Memoria Práctica 3

Técnicas de los Sistemas Inteligentes. Grupo Lunes.

Antonio Coín Castro

Curso 2018-2019

Ejercicio 1

Si ejecutamos el ejercicio proporcionado vemos que no nos proporciona un plan válido. Inspeccionando el dominio y el problema nos damos cuenta de que no estábamos considerando la situación en la que un avión estuviera en una ciudad distinta de la persona que va a transportar. Para arreglarlo simplemente definimos en la tarea compuesta `transport-person` un nuevo método que realice dicha comprobación, para posteriormente mover el avión a la ciudad de la persona y ya proceder como antes.

```
(:method Case3
 :precondition (and (at ?p - person ?c1 - city)
                    (at ?a - aircraft ?c2 - city))
 :tasks (
        (mover-avion ?a ?c2 ?c1)
        (board ?p ?a ?c1)
        (mover-avion ?a ?c1 ?c)
        (debark ?p ?a ?c))
)
```

Comprobamos ahora que resuelve el problema y proporciona un plan válido. La salida se encuentra en `E1/problema-zeno-v01.out.txt`.

Ejercicio 2

En este ejercicio la idea es imponer restricciones de fuel de forma que los aviones tengan que repostar antes de poder continuar con el vuelo hacia otras ciudades. Para conseguirlo, modificamos el predicado derivado que controlaba el nivel de gasolina del avión para ver si tenía suficiente para viajar entre dos ciudades dadas. Ahora en vez de comprobar

simplemente que no esté el depósito vacío, comprueba que lo que gasta en la longitud del trayecto sea menor o igual que lo que tiene.

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

Además, modificamos la tarea `mover-avion` para tener en cuenta estas nuevas restricciones. Comprobamos si tenemos fuel suficiente para volar entre las ciudades que queremos, y si no podemos hacerlo, se pasa el control a un método que primero se encarga de repostar para luego volar.

Si aún así no pudiésemos volar, significaría que nunca vamos a poder volar entre esas dos ciudades, porque la gasolina que gastamos es mayor que la capacidad del depósito (al repostar siempre se llena el depósito). En cualquier caso la tarea primitiva `fly` se encarga de hacer esta comprobación.

```
(:method no-fuel
 :tasks ((refuel ?a ?c1)
         (fly ?a ?c1 ?c2))
)
```

Consideramos además que es posible que tras un vuelo se quede el depósito vacío.

Vemos que ahora podemos resolver el segundo problema y obtener un plan válido. La salida se encuentra en `E2/problema-zeno-v02.out.txt`.

Ejercicio 3

En primer lugar, añadimos el *fluent* `(fuel-limit)` para controlar el fuel máximo que podemos gastar. Además, como ahora hay dos modos de vuelo, se divide en dos el predicado asociado al nivel de gasolina (y sus correspondientes derivados) para tener en cuenta dichos modos.

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

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

También modificamos la tarea `mover-avion` para reflejar las nuevas prioridades de vuelo. En total contiene cuatro métodos, ordenados según la prioridad otorgada a cada modo de vuelo. Además de las condiciones sobre la cantidad de gasolina necesaria para el viaje, en cada método comprobamos como precondición que tras realizarlo no hayamos superado el límite de fuel impuesto.

Los métodos de `mover-avion` quedan así:

```
(:method fuel-fast
   :precondition (and (hay-fuel-fast ?a ?c1 ?c2)
   (>= (fuel-limit)
   (+ (* (fast-burn ?a) (distance ?c1 ?c2)) (total-fuel-used))))

   :tasks ((zoom ?a ?c1 ?c2))
 )

 (:method refuel-fast
   :precondition (and (not (hay-fuel-fast ?a ?c1 ?c2))
   (>= (fuel-limit)
   (+ (* (fast-burn ?a) (distance ?c1 ?c2)) (total-fuel-used))))

   :tasks ((refuel ?a ?c1) (zoom ?a ?c1 ?c2))
 )

 (:method fuel-slow
   :precondition (and (hay-fuel-slow ?a ?c1 ?c2)
   (>= (fuel-limit)
   (+ (* (slow-burn ?a) (distance ?c1 ?c2)) (total-fuel-used))))

   :tasks ((fly ?a ?c1 ?c2))
 )

 (:method refuel-slow
   :precondition (and (not (hay-fuel-slow ?a ?c1 ?c2))
   (>= (fuel-limit)
   (+ (* (fast-burn ?a) (distance ?c1 ?c2)) (total-fuel-used))))

   :tasks ((refuel ?a ?c1) (fly ?a ?c1 ?c2))
)
```

Las tareas primitivas `fly` y `zoom` ya comprueban si podemos volar, así que podemos llamarlas justo después de repostar como en el ejercicio anterior.

En la interpretación realizada del enunciado se asume que siempre que pueda volar a la máxima velocidad (aunque tenga que repostar primero) lo hará. Es decir, la combinación de `refuel + zoom` tiene prioridad sobre `fly`.

Con estas modificaciones resolvemos el problema con límite de fuel, comprobando que siempre que puede viajar a la máxima velocidad lo hace. La salida está en `E3/problema-zeno-v03.out.txt`.

Ejercicio 4

Realizamos una serie de modificaciones al dominio de los ejercicios anteriores para reflejar todas las restricciones pedidas en el enunciado del problema. Las listamos a continuación:

- Modificamos las funciones `fuel-limit` y `total-fuel-used` para que reciban como parámetro un avión, pues ahora podemos tener varios aviones con distintas restricciones de fuel.
- Se añaden dos *fluents*, `people` y `max-people`, y se modifican las tareas primitivas `board` y `debark` para tener en cuenta la capacidad máxima de personas de cada avión. En particular, se incrementa y decrementa en 1 el valor de `people` al embarcar o desembarcar, y se añade la precondición al embarcar de que no se supere la capacidad máxima del avión `max-people`.
- Se añade un predicado `(destino ?p - person ?c - city)` que debe ser realizado por cada persona en los problemas. El `goal` ahora consiste únicamente en la acción `(transport-person)`.
- En las tareas primitivas, añadimos al efecto de `fly` y `zoom` el hecho de que se incrementa el tiempo total de vuelo de cada avión, según la distancia y la velocidad de vuelo en el trayecto. Se añaden para esto los *fluents* `time-limit` y `time-consumed`, y también se añaden dos predicados (y sus derivados) para comprobar que no nos pasamos del límite de tiempo en ninguno de los modos de vuelo:

```
(:derived
 (able-time-slow ?a - aircraft ?c1 - city ?c2 - city)
 (<= (+ (time-consumed ?a)
     (/ (distance ?c1 ?c2) (slow-speed ?a))) (time-limit ?a))
 )

(:derived
  (able-time-fast ?a - aircraft ?c1 - city ?c2 - city)
  (<= (+ (time-consumed ?a)
      (/ (distance ?c1 ?c2) (fast-speed ?a))) (time-limit ?a))
)
```

- Además, es necesario modificar la tarea `mover-avion` para ver en las precondiciones si podemos volar sin pasarnos del límite de tiempo para ese avión. Esto se hace comprobando que se cumplen los predicados que hemos añadido justo arriba, según el caso.

Una vez descritas estas modificaciones 'menores', pasamos a la gran variación del dominio: la tarea `transport-person`. Necesitamos que ahora sea recursiva, para permitir que embarquen, vuelen o desembarquen varios pasajeros a la vez, siempre que se cumplan las restricciones de capacidad y localización. Eliminamos los parámetros de la tarea, y delegamos en los métoodos para que controlen qué se hace en cada momento.

Pasamos a describir brevemente los métodos de la tarea. Siempre que acaban con éxito vuelven a llamar otra vez a `transport-person`, excepto obviamente el caso base.

- **Caso 1**: si una persona está montada en un avión y ese avión está en la ciudad de destino de la persona, la persona desembarca.
- **Caso 2**: si un avión está en una ciudad, una persona está en esa ciudad y esa ciudad no es su destino, la persona embarca en el avión.
- **Caso 3**: si una persona está en un avión y el avión está en una ciudad que no es el destino de esa persona, se mueve el avión a la ciudad de destino de esa persona.
- **Caso 4**: se da cuando una persona en una ciudad 1, un avión en otra ciudad 2, y el destino de la persona es otra ciudad 3 distinta de las dos anteriores. Entonces el avión va a recoger a la persona a la ciudad 1 y luego se mueve a la ciudad 3.
- **Caso 5** *(base de la recursividad)*: se llega a él cuando una persona está en su ciudad de destino, y por tanto ya no se hace nada. También es el caso al que se llega cuando no hay solución posible para la combinación de avión, ciudades y persona elegida.

Notamos por último que estos casos están **ordenados** según lo que se ha considerado mejor tras experimentar con varios problemas en distintas situaciones.

Problemas desarrollados

El **primer problema** considerado es una extensión del problema del ejercicio 3 sin restricciones adicionales, para comprobar que efectivamente este nuevo dominio consigue resolver los problemas anteriores. La salida está en `E4/problema-zeno-v04-1.out.txt`.

El **segundo problema** utiliza ya la matriz de distancias reales entre ciudades proporcionada en el guión, considerando 6 personas y 2 aviones. Imponemos un límite de fuel igual a 4 veces la máxima distancia entre todas las ciudades para cada avión, y comprobamos que dada la distribución de personas y aviones no consigue encontrar un plan para transportar a todas las personas. Sin embargo, aumentando ligeramente el nivel máximo de gasolina para uno de los aviones obtenemos un plan correcto. La salida puede consultarse en el archivo `E4/problema-zeno-v04-2.out.txt`.

Por último, el **tercer problema** consiste en transportar un número elevado de personas (21 en concreto) entre las ciudades, de forma que se cumplan las restricciones de tiempo impuestas. De nuevo observamos que si el tiempo máximo es demasiado bajo, no se consigue encontrar un plan para transportar a todas las personas, por lo que se ha ido aumentando hasta alcanzar un valor viable para el problema. La salida está en `E4/problema-zeno-v04-3.out.txt`.