Intro to AI Homework 2

Roshan Patel 200000858 Abhay Dhiman 192009888

March 19th 2024

1 Introduction

In this assignment we study and investigate search problems in AI.

2 Problem 1

We are tasked to perform an A* Search on the provided map and heuristics to Bucharest. We are told to start at Lugoj and find a path to Bucharest. The order in which the nodes/cities are expanded is the following:

1. (Lugoj, 244, 0, 244)
2. (Mehadia, 301, 70, 241)
3. (Drobeta, 387, 145, 242)
4. (Craiova, 425, 265, 160)
5. (Timisoara, 440, 111, 329)
6. (Pitesti, 503, 403, 100)
7. (Bucharest, 504, 504, 0)

3 Problem 2

A state space with the initial state being 1 and the successors of each state(k) being 2k and 2k+1. We are tasked to get to the goal state of 11 through various search methods.

3.1 BFS

Performing BFS on this state space context we get the following order:

[1,2,3,4,5,6,7,8,9,10,11]

3.2 DLS with limit 3

Performing a Depth Limited Search with a limit of 3, we get the following order:

[1,2,4,8,9,5,10,11]

3.3 IDS

Performing an Iterative Depth Search, we get the following order:

Depth 0: [1]
Depth 1: [1,2,3]
Depth 2: [1,2,4,5,3,6,7]
Depth 3: [1,2,4,8,9,5,10,11]

3.4 Bidirectional Search

Bi-directional Search would work very well in this problem because it has one unique start and one unique goal state. Going from Start state 1 to Goal state 11 down the the tree we first get

[1,2,3]

Working from the goal state to the start state up the tree, 11 only has one predecessor which is 5, so we get

[11,5]

We continue performing BFS in both directions. Start to Goal (Down the tree) we get

[1,2,3,4,5]

We can stop here because we have found 5 in both directions of searching. The branching factor we can conclude is 2 for forward(Start to Goal) because each state has two successors it can visit. The branching factor for going backwards(Goal to Start) is 1 because each node only has one predecessor. Tracing it back we get that the shortest path to the goal state, 11, is

[1,2,5,11]

Problem 3 (5 points)

Which of the following statements are correct and which ones are wrong?

(a) Breadth-first search is a special case of uniform-cost search **False**

(b) Depth-first search is a special case of best-first tree search **False**

(c) Uniform-cost search is a special case of A* search **False**

(d) Depth-first graph search is guaranteed to return an optimal solution **False**

(e) Breadth-first graph search is guaranteed to return an optimal solution **False**

(f) Uniform-cost graph search is guaranteed to return an optimal solution **True**

(g) A* graph search is guaranteed to return an optimal solution if the heuristic is consistent **True**

(h) A* graph search is guaranteed to expand no more nodes than depth-first graph search if the heuristic is consistent **True**

(i) A* graph search is guaranteed to expand no more nodes than uniform-cost graph search if the heuristic is consistent **True**

Problem 4 (2 points)

Iterative deepening is sometimes used as an alternative to breadth-first search. Give one advantage of iterative deepening over BFS, and give one disadvantage of iterative deepening as compared with BFS. Be concise and specific.

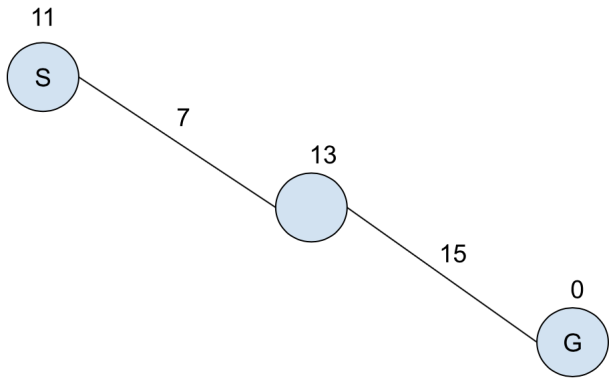
One advantage of iterative deepening as opposed to BFS is that, since it traverses nodes in the manner that DFS does, the space complexity is better than BFS.
One disadvantage of iterative deepening as opposed to BFS is that, since iterative deepening has to start from the root again every time the depth-limit is incremented, its time complexity is worse than BFS.

Problem 5 (10 points)

Prove that if a heuristic is consistent, it must be admissible. Construct an example of an admissible heuristic that is not consistent. (Hint: you can draw a small graph of 3 nodes and write arbitrary cost and heuristic values so that the heuristic is admissible but not consistent).

The reason why a consistent heuristic must be admissible is because the true meaning of the values spit out by any heuristic function lies in their comparison to one another. And so, if the values are always decreasing as you get closer to the goal state, they must be admissible. Even if the numbers overestimate a distance by some 3rd-party measure, they don't do so relative to one another so it doesn't matter, meaning they essentially don't.

Below is a mini graph of 3 nodes, showing a heuristic that is admissible but not consistent.



Problem 6 (3 points)

In a Constraint Satisfaction Problem (CSP) search, explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining.

The reason why, in Constraint Satisfaction Problem (CSP) search it is a good heuristic to choose the variable that is most constrained but the value that is least constraining is because it helps prune large sets of possibilities early on, reducing the computational resources required to solve the problem. A variable that is most constrained immediately removes a large chunk of possibilities that we may have wasted time parsing, and the value that is least constraining allows us to reach a solution more quickly and easily with the constraints for other variables being easier to satisfy due to looser initial constraints.

4 Problem 7

In this problem we investigate a game tree presented in 8.1. The MAX player goes first and the min player second.

4.1 Minmax Algorithm

We perform the minmax algorithm to find the best move for MAX to make. As shown in Figure 1, the best move for MAX is 4.

4.2 Left to Right Alpha Beta Pruning

Figure 2 shows the Minmax game after its been alpha beta pruned from left-to-right.

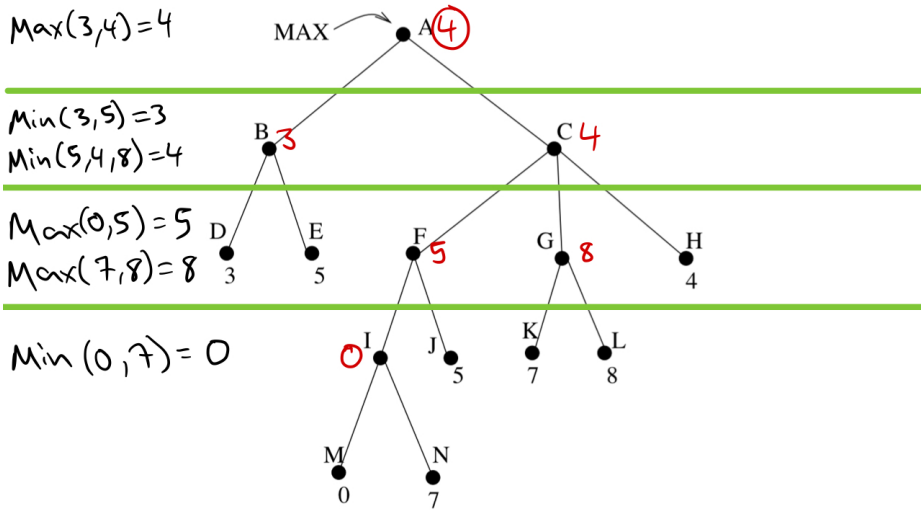


Figure 1: Minmax on game tree

4.3 Right to Left Alpha Beta Pruning

Figure 3 shows the Minmax game after its been alpha beta pruned from right-to-left. As shown in Figures 2 and 3, pruning reduces the amount of nodes visited thus making the algorithm more efficient. However, in this scenario we can see that pruning from right-to-left removed more nodes than left-to-right making it more efficient than the latter. In this particular case proved to be more beneficial because at the third layer with D, E, F, G, H, the max is found. In particular at F, its value should end with being 5. So if we are to go from right-to-left and get to 5 first, then the pruning removes the other branch connected to F. This saves the algorithm some comparison steps thus making it more efficient in this particular case. In general it depends on the structure of the game tree to determine if left-to-right or right-to-left is better.

Problem 8 (10 points)

Which of the following are admissible, given admissible heuristics h_1, h_2? Which of the following are consistent, given consistent heuristics h_1, h_2? Justify your answer.

(a) $h(n) = \min\{h_1(n), h_2(n)\}$

Given admissible heuristics h_1 and h_2, $h(n)$ is admissible. This is because one of h_1 or h_2 is chosen, and they are both guaranteed to not overestimate the distance from the current node to the goal.

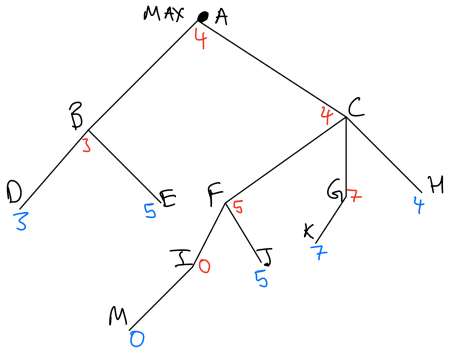


Figure 2: Minmax Game Tree after Alpha Beta Pruning from Left to Right

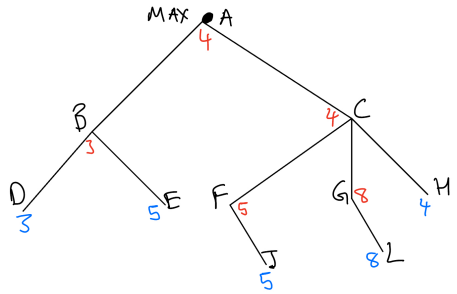


Figure 3: Minmax Game Tree after Alpha Beta Pruning from Right to Left

Given consistent heuristics h_1 and h_2, $h(n)$ is not consistent, since it is possible that at one node h_1 is chosen but at the next node h_2 is chosen. In this case, it is possible that the first node's heuristic is lower than the second node's heuristic, making $h(n)$ inconsistent.

(b) $h(n) = w \cdot h_1(n) + (1 - w) \cdot h_2(n)$, where $0 \leq w \leq 1$

Given admissible heuristics h_1 and h_2, $h(n)$ is admissible. This is because, assuming w remains constant for the entirety of the search, $h(n)$ is artfully combining the information received from both heuristic functions. With a constant weight on each, their sum should also not overestimate the heuristic at a given node. Regardless of the weight, $h(n)$ will never exceed h_1 or h_2.

Given consistent heuristics h_1 and h_2, $h(n)$ is consistent. This is because, assuming w remains constant for the entirety of the search, $h(n)$ is simply combining the information received from both heuristic functions. With a constant weight on each, their sum should also reduce as we get closer to the goal, since the parts that make up the sum, h_1 and h_2, will reduce.

(c) $h(n) = \max\{h_1(n), h_2(n)\}$

Given admissible heuristics h_1 and h_2, $h(n)$ is admissible. This is because one of h_1 or h_2 is chosen, and they are both guaranteed to not overestimate the distance from the current node to the goal.

Given consistent heuristics h_1 and h_2, $h(n)$ is not consistent, since it is possible that at one node h_1 is chosen but at the next node h_2 is chosen. In this case, it is possible that the first node's heuristic is lower than the second node's heuristic, making $h(n)$ inconsistent.

For A*, I would prefer to use heuristic b. This is because it is admissible and consistent both, but more importantly it allows me to factor into my heuristic calculation information from both heuristic function 1 (h_1) and heuristic function 2 (h_2). The more reliable information I can incorporate, the better and more efficient my A* will be.

5 Problem 9

5.1 Hill-climbing is better than simulated annealing

Hill-climbing is a simple algorithm where program will keep "ascending" if the next step is higher than the current. Once the next step is "flat" or the same height or is lower than the current step than the hill-climbing algorithm will stop. Therefore, cases where hill-climbing would be better suited for finding global maxima than simulated annealing is when the "landscape" or the function has only one maxima and that is the global maxima. In this case, randomization is not necessary as there is only one global maxima therefore starting anywhere and applying hill-climbing will take you to that global maxima. This also assumes there are no plateaus.

5.2 Randomization is only necessary

Choosing random states as the next step is an integral part of simulated annealing. However there are some cases where simply choosing random states is enough to find the solution rather than also including hill climbing. These are the cases where the function/landscape is smooth and continuous.

5.3 Simulated annealing is optimal

Covering the cases where only hill-climbing is necessary or only randomization is necessary, we find that simulated annealing is the most optimal for complex functions. This means functions with multiple local optima and that are still continuous but not necessarily smooth. These cases are the most frequent as data in most cases is relatively complex. Simulated annealing will not necessarily find the global optima, but it will escape local optima to find a better solution.

5.4 Improving simulated annealing

To improve the simulated annealing search especially in the cases where the annealing schedule is slow, we can set a threshold. We can set a threshold for the difference between the current state and the proposed next state. This ensures that a random point is not traversed or explored in the case that it doesn't provide a better solution.

5.5 Improving with more memory

Having more memory to hold essentially 1M states we can improve the search. To do so we can simply save the states that are visited in these 1M states. This will force the algorithm to not visit states that are already visited and visit new states. This would help prevent the algorithm from falling into a cycle of visiting the same states thus wasting a step in the schedule. This would allow the search to be more efficient.

5.6 Gradient ascent with simulated annealing

Considering that gradient ascent is essentially performing a search to find the most optimal set of weights and biases, simulated annealing could work very well in trying to find the a good optima. When a state in the gradient space is chosen and is stuck at a local maxima after hill climbing, a new state could be chosen through randomization or by adding some random noise to essentially nudge the state past the maxima. This would help the algorithm escape a local maxima and look for a better set of weights and biases.