Assignment 2

Veton Abazovic — Michael Belmont

March 13, 2019

1. Trace the operation of A* search (the tree version) 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.

City	f(city)	g(city)	h(city)
Lugoj	244	0	244
Mehadia	301	70	241
Drobeta	387	145	242
Craiova	425	265	160
Timisoara	440	111	329
Pitesi	503	402	100
Buchnarest	504	504	0

2. 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.

Breadthfirst search: $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9 \to 10 \to 11$
Depth-limited search: $1 \to 2 \to 4 \to 8 \to 9 \to 5 \to 10 \to 11$
Iterative deepening search: $1; 1 \to 2 \to 3; 1 \to 2 \to 4 \to 5 \to 3 \to 6 \to 7; 1 \to 2 \to 4 \to 8 \to 9 \to 5 \to 10 \to 11$

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?

A bidirectional search would work great with this problem because the predecessor of each state n is $\frac{n}{2}$ which cam be easily computed.
We would first visit node 1 with a forward fringe = 2,3, then visit node 11 with a backward fringe = 5, then visit node 2 with a forward fringe = 3,4,5 then visit node 5 with a backward fringe = 2. Thus, we are left with 2 branching factors in forward direction and 1 in backwards direction.

3. Which of the following statements are correct and which ones are wrong? Provide a short explanation for each answer.

a) **True** when all step costs are equal.

b) **True**. Depth-first search is special case of best-first search with f(n) = -depth(n)

c) **True**. Uniform-cost search is special case of A*search with h(n) = 0. A* with h(n) = 0 (thus f(n) = g(n)) is uniform-cost; Contrariwise,A* with g(n) = 0 (thus f(n) =h(n)) is greedy.

4. 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).

Answer: The definition of consistent heuristic is said to be that for every node n and every successor n' of n generated by any action a, the estimated cost to the goal from n cannot be greater than the total cost of getting to n' + the cost of n' to the goal. The equation is:

$$h(n) \leq c(n, a, n') + h(n')$$

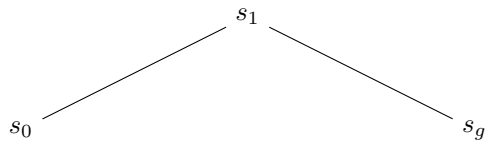
The definition of a heuristic is said to be admissible if it never overestimates the cost of reaching the goal. To prove that a consistent heuristic must also be admissible we must first provide a base case and introduce a new variable k(n) that will represent the lowest cost from n to the goal state.

Base Case: If there is only one node n and that node n is also the goal state then h(n) = 0 and that $0 \leq k(n)$.

Induction Case: We assume that n is only j nodes away from the goal that there are some a that will get us the optimal path from n to the goal state. This meaning that n' is j -1 nodes away from the goal. Then we get $h(n') \leq k(n')$ and since n' is the optimal path from n to the goal state we are left with:

$$h(n) \leq c(n, a, n') + h(n') = k(n)$$

Thus $h(n) \leq k(n)$ and thus consistent heuristics are always admissible.



In the general case we consider nodes in a path s_0, s_1, s_2, ... , s_g where s_0 is our starting state and s_g is our goal state. There is only one action to take from s_i to a successor s_{i+1} to bring us closer to the goal. We assume that the cost to go to each node is 1 and that the cheapest path from any s_i is equal to j-i, where j is the total number of nodes that need to be traveled by from the the current starting point to the goal state. Now we have to define the heuristic as the following:

$$h(s_i) = j - 2[\tfrac{i}{2}]$$

This heuristic is admissible but whenever i is odd then the heuristic becomes non consistent as h(s_i) = h(s_{i+1}) ¿ 1 + h(s_{i+1}).

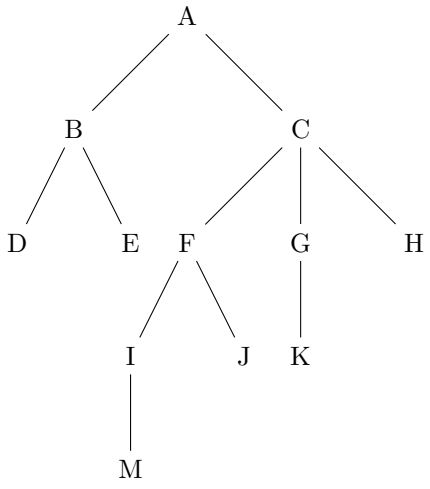
5. In a Constraint Satisfaction Problem (CSP) search, explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining.

Answer: It is a good heuristic to choose the variables that are the most constrained but the value that is least constrained because when a variable that is most constrained has a possibility that it will fail and it is efficiency to fail in the beginning to force backtracking. A value that is least constrained has a possibility that the maximum number of possibilities in the subtrees are able to avoid conflict resulting in smallest chance of backtracking.

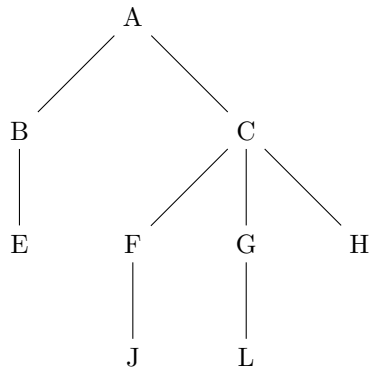
6. Consider the following game tree, where the first move is made by the MAX player and the second move is made by the MIN player

a) The best initial move for MAX player is to node C which has a value of 4.

b) A left-to-right (left branch first, then right branch) alpha-beta pruning on the tree:



c) A right-to-left (right branch first, then left branch) alpha-beta pruning on the tree:



When doing an alpha-beta pruning from right to left traversal is different from doing an alpha-beta pruning from left to right as it can be seen in the above 2 figures. The reason for this is because when we encounter a node that has an $alpha \geq beta$ then the branch next to it will not be traverse as that path will be prune. Thus when doing an alpha-beta pruning from the right or left path the other path will be cut as it is not visited by the algorithm.

7. Which of the following are admissible, given admissible heuristics h1, h2? Which of the following are consistent, given consistent heuristics h1, h2?

a) h(n) = min{h1(n), h2(n)}: This is admissible because h2 ¡ C and h1 ¡ C. This means that the min(h1; h2) is less than C. This means that it is admissible. The fact that this function picks between two consistent heuristics makes the function also consistent.

b) h(n) = wh1(n) + (1-w)h2(n), where $0 \leq w \leq 1$: h1 ¡ C and h2 ¡ C. Therefore, wh1 ¡ h1 ¡ C and wh2 ¡ h2 ¡ C. Both of these values are admissible, which means the function is also admissible. h1 ¡ C and h2 ¡ C + h0. Thus, wh1 ¡ h1 ¡ C + h0 and wh2 ¡ h2 ¡ C +h0. Meaning that these values are consistent, thus resulting to the function being consistent.

c) h(n) = max{h1(n), h2(n)}: The reason this is admissible is because h1 ¡ C and h2 ¡ C. This means that max(h1; h2) is also less than C because its picking the max of both h1 and h2 which is both less than C, and as a result admissible. Because this function picks between two consistent heuristics, this makes the function also consistent.

Question 8

A)

Simulated annealing is useful for not getting stuck within global maximas. If a problem has no global maxima within the cost function, then simulated annealing is not as useful as hill climbing would be.

B)

If the model has absolutely no structure, then hill climbing would not be effective as the model could not provide the geometry for it to proceed properly. Additionally, if there are plateaus, then simulated annealing would be useful. In this case, simulated annealing would be better.

C)

For simulated annealing, you ideally would want good geometric structure, a somewhat consistent cost function, and local maxima within the model.

D)

By storing the maximum as you go through the simulated annealing, we can keep track of the highest value encountered. Keeping the maximum value as well as the state at the time allows us to run the simulated annealing and simultaneously find the global maximum. When it finishes, if we end in a lower value than the recorded maximum, we do not lose that information.

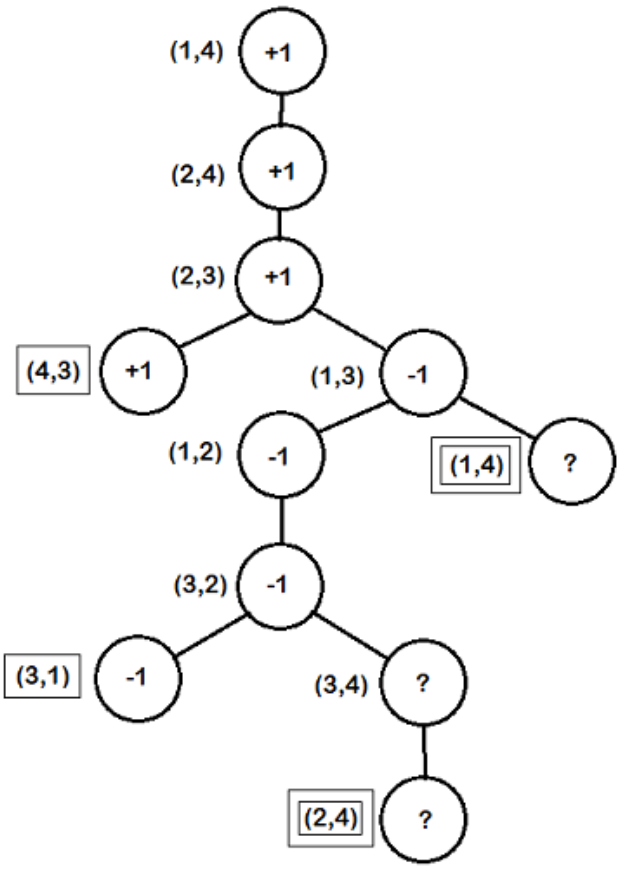
E)

If the ability to store many states is added, we think the best way to optimize the simulated annealing algorithm is to store optimal solutions after every run, and then brute forcing using the stored successful runs. This will allow increased performance with every run, increasing the probability that an optimal run is a similar or the same set of moves as the current run. These runs can also be compared to each other to determine if one is more optimal than another in a similar situation.

Question 9

1-4.

Look at the figure below.



5.

To handle the ? values, the agent is assumed to always choose to win when presented with a choice between winning and entering a ? state. This is equivalent to: min(-1, ?) = -1 and max(+1, ?) = +1. In the case that all possible choices are ?, then the backed-up value is ?.