CPS 170: Introduction to AI **Instructor:** Ron Parr

Homework 1 **Due:** Thursday February 4, 2010

Note: Many of these problems are from the textbook or are modified versions of textbook problems, so credit goes to the textbook authors for developing these problems.

1 Search

Consider the problem of finding the shortest path between two points on a plane that has convex polygonal obstacles as shown in Figure 3.31 (3.22 in the second edition). This is an idealization of the problem that a robot has to solve to navigate in a crowded environment.

For this problem, you will need to do a small implementation. We have provided some functions to help you out in the class directory under `hw1code.zip`. These functions are in Java, but you are not required to use Java.

(a) Suppose the state space consists of all positions (x, y) in the plane. How many states are there? How many paths are there to the goal?

(b) Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. How large is this state space?

(c) Define the necessary functions to read in a navigation problem as a set of polygons (described as a list of vertices), a starting point, and a goal (using the format described in `readme.txt`). Implement a successor function that takes a vertex as input and returns a set of vectors, each of which maps the current vertex to one of the vertices that can be reached in a straight line. (Do not forget the neighbors on the same polygon.) You will also need to implement a goal test, but that should be trivial.

(d) Implement A* search using the straight line distances as your heuristic function. Test your code on the provided `polys.txt` problem, and one other problem of your creation. Show the shortest path returned by your algorithm in some visual form (preferrably using the functions we have provided). Note that this problem is about A* and not about priority queues. You do not need to re-implement a priority queue from scratch; you may use code in standard libraries for this.

2 Search

In this problem we explore searching in the presence of negative path costs.

(a) Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.

(b) Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both trees and graphs.

(c) Suppose that a set of actions forms a loop in the state space such that executing the set in some order results in no net change to the state. If all of these actions have negative cost, what does this imply about the optimal behavior for an agent in such an environment?

(d) One can easily imagine actions with high negative cost, even in domains such as route finding. For example, some stretches of read might have such beautiful scenery as to far outweigh the normal costs in terms of time and fuel. Explain, in precise terms, within the context of state-space search, why humans do not drive around scenic loops indefinitely, and explain how to define the state space and actions for route finding so that artificial agents can also avoid looping.

3 Search

(a) Prove that if a heuristic is consistent, it must be admissible.

(b) Construct an admissible heuristic that is not consistent.

4 Adversarial Search

Prove the following assertion: For every game tree, the utility obtained by MAX using minimax decisions against a suboptimal MIN will never be lower than the utility obtained playing against an optimal MIN. Can you come up with a game tree in which MAX can do still better using a *suboptimal* strategy against a suboptimal MIN?

5 Adversarial Search

Consider the game tree in figure 5.2 (6.2 in the second edition), but with the maxes and mins swapped – assume the root is a min node and second level nodes are maxes.

(a) Indicate which branches will be pruned by alpha-beta search if the nodes are expanded in left-to-right order.

(b) Suppose you change the order of sibling nodes at each level of the tree. Provide an ordering that *maximizes* the amount of pruning alpha-beta will do, and show which nodes are pruned.

(c) Suppose you change the order of sibling nodes at each level of the tree. Provide an ordering that *minimizes* the amount of pruning alpha-beta will do, and show which nodes are pruned.

6 Adversarial Search

Develop a formal proof of correctness for alpha-beta pruning. To do this, consider the situation shown in Figure 5.18 (6.15 in the second edition). The question is whether to prune node n_j, which is a max-node and a descendant of node n_1. The basic idea is to prune it if and only if the minimax value of n_1 can be shown to be independent of the value of n_j.

(a) Node n_1 takes on the minimum value among its children: $n_1 = min(n_2, n_{21}, ..., n_{2b_2})$. Find a similar expression for n_2 and hence an expression for n_1 in terms of n_j.

(b) Let l_i be the minimum (or maximum) value of the nodes to the *left* of node n_i at depth i, whose minimax value is already known. Similarly, let r_i be the minimum (or maximum) value of the unexplored nodes to the right of n_i at depth i. Rewrite your expression for n_1 in terms of the l_i and r_i values.

(c) Now reformulate the expression to show that in order to affect n_1, n_j must not exceed a certain bound derived from the l_i values.

(d) Repeat the process for the case where n_j is a min-node.