CS440-Assignment\_2

Xinyu Meng, netid: xm73, RUID: 193002567

Qichao Lin, netid: ql180, RUID: 196005340

Problem 1 (*10 points):* Trace the operation of A∗ search (use the tree version, i.e. without using a closed list) applied to the problem of getting to Bucharest from Lugoj using the straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the f, g, and h score for each node. You don’t need to draw the graph, just right down a sequence of (city, f (city), g(city), h(city)) in the order in which the nodes are expanded.

Begin the search from Lugoj with the initial state:

(Lugoj, f(244), g(0), h(244))/

After expand (Lugoj, f(244), g(0), h(244)):

We calculate child node of Lugoj and get

(Timisoara, f(440), g(111), h(329))/

(Mehadia, f(311), g(70), h(241))/

Then choose (Mehadia, f(311), g(70), h(241)) to expand as it has the lowest f-value among all childs

Calculate the f,g,h value for each of the child node of Mehadia:

(Lugoj, f(384), g(140), h(244))/

(Drobeta, f(387), g(145), h(242))/

Choose (Lugoj, f(384), g(140), h(244)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Lugoj:

(Timisoara, f(580), g(251), h(329))

(Mehadia, f(451), g(210), h(241))/

Choose the previous (Drobeta, f(387), g(145), h(242)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Drobeta:

(Mehadia, f(461), g(220), h(241))/

(Craiova, f(425), g(265), h(160))/

Then choose (Craiova, f(425), g(265), h(160)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Craiova:

(Drobeta, f(627), g(385), h(242))

(Rimnicu Vilcea, f(604), g(411), h(193))

(Pitesti, f(503), g(403), h(100))/

Choose (Timisoara, f(440), g(111), h(329)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Timisora:

(Arad, f(595), g(229), h(366))

(Lugoj, f(466), g(222), h(244))/

Choose (Mehadia, f(451), g(210), h(241)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Mehadia:

(Lugoj, f(524), g(280), h(244))

(Drobeta, f(527), g(285), h(242))

Choose (Mehadia, f(461), g(220), h(241)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Mehadia:

(Lugoj, f(534), g(290), h(244))

(Drobeta, f(537), g(295), h(242))

Choose (Lugoj, f(466), g(222), h(244)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Lugoj:

(Timisora, f(662), g(333), h(329))

(Mehadia, f(533), g(292), h(241))

Choose (Pitesti, f(503), g(403), h(100)) to expand as it has the lowest f-value

Calculate the f,g,h value for each of the child node of Pitesti:

(Riminicu Vilcia, f(693), g(500), h(193))

(Craiova, f(701), g(541), h(160))

(Bucharest, f(504), g(504), h(0))@

Expand (Bucharest, f(504), g(504), h(0)), since Bucharest is the goal, the algorithm finished.

Problem 2 (10 points): Consider a state space where the start state is number 1 and each state k has two successors: numbers 2k and 2k + 1.

(a) Suppose the goal state is 11. List the order in which states will be visited for breadthfirst search, depth-limited search with limit 3, and iterative deepening search.

For BFS:

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

Using Depth-Limited Search with a limit of 3, the order in which states will be visited is:

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

Using Iterative Deepening Search:

Level 0: 1

Level 1: 1, 2, 3

Level 2: 1, 2, 4, 5, 3, 6, 7

Level 3: 1, 2, 4, 8, 9, 5, 10, 11

(b) How well would bidirectional search work on this problem? List the order in which states will be visited. What is the branching factor in each direction of the bidirectional search?

bidirectional search works well on this problem.

It will go both forward from 1 to 11 and backward from 11 to 1

Forward from 1 to 11:

| Iteration | Node of Forward from 1 to 11 | Forward fringe | Node of Backward from 11 to 1 | Backward fringe |
| --- | --- | --- | --- | --- |
| 1 | 1 | {2,3} | 11 | {5} |
| 2 | 2 | {3,4,5} | 5 | {2, 10} |

Then, in the second iteration, forward and backward will meet and finish the search.

The branching factor for forward search is 2, for backward search is 1.

Problem3 (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. Correct

(b) Depth-first search is a special case of best-first tree search. Correct

(c) Uniform-cost search is a special case of A∗search. Correct

(d) Depth-first graph search is guaranteed to return an optimal solution. Wrong

(e) Breadth-first graph search is guaranteed to return an optimal solution. Wrong

(f) Uniform-cost graph search is guaranteed to return an optimal solution. Correct

(g) A∗graph search is guaranteed to return an optimal solution if the heuristic is consistent. Correct

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

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

Problem4

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.

Iterative deepening depth-first search has space complexity of O(bm) and O(m) with backtracking while BFS has space complexity of O(b^d), where m is the depth of the deepest node and d is the depth of the shallowest node, which is a essential advantage over BFS. This is because iterative deepening only stores the current path from the root node to the current node, whereas BFS stores all nodes at a given depth level in memory.

A disadvantage of iterative deepening is that the time complexity O(b^d) can be slower in most of the situations. Iterative deepening may explore the same nodes multiple times at different depth limits, whereas BFS only expands each node once. Iterative deepening may take longer to find the optimal solution than BFS, especially if the branching factor of the search space is high. This is because iterative deepening may have to search more nodes than BFS to find the optimal solution, and each iteration of the algorithm takes longer than the previous one due to the increasing depth limit.

Problem 5: 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)

(4)-->1-->(1)-->3-->(0)

We take Sn, Sn-1, Sn–2…..S0 is the path we get in A\* from start Sn to the goal S0

We assume the heuristic of all S are consistent, which means for all S, h(Sn) <= c(Sn,Sn-1) + h(Sn-1) and h(S0) = 0, where c means cost, c(Sn,Sn-1) is the cost of Sn to Sn-1, and h(S1) <= c(S1, S0)

So for the state right before the goal state, S1, h(S1) <= c(S1, S0) + h(S0) = c(S1, S0)

Since c(S1, S0) = h\*(S1) where h\*(S1) denotes the real cost from S1 to the goal state, then, h(S1)<=h\*(S1), which shows that the heuristic of S1 is admissible.

Then, for the previous state of S1, h(S2) is consistent so that h(S2) <= c(S2,S1) + h(S1) = c(S2,S1) + c(S1,S0) + h(S0) = c(S2,S1) + c(S1,S0) + 0

As c(S2,S1) + c(S1,S0) = h\*(S2), we can see that S2 is also admissible

Based on what we’ve shown above, we can see that for each of the state Sn before the goal S0, h(Sn) <= h\*(Sn) so that each Sn is admissible.

Therefore, if a heuristic is consistent, it must be admissible.

Problem 6

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.

It is a good heuristic to choose the variable that is most constrained variable because it select the variable that has the fewest number of possible values remaining in its domain As it selects a variable that can quickly fail, which results in pruning the corresponding branch of the search tree, avoiding pointless searches. Also, selecting the variable with the fewest possible values first can reduce the search space more quickly, as it reduces the number of possibilities to consider for the remaining variables. Using the least constraining value for the chosen variable means choosing the value that rules out the fewest choices for the neighboring variables in the constraint graph. This approach leaves more options open for the remaining variables, potentially allowing for a faster and more efficient search and the least constraining value is the most likely to succeed since we want to find the first solution quickly. We will not fail before we try all the values. By using both the heuristics, we can find a balance between the space complexity and time complexity of the constraint graph.

Problem 7:

Consider the following game tree, where the first move is made by the MAX player and the second move is made by the MIN player.

1. What is the best move for the MAX player using the minimax procedure?
2. Perform a left-to-right(left branch first, them right branch) alpha-beta pruning on the tree. That is, draw only the parts of the tree that are visited and don’t draw branches that are cut off(no need to show the alpha or beta values).
3. Do the same thing as in the previous question, but with a right-to-left ordering of the actions. Discuss why different pruning occurs.

a. The best move will be first choose C(4), then the Min player will go H(4).

b.



c.



Problem 8:

Which of the following are admissible, given admissible heuristics h1, h2? Which of the following are consistent, given consistent heuristics h1, h2? Justify your answer.

a) h(n) = min{(n), (n)}

h(n) is admissible.

Since h(n) = min{(n), (n)}

We can get h(n) <= h1(n) or h(n) <= h2(n)

Because h1 and h2 are all admissible,

h1(n) <= h\*(n) and h2(n) <= h\*(n)

Therefore, h(n) <= h\*(n), which shows that h(n) is admissible.

h(n) is consistent.

In case 1, if h1(n) <= h2(n) and h1(n’) >= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h1(n) <= c(n, n’) + h1(n’) based on h1(n) is consistent,

and h(n’) = h2(n’) <= c(n’, n’’) + h2(n’’) where n’’ is the next node of n’,

In n’, since we assume h1(n’) <= h2(n’), we can also get h(n) = h1(n) <= c(n, n’) + h2(n’), which means h is consistent in this case.

In case 2, if h1(n) <= h2(n) and h1(n’) <= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h1(n) <= c(n, n’) + h1(n’) based on h1(n) is consistent,

and h(n’) = h1(n’) <= c(n’, n’’) + h1(n’’) where n’’ is the next node of n’,

Therefore, h is consistent in this case.

In case 3, if h1(n) >= h2(n) and h1(n’) <= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h2(n) <= c(n, n’) + h2(n’) based on h2(n) is consistent,

and h(n’) = h1(n’) <= c(n’, n’’) + h1(n’’) where n’’ is the next node of n’,

In n’, since we assume h2(n’) <= h1(n’), we can also get h(n) = h2(n) <= c(n, n’) + h1(n’), which means h is consistent in this case.

In case 4, if h1(n) >= h2(n) and h1(n’) >= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h2(n) <= c(n, n’) + h2(n’) based on h2(n) is consistent,

and h(n’) = h2(n’) <= c(n’, n’’) + h2(n’’) where n’’ is the next node of n’,

Therefore, h is consistent in this case.

Therefore, h(n) <= c(n,n’) + h(n’), h(n) is also consistent

b) h(n)=wh1(n)+(1−w)h2(n), where 0≤w≤1

h(n) is admissible.

Since h(n) = wh1(n) + (1-w)h2(n) <= max(h1(n),h2(n))

h1(n) <= h\*(n), h2(n) <= h\*(n)

Even if we take the max value for both h1, h2 so that h1(n)=h\*(n) , h2(n) = h\*(n)

h(n) = wh1(n) + (1-w)h2(n) would be at most h\*(n)\*1, which is still admissible.

Since any one of h1,h2 is admissible, h(n) is also admissible under this circumstance

h(n) is consistent if h1,h2 are all consistent

h(n)=wh1(n)+(1−w)h2(n), since w varies from 0 to 1,

Since h1,h2 are all consistent, we can put h1(n) <= c(n,n’)+h1(n’) and h2(n) <= c(n,n’)+h2(n’) into this equation.

which means h(n) <= c(n,n’)+wh1(n’) + (1-w)h2(n’)

And h(n’) = wh1(n’) + (1-w)h2(n’)

Then h(n)<= c(n,n’) + h(n’)

Therefore, h(n) is also consistent.

Since h(n)=wh1(n)+(1−w)h2(n) satisfies the triangle inequality, we can see that h(n) is also consistent.

c) h(n) = max{h1(n), h2(n)}

The heuristic h(n) = max{h1(n), h2(n)} is admissible if h1 and h2 are admissible.

To show that h(n) is admissible, we need to prove that it never overestimates the actual cost of reaching the goal state.

h(n) = h1(n) if h2(n) <= h1(n) or h(n) = h2(n) if h1(n) <= h2(n)

But in both cases, h1 and h2 are admissible, which means h1(n) <= h\*(n), and h2(n) <= h\*(n), where h\*(n) represents the real cost of n.

Since h1(n) and h2(n) are both admissible, we know that it never overestimates the actual cost of reaching the goal state. Therefore, h(n) is also admissible.

h(n) = max{h1(n), h2(n)} is also consistent when h1(n), h2(n) are consistent

In case 1, if h1(n) >= h2(n) and h1(n’) <= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h1(n) <= c(n, n’) + h1(n’) based on h1(n) is consistent,

and h(n’) = h2(n’) <= c(n’, n’’) + h2(n’’) where n’’ is the next node of n’,

In n’, since we assume h1(n’) <= h2(n’), we can also get h(n) = h1(n) <= c(n, n’) + h2(n’), which means h is consistent in this case.

In case 2, if h1(n) >= h2(n) and h1(n’) >= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h1(n) <= c(n, n’) + h1(n’) based on h1(n) is consistent,

and h(n’) = h1(n’) <= c(n’, n’’) + h1(n’’) where n’’ is the next node of n’,

Therefore, h is consistent in this case.

In case 3, if h1(n) <= h2(n) and h1(n’) >= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h2(n) <= c(n, n’) + h2(n’) based on h2(n) is consistent,

and h(n’) = h1(n’) <= c(n’, n’’) + h1(n’’) where n’’ is the next node of n’,

In n’, since we assume h2(n’) <= h1(n’), we can also get h(n) = h2(n) <= c(n, n’) + h1(n’), which means h is consistent in this case.

In case 4, if h1(n) <= h2(n) and h1(n’) <= h2(n’) where n’ is the next node of n

Then in this case, h(n) = h2(n) <= c(n, n’) + h2(n’) based on h2(n) is consistent,

and h(n’) = h2(n’) <= c(n’, n’’) + h2(n’’) where n’’ is the next node of n’,

Therefore, h is consistent in this case.

Therefore, h(n) <= c(n,n’) + h(n’), h(n) is also consistent

Among the three new heuristics, I would prefer to use the heuristic a, which is h(n) = min{h1(n), h2(n)} for A\* because it is always admissible and guarantees an optimal solution.

If the A\* implementation is modifiable, h(n)=wh1(n)+(1−w)h2(n) will be the better heuristic because it provides a weight between two heuristics, in each iteration of A\* , we can modify the weight based on the information we got from the expanded nodes. In this way, A\* will be better follow the optimal track.

Problem 9 (10 points):

Simulated annealing is an extension of hill climbing, which uses randomness to avoid getting stuck in local maxima and plateaux.

a) For what types of problems will hill climbing work better than simulated annealing? In other words, when is the random part of simulated annealing not necessary?

b) For what types of problems will randomly guessing the state work just as well as simulated annealing? In other words, when is the hill-climbing part of simulated annealing not necessary?  
c) Reasoning from your answers to parts (a) and (b) above, for what types of problems is simulated annealing a useful technique? In other terms, what assumptions about the shape of the value function are implicit in the design of simulated annealing?  
d) As defined in your textbook, simulated annealing returns the current state when the end of the annealing schedule is reached and if the annealing schedule is slow enough. Given that we know the value (measure of goodness) of each state we visit, is there anything smarter we could do?  
(e) Simulated annealing requires a very small amount of memory, just enough to store two states: the current state and the proposed next state. Suppose we had enough memory to hold two million states. Propose a modification to simulated annealing that makes productive use of the additional memory. In particular, suggest something that will likely perform better than just running simulated annealing a million times consecutively with random restarts. [Note: There are multiple correct answers here.]  
(f) Gradient ascent search is prone to local optima just like hill climbing. Describe how you might adapt randomness in simulated annealing to gradient ascent search avoid trap of local maximum.

a) When the problem solution doesn’t change direction, which means the agent only needs to move in a single direction to find the global maximum, the random part of simulated annealing would not be necessary. Hill climbing works better when the problem has a smooth, increasing, and continuous landscape with a single global optimum value, as it will continue to search for the global maximum without getting stuck in the shoulder plateaus.

b) Randomly guessing the state will work just as well as simulated annealing for problems if the problem is more random and contains multiple local optimums that spreads evenly. When the problem contains multiple up-and-down hills and more local optimums, the hill-climbing part of simulated annealing will not be necessary since the randomness of the randomly selected successor part of the simulated annealing will be enough to find the solution. Also, since the probability proportional to the function value for the agent to choose a different direction would have no significant difference compared to the complete random choice for the direction of movement.

c) Simulated annealing is useful for problems if the value function is complex and has multiple local optima. In other words, it is useful in finding global optima in the presence of large numbers of local optima. The algorithm assumes the value function is continuous but not necessarily convex. The random part of the algorithm allows for the exploration of the solution space, which can avoid getting stuck on the local optimum, while the hill-climbing part ensures it will go to the global optimum.

d) Since we know the value of each state we visit, we can compare all the state we visited and find the best state seen so far during the annealing and return it as the final result. It guarantees that the final solution is the best solution found so far during the annealing process, which is better than just returning the final state at the end of the annealing process.

e) Use the additional memory to store a set of solutions instead of just two states. In multiple elements in the annealing schedule, record all the continually increasing or decreasing states bounded by 2 million, then when choosing the direction the agent will move, use all the previous function values stored in the memory to make the decision. In this way, we can get a more precise prediction of which way will be more likely to get the global optimum, which will lead to a better time complexity.

f)

Each time using the gradient ascent search to get the local optima, it can randomly choose another node and run the gradient ascent search again. It can use the temperature function used in the simulated annealing to manage the randomness. With the algorithm processes and the temperature decreases, the probability of moving not following the gradient decreases. In the end, it will avoid the trap of local optima.