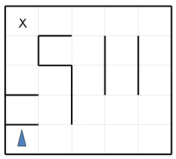
CS 188
Spring 2019

Section Handout 2 Solutions

1 Search and Heuristics

Imagine a car-like agent wishes to exit a maze like the one shown below:



The agent is directional and at all times faces some direction $d \in (N, S, E, W)$. With a single action, the agent can *either* move forward at an adjustable velocity v *or* turn. The turning actions are *left* and *right*, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero). The moving actions are *fast* and *slow*. *Fast* increments the velocity by 1 and *slow* decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its NEW adjusted velocity (see example below). A consequence of this formulation is that it is impossible for the agent to move in the same nonzero velocity for two consecutive timesteps. Any action that would result in a collision with a wall crashes the agent and is illegal. Any action that would reduce v below 0 or above a maximum speed $V_{\max}$ is also illegal. The agent's goal is to find a plan which parks it (stationary) on the exit square using as few actions (time steps) as possible.

As an example: if at timestep t the agent's current velocity is 2, by taking the *fast* action, the agent first increases the velocity to 3 and move 3 squares forward, such that at timestep $t+1$ the agent's current velocity will be 3 and will be 3 squares away from where it was at timestep t. If instead the agent takes the *slow* action, it first decreases its velocity to 1 and then moves 1 square forward, such that at timestep $t+1$ the agent's current velocity will be 1 and will be 1 squares away from where it was at timestep t. If, with an instantaneous velocity of 1 at timestep $t+1$, it takes the slow action again, the agent's current velocity will become 0, and it will not move at timestep $t+1$. Then at timestep $t+2$, it will be free to turn if it wishes. Note that the agent could not have turned at timestep $t+1$ when it had a current velocity of 1, because it has to be stationary to turn.

(a) If the grid is M by N, what is the size of the state space? Justify your answer. You should assume that all configurations are reachable from the start state.

The size of the state space is $4MN(V_{max}+1)$. The state representation is (direction facing, x, y, speed). Note that the speed can take any value in $\{0, ..., V_{max}\}$.

(b) Is the Manhattan distance from the agent's location to the exit's location admissible? Why or why not?

No, Manhattan distance is not an admissible heuristic. The agent can move at an average speed of greater than 1 (by first speeding up to V_{max} and then slowing down to 0 as it reaches the goal), and so can reach the goal in less time steps than there are squares between it and the goal. A specific example: A timestep 0, the agent's starts stationary at square 0 and the target is 9 squares away at square 9. At timestep 0, the agent takes the *fast* action and ends up at square 1 with a velocity of 1. At timestep 1, the agent takes the *fast* action and ends up at square 3 with a velocity of 2. At timestep 2, the agent takes the *fast* action and ends up at square 6 with a velocity of 3. At timestep 3, the agent takes the *slow* action and ends up at square 8 with a velocity of 2. At timestep 4, the agent takes the *slow* action and ends up at square 9 with a velocity of 1. At timestep 5, the agent takes the *slow* action and stays at square 9 with a velocity of 0. Therefore, the agent can move 9 squares by taking 6 actions.

(c) State and justify a non-trivial admissible heuristic for this problem which is not the Manhattan distance to the exit.

There are many answers to this question. Here are a few, in order of weakest to strongest:

(a) The number of turns required for the agent to face the goal.

(b) Consider a relaxation of the problem where there are no walls, the agent can turn, change speed arbitrarily, and maintain constant velocity. In this relaxed problem, the agent would move with V_{max}, and then suddenly stop at the goal, thus taking $d_{manhattan}/V_{max}$ time.

(c) We can improve the above relaxation by accounting for the acceleration and deceleration dynamics. In this case the agent will have to accelerate from 0 from the start state, maintain a constant velocity of V_max, and slow down to 0 when it is about to reach the goal. Note that this heuristic will always return a greater value than the previous one, but is still not an overestimate of the true cost to reach the goal. We can say that this heuristic *dominates* the previous one.

In particular, let us assume that $d_{manhattan}$ is greater than and equal to the distance it takes to accelerate to and decelerate from V_{max} (In the case that $d_{manhattan}$ is smaller than this distance, we can still use $d_{manhattan}/V_{max}$ as a heuristic). We can break up the $d_{manhattan}$ into three parts: d_{accel}, $d_{V_{max}}$, and d_{decel}. The agent travels a distance of d_{accel} when it accelerates from 0 to V_{max} velocity. The agent travels a distance of d_{decel} when it decelerates from V_{max} to 0 velocity. In between acceleration and deceleration, the agent travels a distance of $d_{V_{max}} = d_{manhattan} - d_{accel} - d_{decel}$. $d_{accel} = 1 + 2 + 3 + V_{max} = \frac{(V_{max})(V_{max}+1)}{2}$ and $d_{decel} = (V_{max} - 1) + (V_{max} - 2) + ... + 1 + 0 = \frac{(V_{max})(V_{max}-1)}{2}$. So $d_{V_{max}} = d_{manhattan} - \frac{(V_{max})(V_{max}+1)}{2} - \frac{(V_{max})(V_{max}-1)}{2} = d_{manhattan} - V_{max}^2$. It takes V_{max} steps to travel the initial d_{accel}, $\frac{d_{manhattan}-V_{max}^2}{V_{max}}$ steps to travel the middle $d_{V_{max}}$ and V_{max} steps to travel the last d_{decel}. Therefore, our heuristic is

$$\begin{cases} \frac{d_{manhattan}}{V_{max}}, & \text{if } d_{manhattan} \leq V_{max}^2 = d_{accel} + d_{decel} \\ \frac{d_{manhattan}}{V_{max}} + V_{max}, & \text{if } d_{manhattan} > V_{max}^2 = d_{accel} + d_{decel} \end{cases}$$

(d) If we used an inadmissible heuristic in A* graph search, would the search be complete? Would it be optimal?

If the heuristic function is bounded, then A* graph search would visit all the nodes eventually, and would find a path to the goal state if there exists one. An inadmissible heuristic does not guarantee optimality as it can make the good optimal goal look as though it is very far off, and take you to a suboptimal goal.

(e) If we used an *admissible* heuristic in A* graph search, is it guaranteed to return an optimal solution? What if the heuristic was consistent? What if we were using A* tree search instead of A* graph search?

Although admissible heuristics guarantee optimality for A* *tree* search, they do not necessarily guarantee optimality for A* *graph* search; they are only guaranteed to return an optimal solution if they are consistent as well.

(f) Give a general advantage that an inadmissible heuristic might have over an admissible one.

The time to solve an A* search problem is a function of two factors: the number of nodes expanded, and the time spent per node. An inadmissible heuristic may be faster to compute, leading to a solution that is obtained faster due to less time spent per node. It can also be a closer estimate to the actual cost function (even though at times it will overestimate!), thus expanding less nodes. We lose the guarantee of optimality by using an inadmissible heuristic. But sometimes we may be okay with finding a suboptimal solution to a search problem.

Q2. Heuristics

For parts a and b below, consider the following PacMan problem in a $N \times M$ grid, where $N \geq 2$, $M \geq 2$: there is a food pellet located at each corner, and Pacman must navigate the maze to find each one. Our goal is to find the shortest path that eats all four pellets in each corner.

(a) For each of the following heuristics, say whether or not it is admissible. If a heuristic is inadmissible, give a concrete counterexample (i.e., draw a maze configuration in which the value of the heuristic exceeds the true cost).

- h_1 is the maze distance to the nearest food pellet (if no food pellets remain, $h_1 = 0$).

  [*Admissible* or *Not-Admissible*]
  Eating pellets requires reaching them, which means moving at least this far.

- h_2 is the number of uneaten food pellets remaining.

  [*Admissible* or *Not-Admissible*]
  Eating pellets requires reaching them all, which means at least this many moves.

- $h_3 = h_1 + h_2$

  [*Admissible* or *Not-Admissible*]
  Doesn't quite work because if the first pellet is one step away this heuristic counts 2. Counterexample: In a 2x2 board with pellets in 3 corners, $h_3 = 4$ but $h^* = 3$.

- $h_4 = \max\{h_1, h_2\}$

  [*Admissible* or *Not-Admissible*]
  See textbook. Since both h_1 and h_2 are lower bounds on h^*, their maximum is also.

- $h_5 = |h_1 - h_2|$

  [*Admissible* or *Not-Admissible*]
  Since $h_1 \geq 0$ and $h_2 \geq 0$, this is never bigger than $h_4 = \max\{h_1, h_2\}$.

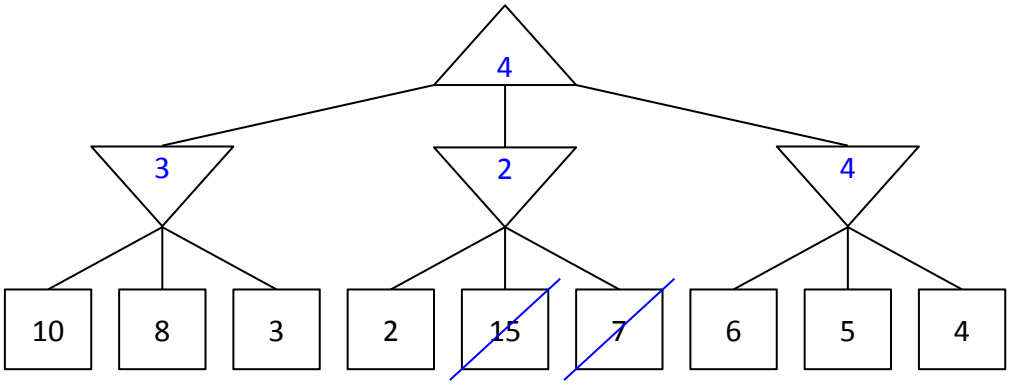
(b) Pick one heuristic from part (a) that you said was inadmissible; call it h_k. Give the smallest integer constant $\epsilon > 0$ such that $h' = h_k - \epsilon$ is an admissible heuristic. Briefly justify your answer.

$\epsilon = 1$ works. It must be at least 1, to fix the counterexample given above. 1 is also sufficient, because the total cost must be at least the cost to get to any pellet (h_1) and the cost of eating the remaining $n - 1$ pellets ($h_2 - 1$). The exception to this is when you are in the goal state, whose heuristic value would be -1 in this scheme. In this case, we can would define our heuristic as:

$$h'_3(s) = \begin{cases} h_1(s) + h_2(s) - 1, & \text{if } s \neq goal \\ h_1(s) + h_2(s), & \text{if } s = goal \end{cases}$$

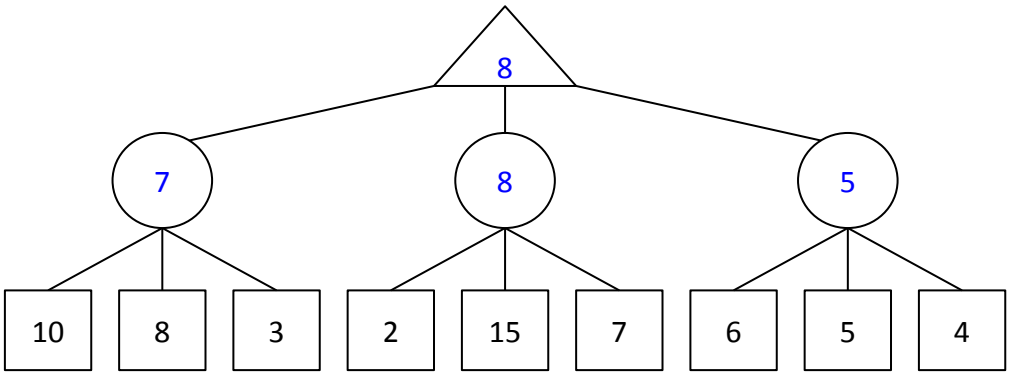
3 Games

(a) Consider the zero-sum game tree shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. Assuming both players act optimally, fill in the minimax value of each node.



(b) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. Assume the search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child.

(c) (optional) Again, consider the same zero-sum game tree, except that now, instead of a minimizing player, we have a chance node that will select one of the three values uniformly at random. Fill in the expectimax value of each node. The game tree is redrawn below for your convenience.



(d) (optional) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. No nodes can be pruned. There will always be the possibility that an as-yet-unvisited leaf of the current parent chance node will have a very high value, which increases the overall average value for that chance node. For example, when we see that leaf 4 has a value of 2, which is

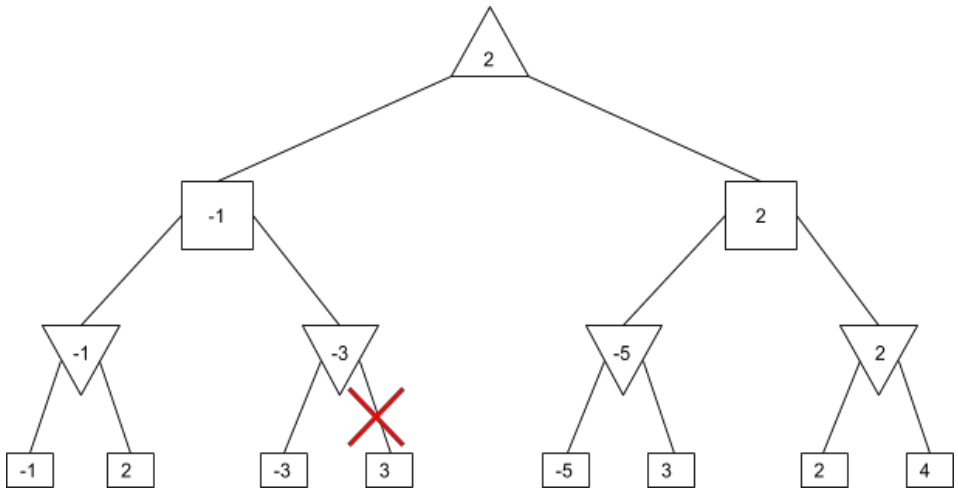
much less than the value of the left chance node, 7, at this point we cannot make any assumptions about how the value of the middle chance node will ultimately be more or less in value than the left chance node. As it turns out, the leaf 5 has a value of 15, which brings the expected value of the middle chance node to 8, which is greater than the value of the left chance node. In the case where there is an upper bound to the value of a leaf node, there is a possibility of pruning: suppose that an upper bound of $+10$ applies only to the children of the rightmost chance node. In this case, after seeing that leaf 7 has a value of 6 and leaf 8 has a value of 5, the best possible value that the rightmost chance node can take on is $\frac{6+5+10}{3} = 7$, which is less than 8, the value of the middle chance node. Therefore, it is possible to prune leaf 9 in this case.

Q4. Fair Play

Consider a game tree with three agents: a maximizer, a minimizer, and an equalizer. The maximizer chooses the highest score, the minimizer chooses the lowest score, and the equalizer chooses tries to *minimize the absolute value* (i.e. equalizer wants to make the game as close as possible, so it chooses whichever value is closest to zero).

We use an upward-facing triangle to represent a max node, a downward-facing triangle to represent a min node, and a square to represent an equalizer node.

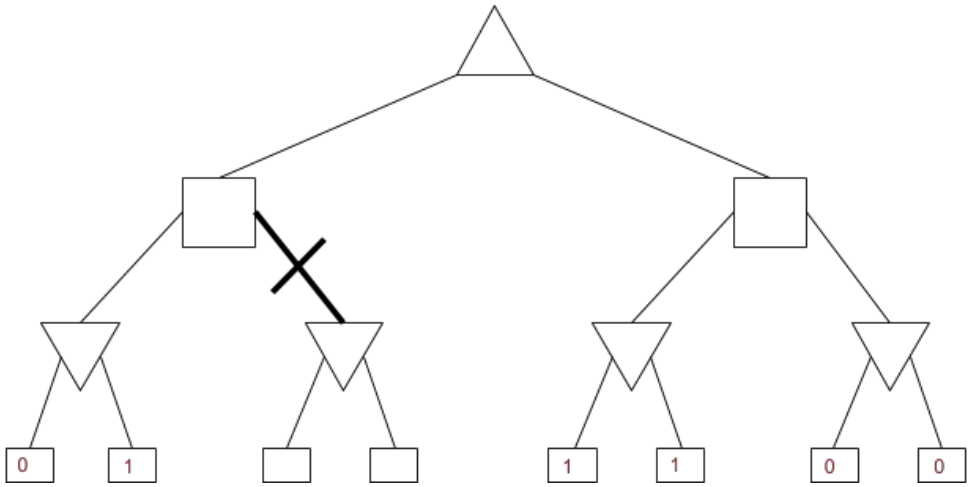
(a) Fill in all values in the game tree below:



(b) In the same game tree above, put an X on the line of all branches that can be pruned, or write "No pruning possible." Assume that branches are explored from left to right.
We can prune this node because once min sees a -3, the min node is worth -3 or lower, so equalizer prefers -1, and the fourth leaf is irrelevant. Note that we cannot prune the sixth leaf: even though the min node is -5 or worse, suppose the seventh and eighth leaves were +10; in that case, if the sixth leaf were -20, the equalizer would prefer the +10 to the -20, causing max to prefer moving right, whereas if in the same case the sixth leaf were -5, the equalizer would prefer the -5 to the +10, causing max to prefer moving left. Hence in some situations the value of the sixth leaf matters. (Note that in min/max trees, pruning can be decided *without* considering the values that subsequent leaves might have.)

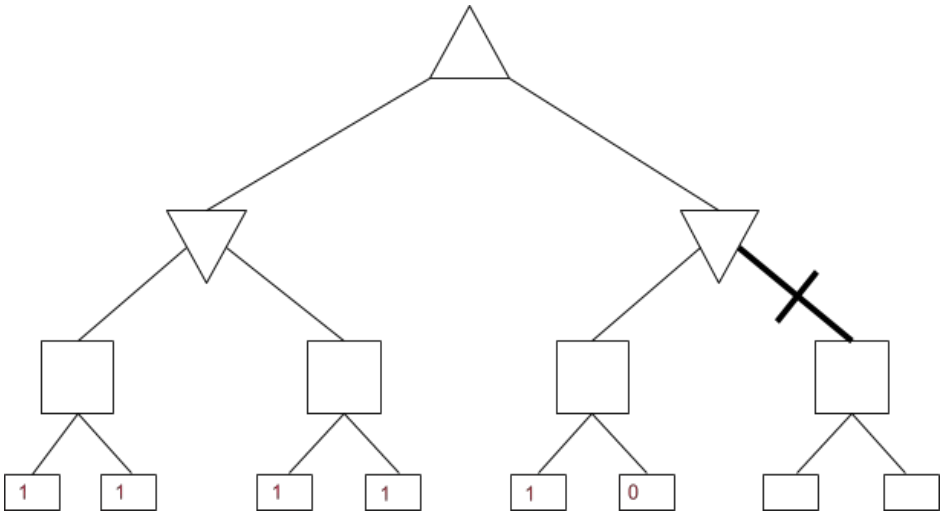
(c) For each of the following game trees, fill in values in the leaf nodes such that only the marked, bold branches can be pruned. Assume that branches are explored from left to right. If no values will allow the indicated nodes to be pruned, write "Not possible." **Be very clear**: if you write "Not possible," we will not look at the values you filled in.

(i) [Hint: what is the *best possible value* from the equalizer's viewpoint?]



0 is the optimal value for the equalizer, so the marked subtree is irrelevant. Note that you also have to put in values for the other leaves so that *no* pruning takes place!

(ii) Note that the order of the players has changed in the game tree below.



Here the pruning is essentially standard alpha-beta pruning because the top two levels of the tree are max and min.