Exercises for Artificial Intelligence I

Dr Sean B Holden, 2010-20

1 Introduction

These notes provide some extra exercises for *Artificial Intelligence I* and, depending on when you download them, their solutions.

2 Introduction and Agents

It is notoriously difficult to predict what will be possible in the future, so your answers might well be amusing to you when you find them in twenty years time.

1. If you haven't seen it already, watch the film *A.I. Artificial Intelligence* paying particular attention to the character *"Teddy"*.
2. A large number of subjects are mentioned in the initial lectures in terms of how they've influenced AI: for example philosophy, mathematics, economics and so on. How do these show up in Teddy's design?
3. What aspects of Teddy are within our current capabilities to design?
4. What aspects of Teddy would you expect to be able to implement within the next fifteen years. How about the next fifty years?
5. Are there aspects of Teddy that you would expect to elude us for one hundred years or more?
6. To what extent does the "natural basic structure" for an agent, as described in the lectures, form a useful basis for implementing Teddy's internals? What is missing?

3 Search

1. Explain why breadth-first search is optimal if path-cost is a non-decreasing function of node-depth.
2. In the graph search algorithm, assume a node is taken from the fringe and found *not* to be a goal and *not* to be in `closed`. We then add it to `closed` and add its descendants to `fringe`. Why do we *not* check the descendants first to see if they are in `closed`?
3. Iterative deepening depends on the fact that *the vast majority of the nodes in a tree are in the bottom level.*
   - Denote by $f_1(b, d)$ the total number of nodes appearing in a tree with branching factor b and depth d. Find a closed-form expression for $f_1(b, d)$.
   - Denote by $f_2(b, d)$ the total number of nodes generated in a complete iterative deepening search to depth d of a tree having branching factor b. Find a closed-form expression for $f_2(b, d)$ in terms of $f_1(b, d)$.
   - How do $f_1(b, d)$ and $f_2(b, d)$ compare when b is large?

4. The $A^\star$ algorithm does not perform a goal test on any state *until it has selected it for expansion.* We might consider a slightly different approach: namely, each time a node is expanded check all of its descendants to see if they include a goal.

   Give two reasons why this is a misguided idea, where possible illustrating your answer using a specific example of a search tree for which it would be problematic.

5. The f-cost is defined in the usual way as

$$f(n) = p(n) + h(n)$$

   where n is any node, p denotes path cost and h denotes the heuristic. An admissible heuristic is one for which, for any n

$$h(n) \leq \text{actual distance from } n \text{ to the goal}$$

   and a heuristic is monotonic if for consecutive nodes n and n' it is always the case that

$$f(n') \geq f(n).$$

   - Prove that h is monotonic if and only if it obeys the triangle inequality, which states that for any consecutive nodes n and n'

$$h(n) \leq c_{n \to n'} + h(n')$$

     where $c_{n \to n'}$ is the cost of moving from n to n'.
   - Prove that if a heuristic is monotonic then it is also admissible.
   - Is the converse true? (That is, are all admissible heuristics also monotonic?) Either prove that this is the case or provide a counterexample.

6. In RBFS we are replacing f values every time we backtrack to explore the current best alternative. This seems to imply a need to remember the new f values for all the nodes in the path we're discarding, and this in turn suggests a potentially exponential memory requirement. Why is this not the case?

7. In some problems we can simultaneously search:

   - *forward* from the *start* state
   - *backward* from the *goal* state

   until the searches meet. This seems like it might be a very good idea:

   - If the search methods have complexity $O(b^d)$ then...
   - ...we are converting this to $O(2b^{d/2}) = O(b^{d/2})$.

   (Here, we are assuming the branching factor is b in both directions.) Why might this not be as effective as it seems?

8. Suggest a method for performing depth-first search using only $O(d)$ space.

9. One modification to the basic local search algorithm suggested in the lectures was to make steps probabilistically, but only if the value of f is *improved.*

   (a) Would it be a good idea to also allow steps that move to a state with a *worse* value for f?

   (b) Suggest an algorithm for achieving this, such that you have some control over the balance between steps that increase or decrease f.