EECS 391 Written HW 2

William Koehrsen wjk68

October 2, 2017

1. A-star search is designed to find a solution path to the goal state, while simulated annealing is a local search algorithm which is used only for finding the goal state. The primary difference between A-star and simulated annealing search is in choosing which node to expanded next. A-star search generates a search tree and expands the node on the front of the priority queue, which will be the node on the frontier with the lowest total cost regardless of the depth of the node. In contrast, simulated annealing will not consider past nodes and will only choose a node that is a successor of the current state. Moreover, the total cost function for A-star search depends on the path distance to the potential node and the heuristic function evaluated at the potential node while the total cost for simulated annealing is the difference in the evaluation function between the potential node and the current node. Simulated annealing generates all successor states of the current state and then selects one at random. If the state decreases the cost, it will be selected with probability 1. If the state increases the cost function then it will be selected with probability where ΔE is the increase in cost, and T is the “temperature”, or a measurement of how much increased cost the algorithm is willing to tolerate when selecting a move. Over time, the temperature, T, is lowered, meaning that a move that increases the cost is less likely to be accepted as the algorithm iterations increase. Simulated annealing was designed to solve the issue of getting stuck in a local optimum when performing gradient descent by allowing moves that temporarily increase the cost function to be selected with the goal of “jumping out” of a local minimum.

**Optimality:** A-star search is guaranteed to be optimal and complete if the heuristic used is admissible and consistent. That is, A-star will find the shortest solution path as long as the heuristic used in the total cost function meets the two criteria.

Simluated annealing is not guaranteed to be optimal or complete because it can still become stuck in a local optimum as in gradient descent. Selecting moves that increase the cost function reduces the likelihood of becoming stuck, but it can still happen if the local minimum is too deep and steep on both sides for the algorithm to “jump out of” by selecting a move that increases the cost.

**Memory complexity:** A-star search has high memory requirements because the goal is to find the solution path to the terminal state. All nodes on the frontier and all expanded nodes, regardless of depth, must be keep in memory. It is essentially guided breadth-first search which means that the memory complexity is exponential in regards to the depth of the solution [O(bd); where b is the branching factor and d is the solution depth]. In contrast, simulated annealing is a local search algorithm, and if the goal is to find only the solution state and not the path to the solution state, simulated annealing must only remember one state at a time. However, if you want to use simulated annealing to find a solution path, then the algorithm must remember all previously explored states resulting in an exponential memory cost.

1. Gradient ascent search moves along the direction of steepest increase, or the largest positive gradient. It therefore can reach a local maxima when the cost function is non-concave and get stuck because both sides of the local maxima have a negative gradient. (Gradient ascent is guaranteed to find the optimal solution if the cost function is concave, that is a line connecting any two points on the curve does not pass through the curve itself). To adapt simulated annealing to this gradient ascent, the algorithm could pick a direction at random, if the gradient is positive in this direction, it would select that direction with probability 1. If the gradient is negative in that direction, it would choose that direction with probability proportional to where D is the value of the gradient. More negative gradients in this case will be less likely to be selected. As the iterations progress, the temperature, T, would be reduced making it less likely that directions that have a negative gradient would be selected.

A heuristic is a method for estimating the ranking of states when complete information about the state, such as distance to the goal, is unknown. In search, the heuristic often takes the form of an estimate of the distance from the state to the goal state. The gradient could be used as a heuristic, but it would not be admissible or consistent. There is no guarantee that the direction with a larger gradient will be closer to the goal than a direction with a smaller, or even negative gradient. The gradient could be considered a heuristic, because it estimates the quality of a state, but it is not an admissible or consistent heuristic and therefore will not make the gradient ascent algorithm complete or optimal.

1. In Minimax search with A being the maximizer and B the minimizer, A can be thought of as selecting the node that results in the maximum minimum terminal state. This is illustrated in the following simple game tree:

In this case, A will choose the left branch because it has a minimum terminal value of 8 which is the maximum minimum value of the three branches (the minimum value of the middle and right branches are 4 and 5 respectively). If A selects the left branch, the worst expected outcome is 8 if B is an optimal minimizer. However, if B is non-optimal, than A can only increase or equal its expected outcome which could be 9 or 12. This demonstrates that when playing against a non-optimal minimum player, the maximizer can only do equal or better than when playing against an optimal minimum player.

The game tree above could present a situation where A can do better using a suboptimal strategy against a suboptimal minimum player. Let’s suppose that B chooses a move completely at random, that is, B is a stochastic player. Let us also suppose that A’s strategy is to choose the branch that has the highest average leaf value. In this case, from left to right the values are 9.7, 6, 12. In this case, A would choose the right branch every time, and although the strategy would not be optimal if playing against an optimal minimum opponent, it would be the best strategy in the long run against a completely random player. In this situation, as the number of games increases, the expected payoff of each branch converges to the average value of the leaf nodes.

1. Each state is represented as (A, B) where the number represents the square occupied by the player. The leftmost square is 1 and the rightmost is 5. A wins if in position 5 and B wins if in position 1. A repeated state is one in which both players are in the same position as a previous state **and** the same player has the choice of move.

A.

+1 with one box = terminal state, A wins

-1 with one box = terminal state, B wins

? with two boxes = terminal loop state with unknown value



B. The loop states have an unknown value. When a player has the choice between winning and entering into a loop, they will choose to win. Therefore, max(?, +1) = +1 and min(?, -1) = -1. Moreover, if a player has a choice between losing and entering into a loop, they will choose to enter the loop. Therefore max(?, -1) = ? and min(?, +1) = ?. The root node has a value of -1 indicating that an optimal minimum player will always win with 5 states.

C. Standard minimax is a depth-first search because the algorithm must reach a terminal state in order to assess the utility of all above nodes. Therefore, with this game tree, the algorithm is susceptible to getting caught in infinite loops. For example, if the minimax algorithm performs DFS and in a situation with multiple nodes always chooses to go on the right branch, it will get stuck in the pattern (2,5); (3,5); (3,4); (2,4); (2,5).

D. In this game, B always wins as evidenced by the value of the top node. This can be generalized to any game in this environment with n states where n is odd.