios límites de fuel, donde se busca el consumo mínimo.



- *problemaTemporadaAlta.pddl*: Intentando transportar a 20 personas en diferentes viajes.



- *problemaTiempo.pddl*: Donde cada avión puede estar un tiempo máximo distinto en el aire.