CS 188 Spring 2021 | Introduction to Artificial Intelligence

Exam Prep 2 Solutions

Q1. Search

(a) Rubik's Search

Note: You do not need to know what a Rubik's cube is in order to solve this problem.

A Rubik's cube has about 4.3×10^{19} possible configurations, but any configuration can be solved in 20 moves or less. We pose the problem of solving a Rubik's cube as a search problem, where the states are the possible configurations, and there is an edge between two states if we can get from one state to another in a single move. Thus, we have 4.3×10^{19} states. Each edge has cost 1. Note that the state space graph does contain cycles. Since we can make 27 moves from each state, the branching factor is 27. Since any configuration can be solved in 20 moves or less, we have $h^*(n) \leq 20$.

For each of the following searches, estimate the approximate number of states expanded. Mark the option that is closest to the number of states expanded by the search. Assume that the shortest solution for our start state takes exactly 20 moves. Note that 27^{20} is much larger than 4.3×10^{19}.

(i) DFS Tree Search

Best Case: ● 20 ○ 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Worst Case: ○ 20 ○ 4.3×10^{19} ○ 27^{20} ● ∞ (never finishes)

In the best case, we would expand states in the perfect sequence of actions to solve the Rubik's cube in the minimum step count of 20 moves. In the worst case, we could get stuck expanding the same states repeatedly in an infinite loop, because with tree search we are allowed to expand the same state multiple times.

(ii) DFS graph search

Best Case: ● 20 ○ 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Worst Case: ○ 20 ● 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

In the best case, we would expand states in the perfect sequence of actions to solve the Rubik's cube in the minimum step count of 20 moves (same as in DFS tree search). In the worst case, we would expand every possible configuration of the Rubik's cube without repeating states.

(iii) BFS graph search

Best Case: ○ 20 ● 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Worst Case: ○ 20 ● 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Because we are given that the shortest solution from our starting state is exactly 20 moves away, we end up having to explore approximately the same number of configurations that the Rubik's cube can possibly have. There is no difference between the best and worst case because of how BFS explores level by level.

(iv) A* tree search with a perfect heuristic, $h^*(n)$, Best Case

● 20 ○ 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Since each edge has cost 1, a perfect heuristic will tell us exactly the sequence of actions needed to get to the goal in the fewest number of moves.

(v) A* tree search with a bad heuristic, $h(n) = 20 - h^*(n)$, Worst Case

○ 20 ○ 4.3×10^{19} ● 27^{20} ○ ∞ (never finishes)

The bad heuristic will make it so we end up exploring many, many configurations of the cube, before eventually getting to explore the goal state. The reason that we don't search infinitely is because as we explore repeated states, we increase $g(n)$, the backward cost to reach that state, because the cost of an action is 1. This means that the amount of nodes with f-value less than $f(G)$ is finite, and so we will eventually explore the goal.

(vi) A* graph search with a perfect heuristic, $h^*(n)$, Best Case

● 20 ○ 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Same as in A* tree search; since each edge has cost 1, a perfect heuristic will tell us exactly the sequence of actions needed to get to the goal in the fewest number of moves.

(vii) A* graph search with a bad heuristic, $h(n) = 20 - h^*(n)$, Worst Case

○ 20 ● 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

Since we keep track of expanded states in graph search, even though this bad heuristic makes us expand further states first, we end up eventually able to explore a goal state after exploring all other states first. This approximates to all unique configurations of the Rubik's cube.

Q2. Searching with Heuristics

Consider the A* searching process on a connected undirected graph, with starting node S and the goal node G. Suppose the cost for each connection edge is **always positive**. We define $h^*(X)$ as the shortest (optimal) distance to G from a node X.

Note: You may want to solve Questions (a) and (b) at the same time.

(a) Suppose h is an **admissible** heuristic, and we conduct A* **tree search** using heuristic h' and finally find a solution. Let C be the cost of the found path (directed by h', defined in part (a)) from S to G.

(i) Choose **one best** answer for each condition below.

1. If $h'(X) = \frac{1}{2}h(X)$ for all Node X, then ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
2. If $h'(X) = \frac{h(X)+h^*(X)}{2}$ for all Node X, then ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
3. If $h'(X) = h(X) + h^*(X)$ for all Node X, then ○ $C = h^*(S)$ ○ $C > h^*(S)$ ● $C \geq h^*(S)$
4. If we define the set $K(X)$ for a node X as all its neighbor nodes Y satisfying $h^*(X) > h^*(Y)$, and the following always holds

$$h'(X) \leq \begin{cases} \min_{Y \in K(X)} h'(Y) - h(Y) + h(X) & \text{if } K(X) \neq \emptyset \\ h(X) & \text{if } K(X) = \emptyset \end{cases}$$

then, ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$

5. If K is the same as above, we have

$$h'(X) = \begin{cases} \min_{Y \in K(X)} h(Y) + cost(X,Y) & \text{if } K(X) \neq \emptyset \\ h(X) & \text{if } K(X) = \emptyset \end{cases}$$

where $cost(X,Y)$ is the cost of the edge connecting X and Y,
then, ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$

6. If $h'(X) = \min_{Y \in K(X)+\{X\}} h(Y)$ (K is the same as above), ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$

(ii) In which of the conditions above, h' is still **admissible** and for sure to dominate h? Check all that apply. Remember we say h_1 dominates h_2 when $h_1(X) \geq h_2(X)$ holds for all X. □ 1 ■ 2 □ 3 □ 4 □ 5 □ 6

(b) Suppose h is a **consistent** heuristic, and we conduct A* **graph search** using heuristic h' and finally find a solution.

(i) Answer exactly the same questions for each conditions in Question (a)(i).

1. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
2. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
3. ○ $C = h^*(S)$ ○ $C > h^*(S)$ ● $C \geq h^*(S)$
4. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
5. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
6. ○ $C = h^*(S)$ ○ $C > h^*(S)$ ● $C \geq h^*(S)$

(ii) In which of the conditions above, h' is still **consistent** and for sure to dominate h? Check all that apply.
□ 1 ■ 2 □ 3 □ 4 ■ 5 □ 6

Explanations:

All the $C > h^*(S)$ can be ruled out by this counter example: there exists only one path from S to G.

Now for any $C = h^*(S)$ we shall provide a proof. For any $C \geq h^*(S)$ we shall provide a counter example.

a3b3 - Counter example: SAG fully connected. cost: SG=10, SA=1, AG=7. h*: S=8, A=7, G=0. h: S=8, A=7, G=0. h': S=16, A=14, G=0.

a4 - Proof: via induction. We can have an ordering of the nodes $\{X_j\}_{j=1}^{n}$ such that $h^*(X_i) \geq h^*(X_j)$ if $i < j$. Note any $X_k \in K(X_j)$ has $k > j$.

X_n is G, and has $h'(X_n) \leq h(X_n)$.

Now for j, suppose $h'(X_k) \leq h(X_k)$ for any $k > j$ holds, we can have $h'(X_j) \leq h'(X_k) - h(X_k) + h(X_j) \leq h(X_j)$ ($K(X_j) = \emptyset$ also get the result).

b4 - Proof: from a4 we already know that h' is admissible.

Now for each edge XY, suppose $h^*(X) \geq h^*(Y)$, we always have $h'(X) \leq h'(Y) - h(Y) + h(X)$, which means $h'(X) - h'(Y) \leq h(X) - h(Y) \leq cost(X,Y)$, which means we always underestimate the cost of each edge **from the potential**

optimal path direction. Note h' is not necessarily to be consistent ($h'(Y) - h'(X)$ might be very large, e.g. you can arbitrarily modify h'(S) to be super small), but it always comes with optimality.

a5 - Proof: the empty K path: $h'(X) \leq h(X) \leq h^*(X)$. the non-empty K path: there always exists a $Y_0 \in K(X)$ such that Y_0 is on the optimal path from X to G. We know $cost(X, Y_0) = h^*(X) - h^*(Y_0)$, so we have $h'(X) \leq h(Y_0) + cost(X, Y_0) \leq h^*(Y_0) + cost(X, Y_0) = h^*(X)$.

b5 - Proof:

First we prove $h'(X) \geq h(X)$. For any edge XY, we have $h(X) - h(Y) \leq cost(X, Y)$. So we can have $h(Y) + cost(X, Y) \geq h(X)$ holds for any edge, and hence we get the dominace of h' over h. Note this holds only for consistent h.

We then have $h'(X) - h'(Y) \leq h(Y) + cost(X, Y) - h'(Y) \leq cost(X, Y)$. So we get the consistency of h'.

Extension Conclusion 1: If we change K(X) into {all neighbouring nodes of X} + {X}, h' did not change.

Extension Conclusion 2: h' dominates h, which is a better heuristic. This (looking one step ahead with h') is equivalent to looking two steps ahead in the A* search with h (while the vanilla A* search is just looking one step ahead with h).

a6 - Proof: $h'(X) \leq h(X) \leq h^*(X)$.

b6 - counter example: SAB fully connected, BG connected. cost: SA=8, AB=1, SB=10, BG=30. h*: A=31, B=30 G=0. h=h*. h': A=30, B=0, C=0.