Classic planning with PDDL

Lab Practice 1 – Part 1/3 – Automated Planning

Introduction

The objectives of this practice are as follows:

- Implement planning problem modeling using formal language that automatic planners understand (PDDL)
- Gain some insight into the capabilities and limitations of current domain-independent planning technologies.

To do this, you will have to model in classic PDDL a logistical problem of an emergency care system that will initially be simple and will become more complex in the following parts of the practice. You will then have to test the domain specification in PDDL with different problem instances of different complexity, using different planners developed by research groups around the world.

Due to the need to experiment with different planners, in this first practice we will focus PDDL with the STRIPS expressiveness level. That is, we will not use different extensions of language such as negative preconditions, negative goals, conditional effects, or the like. Many schedulers support some extensions, but few schedulers support the same ones, and they often have limitations or bugs. As an exception, we will use the ":typing" extension, since it is globally supported by the different planners.

NOTE: Negations are not allowed in preconditions and goals, but negative effects on actions are allowed, which is essential to be able to model virtually any planning problem.

Practice Delivery

The practice consists of three parts that will be delivered together in a ZIP with three folders, part1, part2 and part3, which will include all the source code (PDDL and Python) developed in each of them. A single PDF memory file for all three parts will be included in the root of the ZIP. In this document, the names of the students who carry out the practice will be indicated, and the exercises will be answered and discussed. The report will have a section for each part of the practice, and in each section, a section for each exercise. There is no page limit to the report, it simply must explain in a clear and understandable way the results that are requested. In each exercise, it will be indicated what must be explained in the report.

Exercise 1.1: Emergency Service Logistics, Initial Release

The initial version of the issue will look like this:

- Each injured person will be in a certain location
- Each box will be in a certain location and will have specific contents such as food or medicine.
    - NOTE: Predicates such as (food-box ¿c) or (medicine-box ¿c) should not be used to model the contents of the boxes, as adding new types of content would require modifying the domain. Instead, it should be done in a generic way, so

that, if new content is to be added to the boxes, it can be done from the specification of the problem exclusively.

- Each person will either get a certain type of content or not. That is, it must be recorded if each person has food or not, if they have medicine or not, etc.
- There can be more than one person in a particular location, so it's not enough to just record the location of a box to know if a person has a box.
- Initially, all the boxes will be in a single location called a warehouse. There are no injured people in the depot.
- The goal will be for certain people to have boxes with certain contents. Some people won't need anything, some will need food or medicine, etc. This means that it doesn't have to be necessary to ship all the boxes in the warehouse. People also don't care which box they receive exactly, but that the box has the contents they are interested in.
- A single drone, initially located in the depot, is available to deliver boxes. You can fly directly between any pair of locations (there are no return routes or specific connections between locations). Since the domain will be extended to multiple drones in the future, we will use a specific type for drones and, for the time being, create a single object of that type in the problems.
- The drone has two arms, with which it can pick up a maximum of two boxes, one with each arm. To pick up a box, both (the box and the drone) must be in the same location. You can then fly to another location by moving the box or boxes you have taken. Finally, you can deliver one of the boxes to a specific person who is in that location.

Given this description of the problem, you should do the following:

1. Model the domain in STRIPS mode in PDDL and build two small problems, one with one person and one box, and one with two people and three boxes.
    - Avoid using negative preconditions and goals, as many planners don't manage them correctly, as well as potentially giving you problems later. Also avoid any other extensions that could add complexity to the preconditions, effects, and goals.
    - Delete any comments that may be in the PDDL code. Several older planners are not able to process the issue if it includes feedback.
    - Be very careful with syntax, as some planners can be broken by small syntax errors. For example, be sure to add spaces on either side of "-" when specifying a type, such as in "?c - crate". If you type "?c-crate" it's not the same.

Exercise 1.2: Problem Builder in Python

To check the performance of planners when solving planning problems, it is common to test problems of increasing difficulty, which allow you to find out the ability of the planner to solve within a time limit.

In this exercise, you must perform:

1. Develop a Python program capable of generating problems for the elaborated planning domain. From the code skeleton provided along with the practice statement, which

includes logic and the generation of a partial problem, you must study and complete it to generate complete problems. Once done, it generates problems by invoking this program as in the following example:

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

where d is the number of drones, l the locations, p the people, c the boxes, and g the number of targets the problem will have (the r parameter will be used later).

2. Generate problems of increasing complexity by keeping -d 1 -r 0 and the parameters -l, -p, -c and -g with the same value (e.g. /generate-problem.py -d 1 -r 0 -l 5 -p 5 -c 5 -g 5) and try to solve them with the FF planner. Up to what size of problem are you able to solve with a maximum time of 1 minute?

3. Draw up a graph cross-referencing the size of the problem and the time required to find a solution in the tests carried out.

Exercise 1.3: Comparatively Performing Planners

The FF planner was one of the most successful in the first automatic planning (IPC2000) competitions, but a few years later it was widely surpassed by other planners such as LPG-TD or SGPLAN (IPC2004). In this exercise, we'll compare these three planners. Perform the following exercises using the FF, LPG-TD (with -n 1 option), and SGPLAN40 planners.

1. Generate a sequence of growing problem sizes and investigate the largest problem that can be solved by each of the planners within a time limit of 1 minute. For each, it displays in memory a fragment of the output per scheduler console. Prepare a table with the results of each planner, indicating the size of the problem solved in the time limit (or the error obtained if it had failed) and the number of actions of the plan generated. Which planner would be best based on its ability to process larger problems?

2. Solves the largest problem that the FF planner can solve in 1 minute with LPG-TD and SGPLAN40. For each, it displays in memory a fragment of the output per scheduler console. Draw up a table indicating the time taken by each of them and the size of the solution provided, in number of steps. Which planner would be best based on the time it takes to generate a solution? Which would be the best based on the size of the given solution? (Ties are possible, please indicate this in such a case).