Software Agents

Problem Set VI: MIndiGolog and PDDL

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 critical path heuristic h^m? How would it be interpreted in pacman how does it change with m?
- What is the relationship between h^1 and h^{max}?
- What is the relationship between h^{max}, h^+, and h^{add}? What about h^*?

2. Simplified blocks-world in Golog:

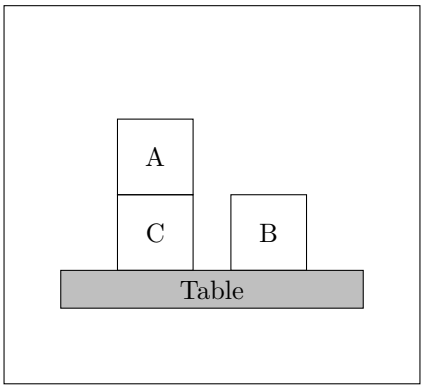

Figure 1: A blocks-world problem.

The robot has two actions

- $PutOn(x, y)$ - which picks up block x and puts it on top of block y
- $PutOnTable(x)$ - which picks up block x and puts it on the table

There are two fluents

- $On(x, y, s)$ - block x is on block y in situation s
- $OnTable(x, s)$ - block x is on the table in situation s

Write down: the initial conditions; the effect axioms; and the precondition axioms. Transform your effect axioms from into successor state axioms, then implement this action theory in prolog. (homework)

3. Implement a STRIPS model of this "2-operation" blocks-world in PDDL. Use Metric-FF to test your model http://fai.cs.uni-saarland.de/hoffmann/metric-ff.html this solver is available on the department machines at /home/subjects/482/local/project/ff

The example TSP of Australia from Nir's lectures is implemented in PDDL below.

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