Advanced Search Exercises

4.1: Local Search Algorithms and Optimization Problems (4 exercises, 1 labelled)

Q. Give the name of the algorithm that results from each of the following special cases:
a. Local beam search with k = 1. b. Local beam search with one initial state and no limit on the number of states retained. c. Simulated annealing with T = 0 at all times (and omitting the termination test). d. Simulated annealing with T = ∞ at all times. e. Genetic algorithm with population size N = 1. (id=4.0 section=4.1)

Q. Exercise 3.16(will insert a link later) considers the problem of building railway tracks under the assumption that pieces fit exactly with no slack. Now consider the real problem, in which pieces don't fit exactly but allow for up to 10 degrees of rotation to either side of the "proper" alignment. Explain how to formulate the problem so it could be solved by simulated annealing. (id=4.1 section=4.1.2)

Q. In this exercise, we explore the use of local search methods to solve TSPs of the type defined in Exercise 3.30 (will insert link later). a. Implement and test a hill-climbing method to solve TSPs. Compare the results with op- timal solutions obtained from the A^* algorithm with the MST heuristic (Exercise 3.30). b. Repeat part (a) using a genetic algorithm instead of hill climbing. You may want to consult Larran˜aga et al. (1999) for some suggestions for representations. (id=4.10 section=4.1)

Q. *[hill-climbing-exercise]%*: Generate a large number of 8-puzzle and 8-queens instances and solve them (where pos- sible) by hill climbing (steepest-ascent and first-choice variants), hill climbing with random restart, and simulated annealing. Measure the search cost and percentage of solved problems and graph these against the optimal solution cost. Comment on your results (id=4.11 section=4.1)

4.3: Searching with Nondeterministic Actions (2 exercises, 2 labelled)

Q. *[cond-plan-repeated-exercise]*: The AND-OR-GRAPH-SEARCH algorithm in :

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then return [action | plan]
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

checks for repeated states only on the path from the root to the current state. Suppose that, in addition, the algorithm were to store every visited state and check against that list. (See BREADTH-FIRST-SEARCH in Figure 3.11 from book for an example.) Determine the information that should be stored and how the algorithm should use that information when a repeated state is found. (Hint: You will need to distinguish at least between states for which a successful subplan was constructed previously and states for which no subplan could be found.) Explain how to use labels, as defined in Section 4.3.3, to avoid having multiple copies of subplans. (id=4.2 section=4.3.2)

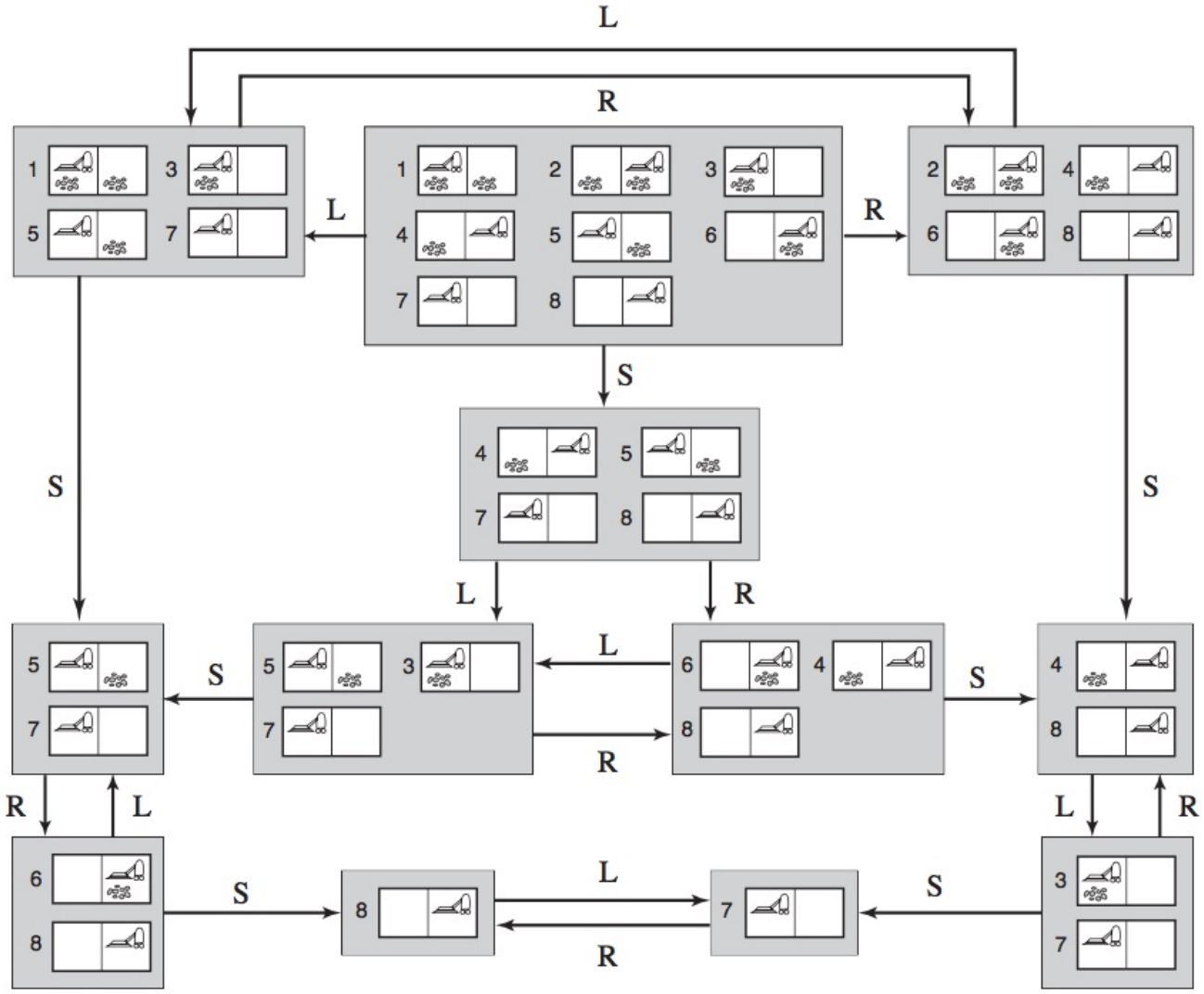
Q. *[cond-loop-exercise]*: Explain precisely how to modify the AND-OR-GRAPH-SEARCH algorithm to generate a cyclic plan if no acyclic plan exists. You will need to deal with three issues: labeling the plan steps so that a cyclic plan can point back to an earlier part of the plan, modifying OR-SEARCH so that it continues to look for acyclic plans after finding a cyclic plan, and augmenting the plan representation to indicate whether a plan is cyclic. Show how your algorithm works on (a) the slippery vacuum world, and (b) the slippery, erratic vacuum world. You might wish to use a computer implementation to check your results (id=4.3 section=4.3.3)

4.4: Searching with Partial Observations (5 exercises, 4 labelled)

Q. In Section 4.4.1 we introduced belief states to solve sensorless search problems. A sequence of actions solves a sensorless problem if it maps every physical state in the initial belief state *b* to a goal state. Suppose the agent knows

$$(h^*(s))$$

, the true optimal cost of solving the physical state *s* in the fully observable problem, for every state *s* in *b*. Find an admissible heuristic *h(b)* for the sensorless problem in terms of these costs, and prove its admissibilty. Comment on the accuracy of this heuristic on the sensorless vacuum problem of :



How well does :

$$A^*$$

perform? (id=4.4 section=4.4.1)

Q. *[belief-state-superset-exercise]* : This exercise explores subset–superset relations between belief states in sensorless or partially observable environments. a. Prove that if an action sequence is a solution for a belief state b, it is also a solution for any subset of b. Can anything be said about supersets of b? b. Explain in detail how to modify graph search for sensorless problems to take advantage of your answers in (a). c. Explain in detail how to modify AND–OR search for partially observable problems, beyond the modifications you describe in (b). (id=4.5 section=4.4)

Q. *[multivalued-sensorless-exercise]* : On page 139 it was assumed that a given action would have the same cost when ex- ecuted in any physical state within a given belief state. (This leads to a belief-state search problem with well-defined step costs.) Now consider what happens when the assumption does not hold. Does the notion of optimality still make sense in this context, or does it require modification? Consider also various possible definitions of the “cost” of executing an action in a belief state; for example, we could use the *minimum* of the physical costs; or the *maximum*; or a cost *interval* with the lower bound being the minimum cost and the upper bound being the maximum; or just keep the set of all possible costs for that action. For each of these, explore whether

$$A^*$$

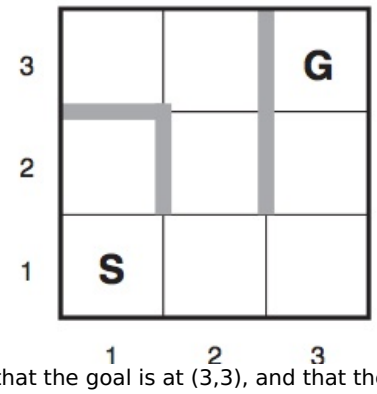
(with modifications if necessary) can return optimal solutions (id=4.6 section=4.4)

Q. *[vacuum-solvable-exercise]* : Consider the sensorless version of the erratic vacuum world. Draw the belief-state space reachable from the initial belief state {1, 2, 3, 4, 5, 6, 7, 8}, and explain why the problem is unsolvable. (id=4.7 section=4.4)

Q. *[vacuum-solvable-exercise]* : Consider the sensorless version of the erratic vacuum world . Draw the belief-state space reachable from the initial belief state N{1,3,5,7} , and explain why the problem is unsolvable. (id=4.7 section=4.4)

4.5: Online Search Agents and Unknown Environments (4 exercises, 2 labelled)

Q. *[online-offline-exercise]* : Suppose that an agent is in a 3×3 maze environment like the one shown in Figure :



The agent knows that its initial location is (1,1), that the goal is at (3,3), and that the actions *Up, Down, Left, Right* have their usual effects unless blocked by a wall. The agent does not know where the internal walls are. In any given state, the agent perceives the set of legal actions; it can also tell whether the state is one it has visited before. a. Explain how this online search problem can be viewed as an offline search in belief-state space, where the initial belief state includes all possible environment configurations. How large is the initial belief state? How large is the space of belief states? b. How many distinct percepts are possible in the initial state? c. Describe the first few branches of a contingency plan for this problem. How large (roughly) is the complete plan? Notice that this contingency plan is a solution for *every possible environment* fitting the given description. Therefore, interleaving of search and execution is not strictly necessary even in unknown environments. (id=4.8 section=4.5.1)

Q. *[path-planning-agent-exercise]* : We can turn the navigation problem in Exercise 3.7 into an environment as follows:

- The percept will be a list of the positions, relative to the agent, of the visible vertices. The percept does not include the position of the robot! The robot must learn its own po- sition from the map; for now, you can assume that each location has a different “view.”
- Each action will be a vector describing a straight-line path to follow. If the path is unobstructed, the action succeeds; otherwise, the robot stops at the point where its path first intersects an obstacle. If the agent returns a zero motion vector and is at the goal (which is fixed and known), then the environment teleports the agent to a random location (not inside an obstacle).
- The performance measure charges the agent 1 point for each unit of distance traversed and awards 1000 points each time the goal is reached.

  1) Implement this environment and a problem-solving agent for it. After each teleporta- tion, the agent will need to formulate a new problem, which will involve discovering its current location.

  2) Document your agent’s performance (by having the agent generate suitable commentary as it moves around) and report its performance over 100 episodes.

  3) Modify the environment so that 30% of the time the agent ends up at an unintended destination (chosen randomly from the other visible vertices if any; otherwise, no move at all). This is a crude model of the motion errors of a real robot. Modify the agent so that when such an error is detected, it finds out where it is and then constructs a plan to get back to where it was and resume the old plan. Remember that sometimes getting back to where it was might also fail! Show an example of the agent successfully overcoming two successive motion errors and still reaching the goal.

  4) Now try two different recovery schemes after an error: (a) head for the closest vertex on the original route; and (b) replan a route to the goal from the new location. Compare the performance of the three recovery schemes. Would the inclusion of search costs affect the comparison?

  5) Now suppose that there are locations from which the view is identical. (For example, suppose the world is a grid with square obstacles.) What kind of problem does the agent now face? What do solutions look like?

(id=4.9 section=4.4)

Q. *[path-planning-hc-exercise]* : In this exercise, we examine hill climbing in the context of robot navigation, using the environment in figure :

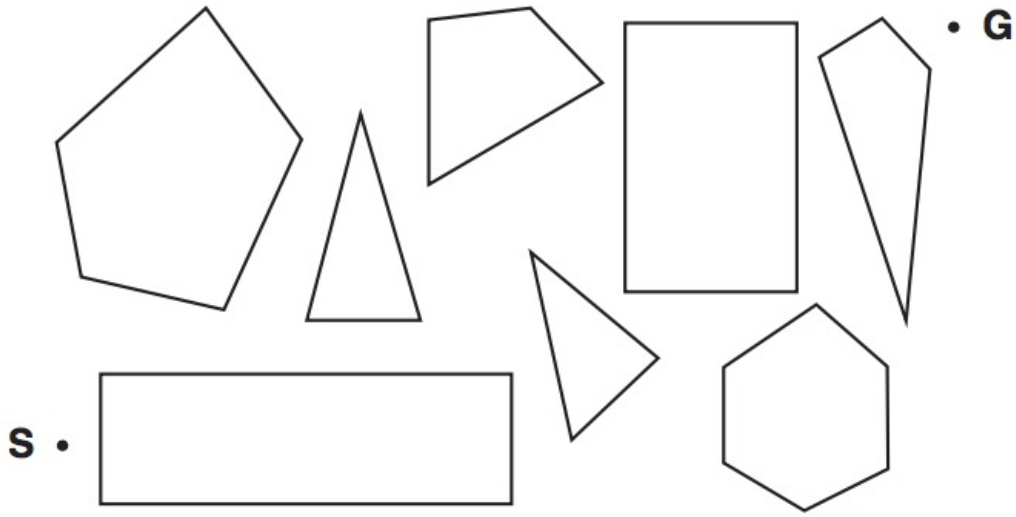


Figure 3.31 A scene with polygonal obstacles. S and G are the start and goal states.

as an example. a. Repeat above Exercise using hill climbing. Does your agent ever get stuck in a local minimum? Is it *possible* for it to get stuck with convex obstacles? b. Construct a nonconvex polygonal environment in which the agent gets stuck. c. Modify the hill-climbing algorithm so that, instead of doing a depth-1 search to decide where to go next, it does a depth-*k* search. It should find the best *k*-step path and do one step along it, and then repeat the process. d. Is there some k for which the new algorithm is guaranteed to escape from local minima? e. Explain how

$$LRTA^*$$

enables the agent to escape from local minima in this case. (id=4.12 section=4.5)

Q. Like DFS, online DFS is incomplete for reversible state spaces with infinite paths. For example, suppose that states are points on the infinite two-dimensional grid and actions are unit vectors (1, 0), (0, 1), (−1, 0), (0, −1), tried in that order. Show that online DFS starting at (0, 0) will not reach (1, −1). Suppose the agent can observe, in addition to its current state, all successor states and the actions that would lead to them. Write an algorithm that is complete even for bidirected state spaces with infinite paths. What states does it visit in reaching (1, −1)? (id=4.13 section=4.5)

Q. Relate the time complexity of

$$LRTA^*$$

to its space complexity. (id=4.14 section=4.5)

Q. *{safe-ratio-exercise}* Extend the state spaces in

Figure 3.8 A sequence of search trees generated by a graph search on the Romania problem of Figure 3.2. At each stage, we have extended each path by one step. Notice that at the third stage, the northernmost city (Oradea) has become a dead end: both of its successors are already explored via other paths.

to make them safely explorable, and prove that no bounded competitive ratio can be guaranteed in safely explorable environments.