Planificación clásica con PDDL

Práctica de laboratorio 1 – Parte 1/3 – Planificación Automática - Curso 2025-26

Introducción

Los objetivos de esta práctica son los siguientes:

- Poner en práctica el modelado de problemas de planificación utilizando un lenguaje formal que los planificadores automáticos entiendan (PDDL)
- Adquirir cierta idea de las capacidades y limitaciones de las tecnologías actuales de planificación independiente del dominio.

Para hacer esto, tendrás que modelar en PDDL clásico un problema de logística de un sistema de atención de emergencia que inicialmente será sencillo y se irá haciendo más complejo en las siguientes partes de la práctica. Después tendrás que probar la especificación del dominio en PDDL con distintas instancias de problema de distinta complejidad, utilizando para ello distintos planificadores desarrollados por grupos de investigación alrededor del mundo.

Debido a la necesidad de probar con distintos planificadores, en esta primera práctica nos enfocaremos PDDL con el nivel de expresividad STRIPS. Es decir, no usaremos distintas extensiones del lenguaje como precondiciones negativas, metas negativas, efectos condicionales u otras similares. Muchos planificadores admiten algunas extensiones, pero pocos planificadores admiten exactamente las mismas, y a menudo tienen limitaciones o errores. Como excepción, sí usaremos la extensión ":typing", ya que esta sí está globalmente soportada por los distintos planificadores.

NOTA: No se permite la negación en precondiciones y metas, pero sí efectos negativos en las acciones, algo imprescindible para poder modelar prácticamente cualquier problema de planificación.

Entrega de la práctica

La práctica consta de tres partes que se entregarán juntas en un ZIP con tres carpetas, parte1, parte2 y parte3, que incluirán todo el código fuente (PDDL y Python) desarrollado en cada una de ellas. En la raíz del ZIP se incluirá un único fichero de memoria en PDF para las tres partes. En este documento se indicará el nombre de los alumnos que realizan la práctica y se responderá y discutirán los ejercicios. La memoria tendrá una sección para cada parte de la práctica, y en cada sección, un apartado para cada ejercicio. No hay un límite 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. En cada ejercicio se indicará qué se debe explicar en la memoria.

Ejercicio 1.1: Logística de servicio de emergencias, versión inicial

La versión inicial del problema será de la siguiente manera:

- Cada persona herida estará en una cierta localización
- Cada caja estará en una cierta localización y tendrá un contenido específico como comida o medicina.

    - NOTA: Para modelar el contenido de las cajas no se deben utilizar predicados del tipo (caja-comida ¿c) o (caja-medicina ¿c), ya que para añadir nuevos tipos de contenidos habría que modificar el dominio. En su lugar debe hacerse de forma genérica, de tal forma que, si se desean añadir nuevos contenidos a las cajas, pueda hacerse desde la especificación del problema exclusivamente.

- Cada persona habrá conseguido un cierto tipo de contenido o no. Es decir, se debe registrar si cada persona tiene comida o no, si tiene medicina o no, etc.

- Puede haber más de una persona en una localización concreta, por lo que no es suficiente con registrar la localización de una caja para saber si una persona tiene una caja.

- Inicialmente todas las cajas estarán localizadas en una única localización llamada depósito. No hay gente herida en el depósito.

- El objetivo será que ciertas personas tengan cajas con ciertos contenidos. Algunas personas no necesitarán nada, algunas necesitarán comida o medicina, etc. Esto significa que no tiene por qué ser necesario enviar todas las cajas del depósito. A las personas tampoco les importa qué caja reciben exactamente, si no que la caja tenga el contenido que les interesa.

- Un único dron, situado inicialmente en del depósito, está disponible para repartir cajas. Puede volar directamente entre cualquier par de localizaciones (no hay rutas de vuelto ni conexiones específicas entre localizaciones). Dado que el dominio será extendido a múltiples drones en el futuro, usaremos un tipo específico para los drones y, por el momento, crearemos un único objeto de dicho tipo en los problemas.

- El dron tiene dos brazos, con los que puede coger un máximo de dos cajas, una con cada brazo. Para coger una caja, ambos (la caja y el dron) deben estar en la misma localización. Después puede volar a otra localización trasladando la caja o cajas que haya cogido. Por último, puede entregar una de las cajas a una persona en específico que esté en dicha localización.

Dada esta descripción del problema, debes modelar el dominio en modo STRIPS en PDDL y construir dos problemas pequeños, uno con una persona y una caja, y otro con dos personas y tres cajas. Ten cuidado con los siguientes aspectos:

- Evita utilizar precondiciones y metas negativas, ya que muchos planificadores no las gestionan correctamente, además de poder darte problemas más adelante. Evita también cualquier otra extensión que pueda añadir complejidad a las precondiciones, efectos y metas.

- Elimina cualquier comentario que pueda haber en el código PDDL. Varios planificadores más antiguos no son capaces de procesar el problema si incluye comentarios.

- Revisa detenidamente la sintaxis, ya que algunos planificadores pueden romperse por pequeños errores de sintaxis. Por ejemplo, asegúrate de añadir espacios a ambos lados de “-” cuando especifiques un tipo, como en “?c - crate”. Si escribes “?c-crate” no es lo mismo.

Ejercicio 1.2: Generador de problemas en Python

Para comprobar el rendimiento de los planificadores a la hora de resolver problemas de planificación, es habitual hacer pruebas con problemas de dificultad creciente, que permitan averiguar la capacidad de resolución del planificador en un tiempo límite.

En este ejercicio debes realizar:

1. Desarrollar un programa en Python capaz de generar problemas para el dominio de planificación elaborado. A partir del esqueleto de código proporcionado junto al enunciado de la práctica, que incluye la lógica y la generación de un problema parcial, debes estudiarlo y completarlo para generar problemas completos. Una vez hecho, genera problemas invocando a este programa como en el siguiente ejemplo:

   ```
   python3 ./generate-problem.py -d 1 -r 0 -l 3 -p 3 -c 3 -g 3
   ```

   donde d son el número de drones, l las localizaciones, p las personas, c las cajas y g el número de metas que tendrá el problema (el parámetro r se utilizará más adelante).

2. Genera problemas de complejidad creciente manteniendo -d 1 -r 0 y los parámetros -l, -p, -c y -g con un mismo valor (ej: ./generate-problem.py -d 1 -r 0 -l 5 -p 5 -c 5 -g 5) y ve probando a resolverlos con el planificador FF. ¿Hasta qué tamaño de problema es capaz de resolver disponiendo de un tiempo máximo de 1 minuto?

3. Elabora una gráfica cruzando tamaño de problema y tiempo requerido para encontrar una solución en las pruebas realizadas.

Ejercicio 1.3: Comparativa rendimiento algoritmos de búsqueda y heurísticas

El planificador “pyperplan” implementa en Python varios algoritmos de búsqueda y heurísticas clásicas de planificación automática. Los algoritmos que implementa son:

- Breadth First Search (BFS)
- Iterative Deepening Search (IDS)
- A* (AStar)
- Weighted A* (WAstar)
- Greedy Best First Search (GBFS)
- Enforced Hill Climbing (EHC)
- Satisficibilidad booleana (SAT)

Las heurísticas que implementa son:

- hADD
- hSA
- hMAX
- hFF
- landmark
- lmcut

Utilizaremos pyperplan para probar la eficiencia de distintos algoritmos de búsqueda no informada e informada

1. Genera una secuencia de problemas de tamaño creciente e investiga el problema de mayor tamaño que pueden resolver los algoritmos BFS, IDS, A* y GBFS en un tiempo límite de 1 minuto. En los algoritmos heurísticos, aplica la heurística hMAX. Elabora una tabla con los resultados de cada algoritmo, indicando el tamaño del problema resuelto en el tiempo límite, el tiempo que tardó en resolverlo, el número de acciones del plan generado y si el plan generado es óptimo. Discute los resultados resaltando aquellos aspectos que te parezcan más llamativos y trata de justificarlos en base a las características del dominio y de los algoritmos utilizados.

2. Heurísticas para planificadores satisficing: Resuelve el problema de mayor tamaño que puede resolver el algoritmo Greedy Best First Search en 1 minuto y resuélvelo con todas las combinaciones de algoritmos GBFS y EHC y heurísticas hMAX, hADD, hFF y Landmark. Elabora una tabla indicando el tiempo tardado por cada combinación y el tamaño de la solución proporcionada. Investiga y explica las diferencias entre ambos algoritmos de búsqueda y explica qué uso hace de ellos el planificador FF.

3. Heurísticas para planificadores óptimos: Investiga cuales de las heurísticas implementadas en pyperplan son admisibles. Resuelve el problema de mayor tamaño que puede resolver el algoritmo A* con heurística hMAX en 1 minuto con BFS, IDS y A*, este último con cada una de las heurísticas que sean admisibles. Elabora una tabla indicando el tiempo tardado por cada combinación y el tamaño de la solución proporcionada. ¿Qué algoritmo o combinación de algoritmo y heurística sería la mejor según el tiempo tardado en generar una solución óptima en este problema? Discute los resultados y trata de justificarlos en base a las características del dominio y de los algoritmos utilizados.