Due Date: 12 February 2019 at 09:30 Eastern

Homework 2

CS 4649/7649 Robot Intelligence: Planning
Instructor: Matthew Gombolay

Instructions:

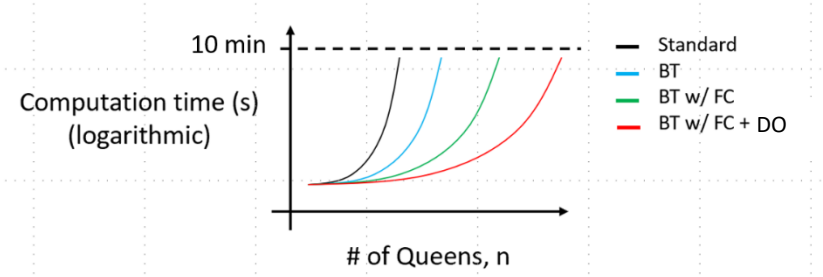
- You may work with one or more classmates on this assignment. However, all work must be your own, original work (i.e., no copy+pasting code). You must list all people you worked with and sources you used on the document you submit for your homework
- Solutions to "by hand" problem must be enclosed by a box and be legible.

Problem 1 (CS 4649: 50 Points; CS 7649 30 Points):

Implement the following search algorithms to solve the n-queens problem, which is defined as the problem of placing n queens on a chess board of size n by n such that no queen can "take" another queen (i.e., no two queens should be on the same row, column or diagonal).

- Standard Search
- Backtrack Search
- Backtrack Search with Forward Checking
- Backtrack Search with Forward Checking and Dynamic Ordering[1]

Generate a plot like the one below by trying n = {1,2,…} for each algorithm until the search method can no longer find a solution within 10 minutes.



[1] Most constrained variable AND least constraining variable

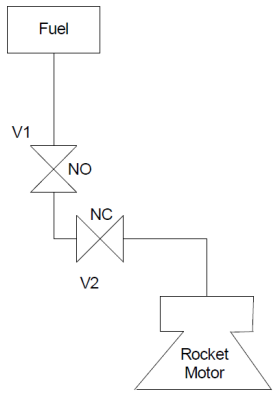
Due Date: 12 February 2019 at 09:30 Eastern

Problem 2 (CS 4649: 50 Points; CS 7649 40 Points):

In this problem you will construct a plan graph for a very simple spacecraft control problem.

A critical stage of many deep space probe missions is orbital insertion. One of the most ambitious simulation-based autonomy demonstrations to date has been robust mission planning, execution and failure recovery for Saturn Orbital Insertion (SOI). During SOI it is essential that the main engine be commanded reliably under failure. In this problem we consider the problem of automatically planning a control sequence for a simple rocket engine system.

Consider an extremely simple rocket engine system, shown at the top of the next page. To fire the engine, fuel must flow to it. Fuel flow is controlled by valves V1 and V2, which are pyrotechnic valves, pyro for short. Pyro valves are initially in one particular state (open or closed). An explosive bolt can be fired that switches a pyro valve to its other state. Thus, an important disadvantage of a pyro valve (with respect to a typical, electrically activated on-off valve) is that a pyro valve can switch states only once. The advantage of using a pyro valve is that it is extremely reliable; it will stay in its initial state until fired. When the valve is fired, it will switch with high probability to its opposite state, where the valve remains. In the following diagram, valve V1 is initially open (indicated by NO for normally open). Firing V1 closes it. Valve V2 is initially closed (NC for normally closed). Firing V2 opens it.



We formulate the problem using the STRIPS plan representation (the STRIPS representation is introduced in the lecture notes and Ch. 11 of AIMA). Our problem is to generate a command sequence that fires the rocket engine, given that initially v2 is closed, v1 is open and the rocket is off. The initial and goal conditions in STRIPS are:

Initial Conditions:
(preconds
(closed-v2)
(open-v1)
(off-rocket))
Goal:
(effects
(rocket-fired)
(fuel-flowing))

Due Date: 12 February 2019 at 09:30 Eastern

We define the operations of firing pyro valves v1 and v2 through the plan operators fire-v1 and fire-v2, given below. Note that fire-v1 moves v1 from open to closed and can only be executed when v2 is open. Likewise, fire-v2 moves v2 from closed to open, and can only be executed when v1 is open:

```
(OPERATOR fire-v1
        (params)
        (preconds (open-v2)
                (open-v1))
        (effects (del open-v1)
                (closed-v1)
                (del fuel-flowing)
                (fuel-not-flowing)))

(OPERATOR fire-v2
        (params)
        (preconds (closed-v2)
                (open-v1))
        (effects (del closed-v2)
                (open-v2)
                (del fuel-not-flowing)
                (fuel-flowing)))


(OPERATOR fire-rocket
      (params)
      (preconds (fuel-flowing)
            (off-rocket))
      (effects (on-rocket)
            (rocket-fired))
```

Part A) Draw the plan graph for this problem, beginning with the initial state, and expanding levels until a level appears that contains all goal variables. Ignore mutex relations for now.

Part B) Draw a plan graph for this problem that includes mutex relations for both actions and variables. Note that this may require adding layers to the graph, relative to Part A. The last level for this graph must include all goal variables with no mutex relations between any of them.

Due Date: 12 February 2019 at 09:30 Eastern

Problem 3 (CS 4649: Bonus 15 Points; CS 7649 30 Points):

Implement* an activity planner that parses in PDDL domain and problem files and returns a solution (i.e., an "activity plan"). When the code you write is placed within a folder on a computer, it should read in two files: A domain file called "domain.pddl" and a problem file called "problem.pddl". Your code should print the solution as a sequence of actions, which are separated by commas. We will supply you with example domain and problem files. You will be graded in part based upon the success of your code in solving a hold-out problem file for one of the two domains we provide as examples. The plan you return must not only be complete and consistent, it must also be optimal.**

*You do not have to implement this parser from scratch. You may use a python parser (e.g., https://pypi.org/project/pddlpy/) so that you do not have to implement the infrastructure from scratch. Further, you may utilize any online solver that you want. **All you must do is make sure that the code you submit parses in the files and returns a complete, consistent, optimal plan.** Your code must run as a single file and cannot require that the TAs install files to get it to work – it should be self-contained.

**Optimal here means that it requires the fewest number of "layers" or time steps.

Problem 4 (7649 Bonus 15 Points; 4649 Bonus 30 Points):

This problem is the same as Problem 3 except that you must implement from scratch the recursive search method described in lecture on 5 February 2019.