50.021 -AI

Alex

Week 09: Planning

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Heuristics for planning

Consider the following STRIPS actions.

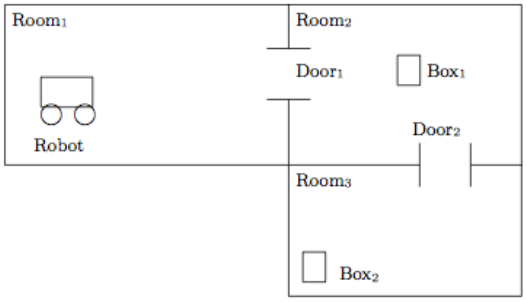
name	Pre	Add	Del
A	m	n,o	$\emptyset$
B	m,o	p	m
C	p	m	p
D	n,o	p	o

Assume the current state is $s = \{m\}$ and the goal is $g = \{m, n, o, p\}$

- What is the value of $h(s)$ (the actual shortest plan length)? Explain why.
- What is the value of $h_{max}(s)$? Explain why.
- What is the value of $h_{add}(s)$? Explain why.
- What is the value of $h_{FF}(s)$? Explain why.

PDDL Planning language

Consider the following planning domain consisting of a robot pushing boxes between connected rooms.



We will represent this domain with the following symbols

- b1, b2 the two boxes; r1,r2,r3 the three rooms, d1, d2 the two doors.
- open(?X) – door ?X is open.
- in(?X,?Y) – box ?X is in room ?Y.
- robin(?X) – the robot is in room ?X.
- join(?X,?Y,?Z) – door ?X joins rooms ?Y and ?Z.

Tasks:

- Specify the initial state shown in the figure. Assume that the doors start out closed.
- Give a STRIPS representation of the following action. Use the above symbols to specify sensible preconditions and effects for the action pushthru(?B,?R1,?R2,?D) the robot pushes box ?B from room ?R1 to ?R2 via door ?D.

  it should look like this

  (:action pushthru
  :parameters (?a1 ?a2 . . .)
  :precondition (and (whatever1) (whatever2). . . (whatever99))
  :effect (and . . .)
)
- We want an action to open a door ?D. This action should work when the robot is in either of the rooms that the door joins. Give a STRIPS representation for this action; make sure that you have all the necessary facts in your initial state.

Planner Search

Download the code for this week. The code defines a simple planner The key data structures are:

- states are represented by a Python set of propositions (assertions) each of which is a tuple of the form (type, arg1, arg2, ...), for example, ('free', 'partA').
- actions are represented as instances of the Operator class (in strips.py). An action is specified by a name, a list of preconditions (facts that must be present in the state for the action to be applicable), add effects (facts that are added to the current state)

and delete effects (facts that are deleted from the current state). An action op (an instance of Operator) has two key methods: op.applicable(state) checks whether the action is applicable in a state and op.apply(state) produces the new state specified by the action.

- tasks are represented by an instance of the Task class (in strips.py). A task specifies the initial state, a list of all the possible facts, a list of goal facts (which must be present in the final state) and a list of action instaces. The Python class Task has several essential methods for defining the search problem; you should look at the definition.

The file main.py is meant to be called as follows:

python main.py directoryName fileName

where directoryName indicates where the domain.pddl and problem files can be found. filename is the name of a problem file (without any extension). For example:

python main.py prodigy-bw bw-simple

your tasks:

- Finish the planner implementation by writing a class PlanProblem which is derived from search.Problem in main.py.
- Experiment with `astar_search` for your search. Start with the trivial heuristic that always returns 0 and try a couple of problems in the prodigy-bw directory. Use an InstrumentedProblem (from the old code from week8, copy the class, you can derive it from your PlanProblem class) to keep track of how many expansions are done. Then, try a simple heuristic such as counting the number of unsatisfied goals and see the effect on the efficiency of the search.

Homework 1: Writing PDDL –

deadline is in 1.5 weeks - Monday 24th of July

The objective of this problem is for you to get some experience in specifying a domain and some problems in PDDL and running your planner on them.

Create a painting directory. Start your domain file by copying the prodigy-bw/domain.pddl file into the painting directory. This has the definition of the "standard" blocks-world domain.

- Extend the domain by adding a new predicate (color-of ?x ?color) that allows us to specify the color of an object. There are a small number of colors (red, green, blue). We will add another type of object, sprayers, that come in different colors. Sprayers can be moved around just like blocks. Add a new operation spray that uses a sprayer of a particular color to change the color of a block to that color. In order to spray an object, that object must be on the table and clear and the sprayer must be held by the arm.
- You will need to introduce additional predicates to indicate the "types" of the objects. The planner will check these types and will avoid instantiating too many meaningless operators. This has a big effect on efficiency. You should make sure that planner is printing the Task instance that is created from parsing the input; this will show you all the operator instances.
- Define a problem (po.pddl) with one block that starts out red and a green sprayer, both on the table and both are clear, and the goal is that the block is green and the arm is empty.
- Define a problem (p1.pddl) with two blocks (A and B), with A on B. They both start out red. There is a green sprayer on the table and clear. The goal is that B is green and A is on B and the arm is empty.
- Use the Python planner you have been working on to test the examples. To run the example in po.pddl in the painting directory you can run

  python main.py painting po

  or you can set the dirName and filename variables in the main.py and evaluate the file.
- If you have extra time, you can include painting with a brush in addition to a sprayer. Painting with a brush requires paint cans of different colors. A much more elaborate version of this assignment can be found here `http://www.csee.umbc.edu/courses/graduate/671/fall12/hw/hw6/`.

Homework2: More on planner search

deadline is in 1.5 weeks - Monday 24th of July

- Implement one (or optionally more) of the heuristics listed below and add it to the Python planner.

  - h_{max}
  - h_{add}
  - h_{FF}

- The slides from Thursday's lecture describe these heuristics. Note that the h_{max} and h_{add} heuristics only need the "forward" pass described in the slide "Computing h_{FF}: Relaxed Planning Graphs (RPG)" (slide 47/78); to compute h_{FF} you also need the backward pass described in "Computing h_{FF}: Extracting a Relaxed Plan" (slide 48/78). At the bottom of this file are examples of computing these heuristics for the logistics domain in the lecture slides. Make sure that you debug the heuristic computation separately, before you try to run it on a full planning problem. In the main.py file, after you create an instance of the planning problem (a subclass of search.Problem), you can call the heuristic function defined there:

```
prob = PlanProblem(task)
# In search.py, heuristics takes an instance of search.Node
ffv = prob.h(search.Node(task.initial_state))
print 'initial h_FF', ffv
```

Here are the values we obtained for h_{FF} in the initial state (and the final plan length) of the following problems (from the prodigy-bw domain):

```
problem    h_ff plan
bw-simple   2   2
bw-sussman  5   6
bw-large-a 12  12
bw-large-b 16  18
bw-large-b 26  28
bw-large-d 34  36
```

You should also have your implementation print the facts and actions at each of the levels in the forward pass as well as the actions selected in the backward pass. Check these results by hand on simple cases, such as bw-simple.pddl. Make sure that your results match what you expect.

- Compare the performance of the planner using the simple heuristic h_G (counts unsatisfied goals) versus using the FF heuristic on some examples from the prodigy-bw, and some examples from logistics-strips and the examples from the block painting domain that you wrote on Thursday.

```
def h_G(self, node):
    return len(self.task.goals - node.state)
```

- (Optional) Implement Helpful Actions as described in class and in the slides and report the impact in performance.
- (Optional) You should expect that your planner will be substantially slower than the highly optimized planners used in the planning competitions (Blackbox or FF), but you could compare them with your best run times on the examples above. You can get Blackbox here `http://www.cs.rochester.edu/u/kautz/satplan/blackbox/index.html` and FF-X `https://fai.cs.uni-saarland.de/hoffmann/ff.html` here.
- Submit your file main.py with your heuristic implementation and a PDF file with your testing results. Please make sure that you include a description of how you carried out the testing.