AI Planning for Autonomy

Problem Set V: PDDL and Project 1

1. Discuss in your group the heuristics you used in project 1. Are any of them related to the domain independent heuristics we have covered in class?

- What is the (optimal) delete relaxation heuristic h^+? How would it be interpreted in pacman?
- What is the relationship between h^{max}, h^+, and h^{add}? What about h^*?

2. The robot can pick up a block and put it down on another block (or the table) in a single action. You've got actions `Move(Block,FromTable,ToBlock)` and `Move(Block, FromBlock, ToTable)`. Compared to last week, you now no longer need to keep track of what the robot is holding or if the hand is empty.

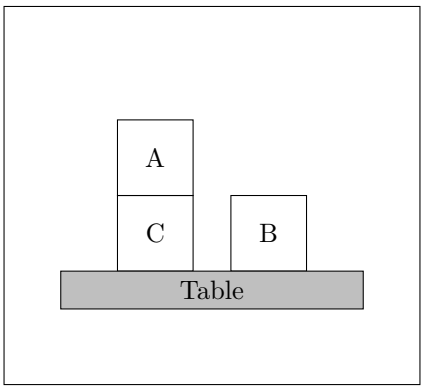

Figure 1: A blocks-world problem.

Compute the values of each of the following heuristics for this problem

- h^{ff}: Use h^{max} for the best-supporters function.
- h^{ff}: Use h^{add} for the best-supporters function.

3. Implement a STRIPS model of this "2-operation" blocks-world in PDDL. Use `http://editor.planning.domains` to write your model. The integrated solver is very limited, so you will want to download docker and install the planners in your computer by typing the command: *docker pull lapkt/lapkt-public*

See `https://hub.docker.com/r/lapkt/lapkt-public/` for more instructions on how to run the available planners

The example TSP of Australia from lectures is implemented in PDDL in the figures below. It is also accessible in editor.planning.domains so you can try the solver on the cloud, and edit the TSP: editor.planning.domains link with Domain and Problem file

See `http://www.hakank.org/pddl/` for more examples.

```
(define (domain tsp)
   (:requirements :typing)
   (:types node)
   ;; Define the facts in the problem
   ;; "?" denotes a variable, "-" a type
   (:predicates (move ?from ?to - node)
                (at ?pos - node)
                (connected ?start ?end - node)
                (visited ?end - node))
   ;; Define the action(s)
   (:action move
             :parameters (?start ?end - node)
             :precondition (and (at ?start)
                                (connected ?start ?end))
             :effect  (and (at ?end)
                           (visited ?end)
                           (not (at ?start)))))
```

Figure 2: tsp-domain.pddl

```
(define (problem tsp-01)
   (:domain tsp)
   (:objects Sydney Adelade Brisbane Perth Darwin - node)
   ;; Define the initial situation
   (:init (connected Sydney Brisbane)
          (connected Brisbane Sydney)
          (connected Adelade Sydney)
          (connected Sydney Adelade)
          (connected Adelade Perth)
          (connected Perth Adelade)
          (connected Adelade Darwin)
          (connected Darwin Adelade)
          (at Sydney))
   (:goal
       (and (at Sydney)
            (visited Sydney)
            (visited Adelade)
            (visited Brisbane)
            (visited Perth)
            (visited Darwin))))
```

Figure 3: tsp-problem.pddl