Práctica 3 - TSI

Miguel Lentisco Ballesteros

Nota previa

He cambiado los predicados derived `igual` y `different` porque vi que no funcionaban bien, ahora deberían ir bien.

Ejercicio 1

Al ejecutar el primer problema (problema-zeno-v01.pddl) nos encontramos que no encuentra ningún plan, ¿por qué? Analizando la situación vemos que no hay ningún avión donde haya gente por lo que todos los intentos de realizar un plan fallan porque no hemos contemplado esta situación (p1 está en c1 y el avión en c4).

Solución: añadir un método al task `transport-person` de manera que si no hay un avión en la ciudad de la persona entonces movemos un avión de la ciudad donde esté hacia la ciudad de la persona que hay que transportar y volvemos a llamar al método recursivamente (donde podrá viajar ya que ahora sí hay un avión en su ciudad).

Ejercicio 2

Ahora al ejecutar el problema dos (problema-zeno-v02.pddl) tampoco encuentra plan, ¿que ha cambiado? Pues antes el avión a1 tenía una capacidad bastante grande y ahora se ha limitado, entonces al hacer distintos vuelos se queda sin gasolina y no tenemos ninguna manera de recargarla.

Solución: añadir un task `comprobar-fuel` que se encarga de ver si tiene suficiente gasolina para hacer el vuelo, si tiene no hace nada y si le falta entonces recarga con la acción proporcionada de las primitivas. Entonces en `mover-avion` hacemos una llamada a `comprobar-fuel` antes de despegar.

Añadimos un predicado derivado `falta-fuel` que nos dice si falta fuel si la gasolina actual es menor que la que se va a gastar volando y si es verdad entonces recargamos.

Ejercicio 3

En este ejercicio nos piden ahora que consideremos los modos de vuelo rápido y lento (`zoom` y `fly`) y que por defecto si podemos hagamos vuelo rápido. Además se incluye un límite de gasolina gastada entre todos los aviones que no se puede superar y hay que considerar esto.

Solución: Primero añadimos una function `fuel-limit` que representa el límite de gasolina gastada por todos los aviones. Ahora añadimos 2 funciones derivadas que nos dirán si podemos viajar en modo rápido o lento (que no excedan `fuel-limit`) usando la function que ya estaba `total-fuel-used` , `viaje-rapido-posible` y `viaje-lento-posible` que miran que `fuel-limit` sea mayor que la cantidad de gasolina a gastar en modo rápido/lento respectivamente.

Crearemos un nuevo task `modo-vuelo` que se encargue de manejar esta casuística e integrandola en `mover-avion` , de manera que intenta primero viajar en modo rápido y si no puede en lento.

Finalmente tendremos que cambiar el método de recargar gasolina para ser eficientes, ya que la cantidad de gasolina que se gasta es diferente en modo rápido que de lento. Usando las function derived que acabamos de crear, si podemos viajar rápido entonces comprobará si tiene que rellenar por tanto cambiamos el predicado `falta-fuel` por dos nuevos `falta-fuel-rapido` y `falta-fuel-lento` (lo mismo pero según `fast-burn` o `slow-burn`).

Ejercicio 4

En este ejercicio nos piden que ampliemos las funcionalidades:

1. Cambiar `debark` y `board` de manera que haya una capacidad máxima de pasajeros.
2. Modificar el dominio para poder llevar varios pasajeros en el avión a la vez.
3. Añadir restricciones temporales a los aviones.

Soluciones

1.

Para esto añadimos las funcion `max-people-capacity` y `current-people-capacity` que miden el límite y capacidad actual de pasajeros de un avión respectivamente, y modificamos `debark` y `board` para incrementar y decrementar los pasajeros (y solo aceptar si queda hueco en `board`), añadimos el predicado derivated `hay-hueco` para ver si hay hueco en el avión.

2.

Cuando hacemos la task para mover el avión `hacer-vuelo` , entonces añadimos las tasks `entrar-avion` y `salida-avion` , que son tasks recursivas de manera que mientras haya hueco y

haya gente en la misma ciudad que el avión entonces embarcan; cuando acaba, se vuela y al bajar se repite lo mismo con el desembarco (mientras haya gente en el avión); usando así el nuevo predicado `destino` para saber a donde van los pasajeros.

Ahora al volar los aviones podrán meter gente y desembarcar, evitando vuelos innecesarios.

3.

Añadimos las functions `fly-time-max` `fly-time` que representan el tiempo máximo y actual de un avión respectivamente. Añadimos los predicados derivados para que se cumpla que no se pase el límite `aircraft-has-time-slow` y `aircraft-has-time-fast` y las añadimos como precondiciones a los métodos `fly` y `zoom` respectivamente.

Ahora los aviones tienen un tope de tiempo que no pueden pasar (individual), por lo que modificamos `modo-vuelo` para que tenga en cuenta esto a la hora de elegir modo.

Estrategias

He hecho varias estrategias para minimizar el uso de gasolina, tiempo y acciones:

- Cuando se ha comprobado que se puede volar de x manera (rápida o lenta) entonces solo recarga si la cantidad de gasolina que necesita para ese modo no es sufiente (ahorramos tiempo y acciones).
- Cuando transportamos una persona de una ciudad a otra, la embarcamos primero a ella para asegurar que viaja y después mientras haya gente en la misma ciudad que vaya al mismo sitio y quede hueco en el avión se embarcan para ahorrar el nº de vuelos (y ganamos en todos los aspectos)
- Cuando para transportar a una persona haya que traer un avión desde otra ciudad (llamamos a `traer-avion`) entonces se consideran varias cosas (y en todo caso al final de todo se llamará de nuevo a `transport-person` de la persona inicial):
  - Si en la ciudad donde está el avión hay al menos una persona que necesita ser transportada a otro lado se llama a `otra-persona`:
  - Si hay personas que van a la misma ciudad donde se va a mover el avión o tienen el mismo destino que la persona original que estabamos transportando entonces empezamos a subir con prioridad los que van a la ciudad donde va el avión, y después los que van a la misma ciudad que la persona inicial; finalmente movemos el avión.
  - Si no se cumple esto, entonces la persona a mover va a otra ciudad completamente distinta y priorizamos transportar esta ya que tiene el avión en su propia ciudad, llamamos a `transport-person`
  - En caso contrario no tenemos ningun problema de mover el avión, así que lo movemos directamente.

Esta última estrategia obviamente no encontrará la solución óptima pero es una buena táctica a la hora de ahorrarnos nº de viajes innecesarios ahorrando así tanto en tiempo, gasolina como acciones.

Problemas

Tenemos entre manos tres restricciones importantes: el nº de personas a transportar, el nº de fuel máximo a usar y el tiempo disponible para cada avión.

Por tanto he creado 3 problemas `problema-zeno-v04-i` de i = 2 al 4, de manera que cada uno quita dos restricciones y solo se queda con una, el problema 2 se enfrenta con muchas personas (25), el problema 3 tiene una restricción de tiempo muy grande (100/90) y el problema 4 una restricción de fuel menor incluso la que se dice de 4 vuelos entre las dos ciudades más alejadas (viajando lento serían como unos 5000 pero se deja en 4300).

Para todos los problemas se encuentra una solución, y creo que buena.

Pero para probar realmente la bondad del dominio he incluido el problema 1 que pone al límite las 3 restricciones con 25 personas a transportar, 27k de fuel y 300 de tiempo cada avión; aun así el planificador consigue encontrar un plan bastante bueno.