Automated Planning

Planning tool Manual

Viktoria Biliouri
Maria Ramirez Corrales

January 20, 2021

Contents

1 Planning Problem Examples

In order to check the functionality of our tool, we recreated two simple well-known planning problems, the blocksworld problem, and the Robot and rooms problem, which files are included in the matching folders. Each one of these planning problems contains a domain pddl file (*domain.pddl*) and a problem pddl file (*p01.pddl* and an additional *p02.pddl* for the blocksworld).

The blocksworld problem, represents a problem where we have different blocks that we can be on the table, or on each other, and they can be moved by using the actions *pickup*, *putdown*, *stack*, and *unstack*. The predicates that define the situation of the world are:

- onTable(block), that defines that one block is on the table
- on (block1, block2), that defines that the block1 is on the block2
- clear(block), that defines that there is not another block above this one.
- holding(block), that defines that this is held and it is not yet placed in a position.

The *p01.pddl* and *p02.pddl* contain the specific details of the problems that we want to solve. In our case this problem has two(2) for *p01.pddl* and three(3) for *p02.pddl* blocks named *a*, *b* and *c*, and each one of them defines the initial state of the blocks world and the desired goal state.

For the Robot and Rooms problem, we have a situation where there is a robot that it is able to open windows, that exist in different rooms. The robot can move from one room to an other, and close the windows that are open. The predicates that define this problem are:

- isIn(room), that defines in which room is the robot
- hasDoor (room1, room2), that defines that the robot can be moved from room1 to room2
- isOpen(window), that defines that the window is open.
- isWindowIn(window, room), that defines that the window is in which room.

The *p01.pddl* contains the information of the problem that we want to solve. In particular, we have three(3) windows, *w1*, *w2* and *w3* and two(2) rooms named *rm1* and *rm2*. After the definition of the objects, we have the initial and the goal state (which is that all the windows are open).

Note: The planning problems we created do not include conditional effects. That

is because, despite the fact that we modified the parser in order to parse store and ground the conditional effects, and this is functional, we did not have the time to integrate the conditional effects to the creation of the next states. Since, we were restricted on time, we decided to devote our time for functional a* and heuristics.

2 Execution Setup

In this part of the manual, we will give you some guidance about the execution setup of our code. The content that should necessarily exist in the machine where the tool executes, is our code *Astar_planner.c* and/or the Linux 64-bit executable file *Astar_planner*, one domain pddl file and one problem pddl file. If these files are able, the binary is ready to be executed, by:

- Step 1: adding as first argument the pddl file that contains the domain, and as second argument the pddl file that contains the problem. Attention should be paid, if these files are not in the same folder with the executable. In that case the names of the files should be added with the files' path. For example, if you execute the binary without changing our folder's layout, the command for executing the tool for the blockworld problem will be:
  `.\Astar_planner blocksworld\domain.pddl blocksworld\p01.pddl`
- Step 2: In order for the tool to fully execute for each problem, the domain pddl and the problem pddl should belong to the same domain name. For this reason we have added a condition, that exits the tool with an error message if this restriction is not satisfied.
- Step 3: An other restriction that you should take into consideration is the fact that our tool supports specific requirements of the pddl files. It only supports STRIPS, typing, negative preconditions and conditional effects. If an additional requirement exists in the planning problem, then the tool exits with an error message.
- Step 4: After all the restrictions are satisfied, you will notice all the actions and the relaxed actions printed on the console. Now it is the time, that the user inserts their preference regarding the heuristic that the algorithm should use. In particular: Press 1 for the Delete Relaxation Heuristics, or 2 for the Critical Path Heuristics.

If this process is effectively done, a plan of states will be printed on the console.

3 Execution Results - Evaluation

We have tested and obtained a successfully result using our planning tool only with the three referred problems.

If this planning tool only has succeeded with those three problems, it is because it was not possible to compute in a easy way the memory that needed to be allocated for every process, so some values on the memory allocation declaration are done with constant values that admit only small problems. When the improvements of the code will be done, these values will be better computed and this code could be used for bigger problems.

Anyway, we obtained some results from those three problems:

For the problem 1 of the blocksworld domain, that has only one goal state, the plan obtained with the different heuristics are the same because, indeed, Delete relaxation heuristics with h_{max} has the same result as h^1 heuristics, that is what happens with the critical path when the goal has only one goal atom.

For the problem 2 in the blocksworld domain and the problem on robot and rooms domain, the paths are different depending on the heuristics. We think this is normal, not only because of the type of used heuristics, but because in critical path heuristic algorithm we do not use the action costs that may retard the choice of some crucial actions when using delete relaxation heuristic.

To prove this theory we can presume that in the delete relaxation heuristic (that uses the action costs) solution for the robot and rooms problem, it chooses the *move* action more times because it costs less than the *open* action, even if the number of steps to achieve the goal is longer. However the goal is that all the windows are open so *open* is an action more crucial for the result.

For both problem 2 on blocksworld and problem on robot and rooms domain, the plan is shorter for the critical path heuristics because of this same reason.