Planificación con números y optimización

Práctica de laboratorio 1 – Parte 2/3 – Planificación Automática - Curso 2023-24

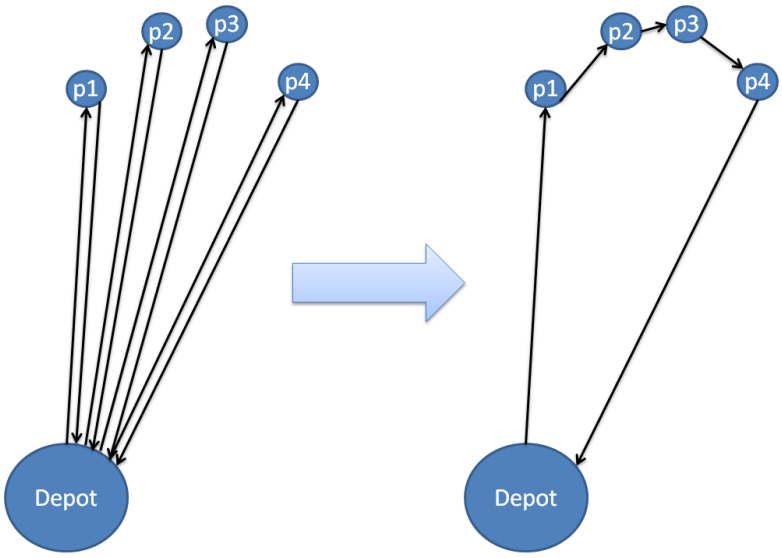
Memoria de la práctica

Para cada una de las tres partes de la práctica 1 será necesario escribir una breve memoria discutiendo los resultados de cada ejercicio. No hay un mínimo ni máximo de páginas para la memoria, simplemente debe explicar de forma clara y comprensible los resultados que se piden. Se puede incluir parte de la salida de los planificadores, si esta ayuda a explicar los resultados, así como tablas cuando puedan ayudar a sintetizar y comparar los resultados de distintas pruebas. La memoria debe entregarse en PDF. En cada ejercicio se indicará qué se debe explicar en la memoria.

Ejercicio 2.1: Servicio de emergencias, transportadores

Hasta ahora un dron era capaz de llevar un máximo de dos cajas, una por brazo. El planificador tenía en cuenta en qué brazo llevaba cada caja. Ampliar el número de brazos del planificador para mejorar su capacidad de carga resulta ineficiente, ya que el planificador considera estados distintos cada posible combinación en la que cada uno de los brazos agarra cada una de las cajas, lo que hace que el espacio de búsqueda (la cantidad de posibles estados) crezca exponencialmente con el número de brazos sin que esto redunde en ningún beneficio para la resolución del problema.

En este ejercicio consideraremos una forma alternativa de mover cajas mediante transportadores, estructuras que permitirán al dron volar transportando varias cajas al mismo tiempo, pero donde el orden no es relevante. El transportador sería como un container donde caben varias cajas, que deben ser metidas o sacadas de él con acciones explícitas. Un dron puede cargar hasta cuatro cajas en un transportador, que debe estar en su misma localización. Se puede suponer que los transportadores están inicialmente en la ubicación del depósito, al igual que todas las cajas. Una vez cargadas las cajas, puede llevar el transportador a un lugar donde haya personas que necesiten suministros y descargar una o más cajas de él. Puede continuar llevando el transportador a otra ubicación, descargando cajas adicionales, etc., y no tiene que regresar al depósito hasta que se hayan entregado todas las cajas del transportador.



Aunque actualmente solo es necesario un único transportador para el único dron que se utiliza en los problemas, se debe crear un tipo específico para el transportador. Es necesario saber qué y cuantas cajas hay en un transportador, para evitar cargarlo más allá de su capacidad. La capacidad de los transportadores será la misma para todos ellos y se podrá especificar en cada problema, por lo que formará parte del fichero de problema y no del dominio.

Es importante no introducir ranuras específicas en los transportadores (ej.: ranura1, ranura2, ranura3, etc.) ya que esto aumentaría el espacio de estados del problema de forma innecesaria, haciendo que existan estados redundantes, tal y como pasaría si aumentásemos el número de brazos que tiene el dron. Por ejemplo, si usásemos ranuras, se considerarían dos estados diferentes, aunque completamente equivalentes desde la lógica del problema, si una caja A está en la ranura 1 y una caja B en la ranura 2 que si la caja A está en la ranura 2 y la caja B en la ranura 1. Los estados redundantes harían que el espacio de búsqueda se amplíe innecesariamente, y llevaría a un mayor tiempo de procesamiento por parte del planificador cuando tenga que resolver un problema. Por este motivo no debemos registrar en qué posición está la caja, si no, simplemente, qué cajas hay y cuantas son en total.

Números mediante predicados

Dado que muchos planificadores no soportan valores numéricos, siempre existe la posibilidad de representar números mediante objetos y predicados PDDL estándar. Para ello, primero declararemos un tipo de dato numérico:

```
(:types … num …)
```

Después declararemos objetos que representen distintos números enteros:

```
(:objects … n0 n1 n2 n3 n4 n5 n6 n7 – num)
```

Para el planificador esto son simplemente objetos, por lo que tendremos que definir cualquier operación que queramos realizar con ellos mediante predicados:

```
(:predicates (siguiente ?numA ?numB – num)
```

En la inicialización del problema, habrá que establecer las relaciones entre números:

```
(:init … (siguiente n0 n1) (siguiente n1 n2) … (siguiente n6 n7)
```

Después podemos usar los objetos en las precondiciones y efectos de las acciones. Por ejemplo, en la siguiente acción de un dominio de planificación con ascensores, la acción de subir un piso se implementa de la siguiente manera:

```
(:action subir-piso
        :parameters (?a – ascensor ?desde ?hasta – num)
        :precondition (and
              (en ?ascensor ?desde)
              (siguiente ?desde ?hasta))
        :effects (and
              (not (en ?ascensor ?desde))
              (en ?ascensor ?hasta)))
```

Al no estar definido `(siguiente n7 …)`, cuando el ascensor se encuentre en el piso 7 la precondición `(siguiente ?desde ?hasta)` no se podrá cumplir nunca, por lo que no podrá subir más allá. Por otro lado, no es necesario definir el predicado “anterior”, como

opuesto a “siguiente”, ya que este último puede utilizarse también al revés, para verificar si un número es anterior a otro.

Acciones

Dependiendo de cómo se estructurase el dominio para el ejercicio anterior, es probable que sea necesario definir al menos tres nuevos operadores para este ejercicio: poner-caja-en-transportador, mover-transportador y coger-caja-del-transportador. La acción de poner una caja en el transportador debe estar precedida por una acción de coger la caja por parte del dron. Igualmente, la acción de sacar una caja del transportador continuará con la acción de entregar esa caja a alguna persona.

En este ejercicio se debe hacer lo siguiente:

1. Modificar el dominio según todo lo explicado previamente. Este nuevo dominio debe hacerse en un nuevo directorio para mantener intactas las versiones anteriores del dominio.
2. Adaptar el generador de problemas para que genere problemas acordes al nuevo dominio, incluyendo la inicialización de valores numéricos y transportadores.
3. Verifica que los dominios y problemas generados pueden ser resueltos correctamente utilizando las nuevas acciones relacionadas con el transportador. Haz pruebas con los planificadores de la parte 1 en problemas de pequeño tamaño. ¿Qué planificadores hacen uso de los transportadores? ¿Cuáles no?

Ejercicio 2.2: Servicio de emergencias, costes de acción

Como se indicó anteriormente, la intención detrás del uso de transportadores es que un dron no necesite regresar al depósito para cada caja. Sin embargo, para un planificador no es obvio que ir y venir largas distancias sea ineficiente, dado que no le estamos indicando el coste del desplazamiento. De hecho, usar el transportador puede resultar en planes más largo, al menos si consideramos únicamente el número de acciones. Por ello, es necesario añadir costes a las distintas acciones del dominio.

Costes de acción

Utilizando los conocimientos vistos en laboratorio, amplía el dominio y problemas anteriores mediante la introducción de costes de acción para asegurar que el vuelo entre grandes distancias tenga un coste comparativamente alto con coger y dejar cajas en una misma localización. Para ello, modifica el generador de problemas y haz uso de la función `flight_cost()`, que especifica el coste de volar entre localizaciones generadas aleatoriamente.

Para añadir costes a las acciones se debe hacer lo siguiente:

1. Añadir :action-costs a la sección de requisitos de la especificación del dominio. Este requisito limita la funcionalidad de optimización en PDDL a la minimización de un fluent denominado “total-cost”, que representa el coste de las acciones.
2. Definir una función de coste total denominada “total-cost” y una función de coste de vuelto “fly-cost” que reciba como parámetro origen y destino del vuelo, y devuelva su coste.
3. Incrementar el coste total en una cantidad fija como efecto de aquellas acciones con coste constante.

4. En las acciones de vuelo, añadir un efecto que incremente el coste total en función del coste de vuelto entre origen y destino.
5. Modificar el generador de problemas para que inicie el coste total a 0.
6. Modificar el generador de problemas para inicializar los valores de la función fly-cost entre cada par de localizaciones. Se debe hacer uso de la función flight_cost() para ello. Los costes siempre serán enteros no negativos.
7. Añadir una métrica de minimización del coste total en el generador de problemas.
8. Utiliza el generador para generar algunos problemas de prueba de distintos tamaños, manteniendo siempre 1 dron y 1 transportador con tamaño 4.
9. Investiga qué tamaño de problema son capaces de resolver los siguientes planificadores en un tiempo máximo de 1 minuto para devolver una primera solución: Metric-FF, OPTIC y Fastdownard con los alias LAMA (lama), BJOLP (seq-opt-bjolp) y FDSS2 (seq-opt-fdss2).
10. Para el mayor problema que haya resuelto metric-ff en 1 minuto, crea una tabla que incluya a todos los planificadores anteriores indicando qué tiempo tardan en resolverlo, qué tamaño de solución devuelven, y si los planificadores son óptimos, *anytime* o *portfolio*.