Automated Planning

Classical Planning Assignment
Assigned: 24 August
Due: Wednesday, 31 September, 8am

Prof. Felipe Meneguzzi[*]

August 19, 2015

1 Rescue Robot Domain

You must work on this project **individually**. You are free to discuss high-level design issues with the people in your class, but every aspect of your actual formalisation must be entirely your own work. Furthermore, there can be no textual similarities in the reports generated by each group. Plagiarism, no matter the degree, will result in forfeiture of the entire grade of this assignment.

For this assignment you will be using the open source implementation of Graphplan available with the JavaGP library[1]. You can obtain a copy of this software from its SourceForge project site, or you can download a more conveniently packaged version of the software from our Moodle area. Your deliverables must be handed in via Moodle using the correspondingly named upload rooms. At the end of this assignment, you will upload **one zip file** containing the problem specification files in the PDDL format, specifically:

- one file named `robby.pddl` containing the domain encoding;
- three files containing the problems you formalised, named pbσ, where σ is the sequential number of your problem, e.g. `pb1.pddl`, `pb2.pddl`;
- the same number of files (with similar names) showing the traces created by JavaGP for the problems you created (e.g. `pb1.pddl` and `pb1.txt`); and
- one report following the guidelines detailed below.

2 Overview

In this assignment you will formalise *Robby*, a search and report robot. Robby's job is to navigate the mundane world of office hallways and rooms in the event of some unforeseen disaster, and look out for resources to rescue.

For this project, we will model a simple model of the world that Robby has to work in. This is essentially a long office hallway that is split into various segments or hallway-locations. These hallway-locations may or may not be connected to various rooms.

You will need to model two different types of locations, hallways and rooms. Robby can be "at" a particular location at a given point, and only at that location. Two locations can be "connected" to each other, enabling Robby to navigate between them (regardless of whether they are rooms or hallways). Note that connections are symmetric, so you need to model the fact that if Robby can go from A to B, then going from B to A is also possible. You can also assume that a hallway-location is a whole object - if Robby is anywhere in a given hallway-location, then Robby can enter any of the rooms connected to that hallway-location, and move from/to any of the other hallway-locations connected to it.

[*]Thanks to Subbarao Kambhampati and Kartik Talamadupula from Arizona State University for allowing me to use their domain `http://rakaposhi.eas.asu.edu/cse471/`.

[1]`http://sourceforge.net/projects/emplan/`

Navigating between hallway-locations and rooms is achieved via “enter” and “exit” actions. The enter action enables moving from a hallway-location to a room, while the exit action enables the opposite - moving from a room to a hallway-location. Remember that the two need to be connected in order to perform an enter or an exit - and you need to model those connections.

To navigate between two connected hallway-locations, Robby uses a special “move” action that only works on locations of type hallway. In the domain and problems that you are to model, Robby should not move within rooms or from one room to another directly, and only enters and exits from hallway-locations.

To ensure that Robby doesn’t cheat and visits all locations of interest, there are special beacons “in” those locations. The beacons can be in either hallway-locations or rooms (since they both are subtypes of “location”). Robby needs to necessarily be at a given location in order to spot the beacon at that location. Once Robby spots and “reports” that beacon, he can move on to the next task at hand (so you may want to model the reporting of beacons as goals). Finally, there is a destination that Robby must end up at - this is another goal that you must model.

Whenever we need to use automated tools to solve problems on our behalf, we must provide a consistent specification of the *transition system* of the underlying problem. If we specify the problem poorly, we jeopardise the planner software ability to generate valid responses, leading to false negatives and unnecessarily long waiting times for the planner at best, or incorrect plans at worse. Thus, specifying the Robby domain in PDDL gives you a chance to develop your skills in designing consistent transition systems, helping you avoid bugs in the software you will write in the future to handle all kinds of other processes.

Your assignment is to develop a domain file from the specification above, and then model the situations depicted in the images below as individual problem files. The following hints may be useful, but you are welcome to use your creativity as long as you adhere to the specification mentioned above:

- You need actions for moving between hallway-locations, as well as actions to enable the entering and exiting;
- You need an action to look for a beacon at a specific location and report it, so Robby can establish that he has been to a particular place (There is no sensing involved here; if a beacon is declared as being in a particular location, and Robby is also at that location, then he can report that beacon.);
- Look at the words in quote-marks in the specification above. They may give you a good skeleton to base your domain on.

3 Problem Instances

Below are images of the problem instances that you need to model in PDDL, once you are done making your domain file. The legend that accompanies each image should be fairly self-explanatory; remember, you must model connections between hallway-locations for Robby to move from one to the other, and you can only enter and exit between a hallway-location and a room that are connected. In the instances shown in Figures 1 through 3, any locations that share an edge can be considered connected (you do not need to explicitly model doors). Locations that share only corners and no edges are not connected.

Notice that the above pictures give you an idea of the initial state of the world (which you must encode in your PDDL problem file). They also tell you what the goals are - Robby’s final location (in green), and the various beacons that must be reported on the way. If you look at the syntax of example PDDL problem files, you will see that these are the three main parts of a problem file - (1) the objects, (2) the initial state, and (3) the goals.

4 Grading

In order to properly evaluate your work and thought process, you will write a 2-page report in the AAAI conference format explaining your encoding and experiments. These guidelines are to be followed **exactly**. **Reports that are less than two pages of actual content, or not in format will receive 0 marks for the report criterion.** This report will be included in the deliverables of **Part B** of

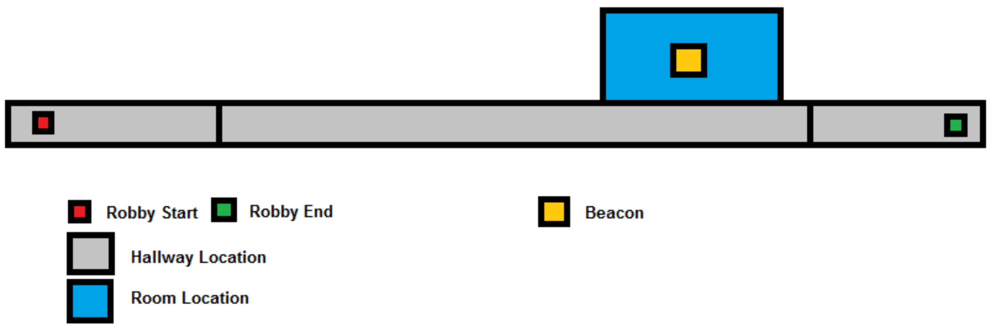

Figure 1: Problem 1

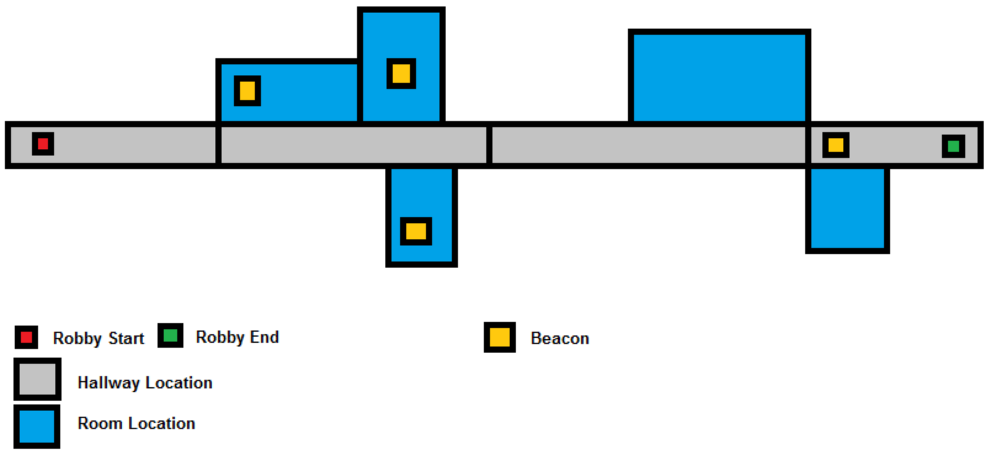

Figure 2: Problem 2

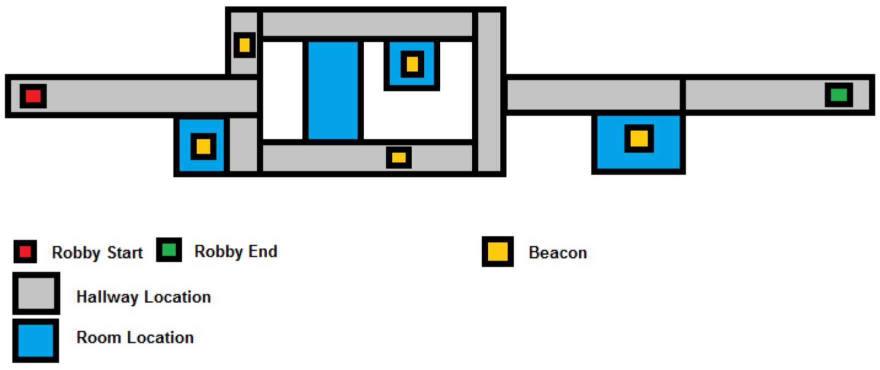

Figure 3: Problem 3

the assignment. The formatting instructions are available at the AAAI 2015 website[2]. The report must have the following sections:

- An introduction with your understanding of the problem domain, outlining the remainder of the paper;
- Two domain formalisation sections explaining your approach to formalising the problems from Section 3
- One experimentation section where the performance of JavaGP is measured using your action formalisation for each of the domains, on multiple problems.
- One conclusion section, where you will summarise your experience in encoding planning domains and discuss the performance of JavaGP, and any limitations encountered in solving the problems you encoded.

Grading will take consider elements of your encoding, experimentation and reporting of the work done. The criteria, as well as their weight in the final grade is as follows:

- Domain Encoding (30%) — correctness of the domain encoding, in relation to the domain specification from Section 2;
- Problem specifications (20%) — correctness of the problem specifications used for the experiments, particularly the initial state specification, as missing predicates here will jeopardise JavaGP's ability to solve your problem;
- Overall report readability (20%) — how accessible and coherent your explanation of your encoding is;
- Experiments (30%) — how coherent the proposed experiments are in measuring the performance of JavaGP, notice that this criteria is complementary to the one regarding problem specification.

5 Sample PDDL files

We have provided two template PDDL files at Domain Template and Problem Template to get you started, these were obtained from the myPDDL sublime package. We have also provided some sample files for other domains that you should look at before attempting to encode your own domain file and problem files. These will give you an idea of the information that goes into each of those files, and the syntax. You can also use these samples to test out the running of the planner, and to see what the output should look like.

- Blocks World Domain
- Gripper domain

6 Miscellaneous Advice

Here are some lessons we learned in creating our own solution and writing papers/reports:

- JavaGP does not parse "or" conditions or effects. You need to specify conditions / effects as a list of predicates bound together by a single "and".
- As a first step, you should at least look at the sample domain and problem files given at the beginning of this specification. Additionally, here is a link to PDDL domain and problem files (for scenarios different from the one you need to model in this project), from past iterations of the International Planning Competition (IPC).

[2] `http://goo.gl/OxC9H5`

- The best way to figure out how to model a domain and associated problems is to look at these examples. If you feel the need for documentation, here is a paper that talks about the complete PDDL specification, with BNF specification at the end (Appendix A): PDDL 2.1 Specification
- Using type predicates increases memory usage and slows down the creation of ground atoms but speeds up the (dominant) time to solve the problem (PDDL, however, allows typing of parameters);
- Tables and graphs are a useful tool to show runtime performance of software[3];
- In order to evaluate the performance of a planning encoding, you need to specify problems with most of the parameters locked in, and measure runtime as one parameter increases (e.g. number of locations, number of containers, etc);
- Pasting your entire domain specification (or problems) into the paper does not count as content (now you cannot say you were not warned);
- Overly large figures used to simply fill space in the report are also not a good idea;
- Reviewers have a more pleasant reading experience when papers are generated using LaTeX, it is very easy to spot the difference.

[3]GnuPlot is an excellent (and free) graph making software, available at `http://www.gnuplot.info/`