Artificial Intelligence: Supervision 1

Daniel Chatfield

May 8, 2015

Search

1. Explain why breadth-first search is optimal if path-cost is a non-decreasing function of node-depth.

   If path-cost is a non-decreasing function of node-depth then, by definition, the path-cost of reaching a node at depth $k+1$ is greater than the path-cost of reaching a node at depth k. It is therefore optimal to search all nodes at each depth first.

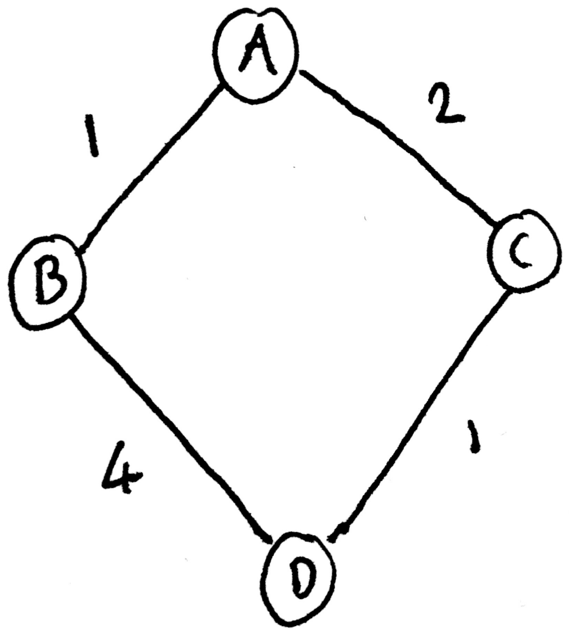
2. In the graph search algorithm, assume a node is taken from the `fringe` and found *not* to be a goal and *not* to be in `closed`. We then add it to `closed` and add its descendants to `fringe`. Why do we not check the descendants first to see if they are in `closed`?

   They might not be in `closed` when they are added to `fringe` but may be added before the node is taken from the queue. It would therefore still be necessary to check when removing from the queue so it hasn't saved you anything.

3. The A^* 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.

   Running a goal test on descendants each time a node is expanded stops the algorithm from being optimal.



In this example, if searching for an optimal path from A to D, B will be expanded. If a goal test is performed on its descendants then the algorithm will terminate since D is a descendant despite the fact that this is not the optimal path.

Not sure about other reason

4. 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

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

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

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

$$\begin{aligned} f(n) &= p(n) + h(n) \\ f(n') &= p(n') + h(n') = p(n) + c_{n \to n'} + h(n') \end{aligned}$$

If the triangle inequality is satisfied we have:

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

and thus we have:

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

Therefore:

$$f(n) \leq f(n')$$

The reverse implication is similar, starting with $f(n) \leq f(n')$, expand out and cancel the $p(n)$ to be left with the triangle inequality.

(b) Prove that if a heuristic is monotonic then it is also admissible.

There are two cases, either the goal is reachable or it isn't.

Case where goal is unreachable In this case the actual distance is infinite and thus anything is admissible since it is not possible to overestimate the distance.

Case where goal is reachable If the heuristic is monotonic then we have:

$$f(n) \leq f(n')$$

Expanding this gives us

$$p(n) + h(n) \leq p(n') + h(n') \tag{1}$$
$$p(n) + h(n) \leq p(n) + c_{n \to n'} + h(n') \tag{2}$$
$$h(n) \leq c_{n \to n'} + h(n') \tag{3}$$

Since $h(n_{goal})$ is defined to be zero, for the node preceding the goal in the path we have:

$$h(n) \leq c_{n \to n_{goal}} \tag{4}$$

Using (4) as the base case and performing induction using (3), it can be seen that (4) holds for any node n. This inequality is the requirement for admissibility.

(c) Is the converse true? (That is, are all admissible heuristics also monotonic?) Either prove that this is the case or provide a counterexample.

The converse is not true, a counterexample is a tree with 3 nodes, X, Y and Z. The path costs are $C_{X \to Y} = 1$ and $C_{Y \to Z} = 2$. Let $h(X) = 3$ and $h(Y) = 1$.

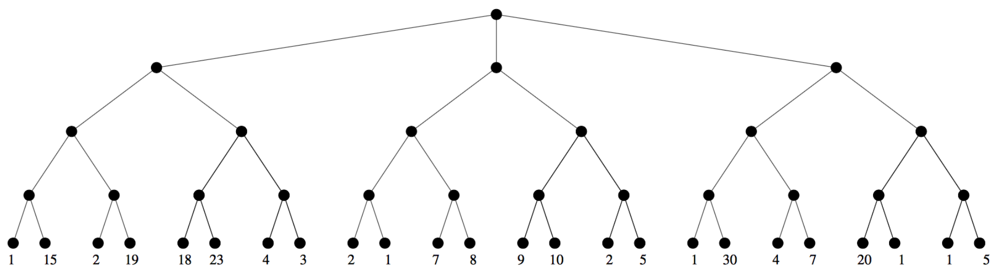
The heuristic is admissible since it never overestimates the distance, however it is not monotonic as $f(X) = 3$ and $f(Y) = 2$, and thus $f(n) \leq f(n')$ does not hold.

5. 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?

Not sure

Games

1. Consider the following game tree:



Large outcomes are beneficial for Max. How is this tree pruned by $\alpha - \beta$ minimax if Max moves first? (That is, Max is the root.) How is it pruned if Min is the root, and therefore moves first?

I'm labelling sub trees using M, L and R to indicate middle, left and right respectively.

The pruned subtrees are:

- LR
- CLL
- CR
- RLL
- RLR
- R

2. Implement the $\alpha - \beta$ pruning algorithm and use it to verify your answer to the previous problem.

3. Is the minimax approach to playing games optimal against an imperfect opponent? Either prove this is the case or give a counterexample.

No it isn't, assume that the opponent always chooses the opposite of what it should. The following tree would result in a non-optimal result.

```
    / \
   /   \
  /\  / \
 1 10 2  3
```