# 6.825 Exercise Problems

Weeks 1 and 2 Solutions

September 23, 2004

## 1 A Problem-Solving Problem

A Mars rover has to leave the lander, collect rock samples from three places (in any order) and return to the lander.

Assume that it has a navigation module that can take it directly to places of interest. So it has primitive actions *go-to-lander*, *go-to-communications-location*, *go-to-rock-1*, *go-to-rock-2*, and *go-to-rock-3*.

We know the time it takes to traverse between each pair of special locations. Our goal is to find a sequence of actions that will perform this task in the shortest amount of time.

1. (10 points) Formulate this problem as a problem-solving problem by specifying the search space, initial state, path-cost function, and goal test. Assume that the world dynamics are deterministic.

   Answer: Search space: `current-location` x `have-rock-1?` x `have-rock-2?` x `have-rock-3?`
   Initial state: (*lander*,0,0,0) Path cost: Sum of times taken on legs of trip Goal test: Need to be in state (*lander*,1,1,1)

2. (5 points) Say what search technique would be most appropriate, and why. If your search technique requires a heuristic, give an appropriate one.

   Answer: Use `A*` search with the heuristic function $h(n)$ = `(Number of rocks uncollected + 1)` x `shortest-leg-time`.

3. (5 points) Now assume that, in addition, on every day that it is not at the lander, the rover needs to be at a special communications location at 3PM. Any plan that misses a communications rendezvouz is unsatisfactory. How would you modify the search space, path-cost function, and/or goal-test to handle this additional requirement? Assume that there's an additional action, called *communicate*, which can only be executed when the rover is at the communications location, and which waits until 3PM, and then executes a communications sequence that takes an hour.

   Answer: The new state-space would have to include time as well as a bit specifying whether or not a communications requirement had been missed. This bit should get set whenever an action other than waiting at the communications point takes the robot over the 3PM boundary. Thus, we get `current-location` x `have-rock-1?` x `have-rock-2?` x `have-rock-3?` x `time` x `missed-com-requirement?`
   The path cost is still the sum of times, and the goal test is (*lander*,1,1,1,*,0)

## 2 AIMA Problem 2.4

**Let us examine the rationality of various vacuum-cleaner agent functions.**

1. **Show that the simple vacuum-cleaner function described in Figure 2.3 is indeed rational under the assumptions listed on page 36.**

   To be rational in these circumstances it maximizes its performance measure 1 pt per each clean square.

   If there is dirt here, suck gains you a sure point now and doesn't impact your chances of getting other dirt in the future, so suck. If There's no dirt here, the only possible way of getting another point in the future is to move to the other square.

2. **Describe a rational agent function for the modified performance measure that deducts one point for each movement. Does the corresponding agent program require internal state?**

   This objective function doesn't require memory even though the problem is partial order. But what if you have to pay 1 for each move or such action? Then you have to remember where you are.

3. **Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn?**

   In the absence of special distributional assumptions, our first policy is probably best. Though if we could learn the rates at which the individual squares become dirty and adjust our policy accordingly, that would be cool.

# 3 Propositional Logic

Which of these are legal sentences? Give fully parenthesized expressions.

- $P \rightarrow Q \rightarrow R$ Answer: $(P \rightarrow Q) \rightarrow R$
- $P, R \rightarrow Q$ Answer: **Not legal**
- $A \wedge (B \vee C \vee \neg D) \leftrightarrow \neg\neg Z$ Answer: $(A \wedge (B \vee C \vee (\neg D))) \leftrightarrow (\neg(\neg Z))$
- $\neg P(Q)$ Answer: **Not legal in propositional logic**

# 4 DPLL

1. How would you modify DPLL so that it
   - returns a satisfying assignment if there is one, and false otherwise?
     Answer: Every time you assign a value to a variable, keep track of that assignment, and return all assignments when returning *true*.
   - returns *all* satisfying assignments?
     Answer: Make two changes. First, remove the pure literal rule, and second, whenever a call to DPLL made immediately after setting a variable to *true* returns *true*, don't just stop, but instead store the assignment and call DPLL with that variable set to *false*.
2. Would using DPLL to return all satisfying assignments be any more efficient that simply listing all the assignments and checking to see whether they're satisfying? Why or why not?
   Answer: Yes, it would be more efficient, because DPLL applies unit clause rules to make forced choices and then simplifies the sentence to prune out the search space.

# 5 State Spaces and Search

A robot has to deliver identical packages to locations A, B, and C, in an office environment. Assume it starts off holding all three packages. The environment is represented as a grid of squares, some of which are free (so the robot can move into them) and some of which are occupied (by walls, doors, etc.). The robot can move into neighboring squares, and can pick up and drop packages if they are in the same square as the robot.

1. Formulate this problem as a search problem, specifying the state space, action space, goal test, and cost function.

   **Solution:**

   *The state space needs to include enough information so that, by looking at the current values of the state features, the robot knows what it needs to do. For this task, the robot needs to be able to determine its position in the grid and which packages it has already delivered.*

- State: { (x,y), deliveredA, deliveredB, deliveredC }
- Action space: MoveN, MoveE, MoveS, MoveW, DropA, DropB, DropC.
- Goal test: { (x,y), delA, delB, delC } = { (any x, any y), 1, 1, 1 }
- Cost function: cost of 1 for each action taken.

**Common Mistakes:**

- Including only the number of packages in the state space: this is insufficient because, if we know we are holding 2 packages, we don't know if, for example, we've already been to A, or if we should try to go there next.
- Specifying a goal test that couldn't be derived from the state space that was described.

2. Give a non-trivial, polynomial-time, admissible heuristic for this domain, or argue that there isn't one.

   **Solution:**

   *There are many acceptable solutions. Some examples:*

   - Straight line or manhattan distance to furthest unvisited drop-off location.
   - Straight line or manhattan distance to nearest unvisited drop-off location. (slightly looser underestimate than the previous one)
   - Number of packages left to deliver. (Not as "good" as the above two, but still acceptable.)

   **Common Mistakes:**

   - Trying to solve the Travelling Salesman Problem: In the general case, not poly in the number of drop-off locations.
   - Summing the distances between current robot location and all unvisited drop-off locations: Can overestimate the distance.
   - Summing up the perimeter of the convex hull of robot location and drop-off locations: Can overestimate the distance.

3. Package A has to be delivered to the boss, and it's important that it be done quickly. But we should deliver the other packages promptly as well. How could we encode this in the problem?

   **Solution:**

   *The idea of putting "pressure" on the delivering to the boss is correctly handled in the cost function. We want to emphasize delivery to A, but we don't want to ignore B if we happen to pass by it. Please note that to encode something into the "problem," it must be encoded into one of the problem components: the state space, action space, goal test, or cost function. For example:*

   while we haven't delivered to A, actions cost 2; after that, actions cost 1.

   **Common Mistakes:**

   - Decreasing the cost instead of increasing the cost.
   - Constraining the order of the problem; requiring delivery to A before considering the other two locations at all.
   - Modifying the search heuristic instead of the cost function.

4. Now, consider the case where the robot doesn't start with the packages, but it has to pick up a package from location 1 to deliver to location A, a package from location 2 to deliver to B, and from 3 to deliver to C. What is an appropriate state space for this problem?

   **Solution:**

   *Again, multiple solutions are possible. One example:*

   - State: { (x,y), deliveredA, deliveredB deliveredC, holdingA, holdingB, holdingC }

   **Common Mistakes:**

   - Again, just the number of packages and the number of visited locations are insufficient.

5. What is a good admissible heuristic?

   **Solution:**

   - The same heuristic as before. (Not so "good," but acceptable.)
   - The maximum (distance from my location to unvisited pickup location $s_i$ plus the distance from $s_i$ to unvisited delivery location).