CS440 (Intro to AI) - HW2

Janine Yanes (209008173), Aadi Gopi (207003115), Jason Cai (208004639)

March 14, 2025

1 Problem 1

(Lugoj, 244, 0, 244) → (Mehadia, 311, 70, 241) → (Drobeta, 387, 145, 242) → (Craiova, 425, 265, 160) → (Timisoara, 440, 111, 329) → (Pitesti, 503, 403, 100) → (Bucharest, 504, 504, 0)

2 Problem 2

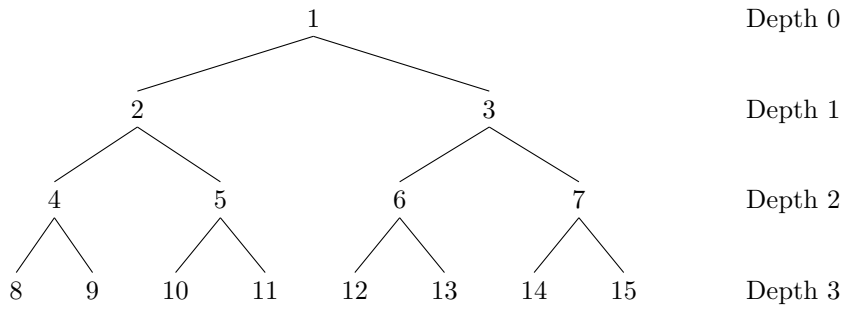

Figure 1: Search tree for Problem 2 (up to depth 3)

a) Breadth-first search: $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9 \to 10 \to 11$
Depth-first search (limit = 3): $1 \to 2 \to 4 \to 8 \to 9 \to 5 \to 10 \to 11$
Iterative deepening search:

- Iteration 1 (limit = 0): 1
- Iteration 2 (limit = 1): $1 \to 2 \to 3$
- Iteration 3 (limit = 2): $1 \to 2 \to 4 \to 5 \to 3 \to 6 \to 7$
- Iteration 4 (limit = 3): $1 \to 2 \to 4 \to 8 \to 9 \to 5 \to 10 \to 11$

b) The bidirectional search works well in this situation, as it visits less states than other search algorithms:

$\text{visited}_{\text{forward}}, \text{visited}_{\text{backward}} = 1, 11 \rightarrow 2, 5 \rightarrow 3, 2$
The branching factor would be 2 in the forward direction (each state has two successors) and 1 in the backward direction (each state has one predecessor).

3 Problem 3

a) True; when all step costs are equal, the path cost of node n is proportional to its depth.

b) True; depth-first search is best-first search with $f(n) = -\text{depth}(n)$.

c) True; uniform-cost search is A* search with a heuristic of $h(n) = 0$.

d) False; depth-first search returns the first path found that contains the goal.

e) False; breadth-first search is only guaranteed to return an optimal solution if the costs of all the edges are equal.

f) True; uniform-cost search expands paths in order of least total cost.

g) True; if a heuristic is consistent, then it is optimal.

h) False; A* search with a consistent heuristic always returns an optimal solution while DFS does not, and finding a sub-optimal solution could require less expanded nodes than an optimal one.

i) True; with the A* search heuristic being consistent, the most extreme outcome would be the two searches expanding the same number of nodes.

4 Problem 4

- Advantage: BFS has a space complexity of $O(b^d)$ while iterative deepening has a (lower) space complexity of $O(bd)$.
- Disadvantage: Interactive deepening does repeated computations and can revisit nodes multiple times, causing it to generate more nodes compared to BFS.

5 Problem 5

Given that $h(n)$ is consistent, prove that $h(n)$ is admissible.
In other words, given that $h(n) \leq c(n, a, n') + h(n')$ and $h(\text{goal}) = 0$, where $c(n, a, n')$ is the step cost of going from n to n' through action a, prove $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost of the shortest path from n to the goal.
Proof by induction:

- Base case: Let n' be any goal node, such that there is only 1 node on the shortest path to any goal from n. This means that $c(n, a, n') + h(n') = h^*(n) + 0 = h^*(n)$. Since $h(n) \leq c(n, a, n') + h(n')$, the statement $h(n) \leq h^*(n)$ holds for the base case.
- Assume $h(n) \leq h^*(n)$ holds when there are k nodes on the shortest path to any goal from n.
  To prove that the statement holds when there are $k + 1$ nodes on the shortest path to any goal from n, let n' be on the shortest path k steps from the goal, such that $h(n') \leq h^*(n')$. In other words, $h(n) \leq c(n, a, n') + h(n') \leq c(n, a, n') + h^*(n')$. Since $h^*(n) = c(n, a, n') + h^*(n')$, this means that $h(n) \leq h^*(n)$.
- As a result, we can conclude that when $h(n) \leq c(n, a, n') + h(n')$ and $h(\text{goal}) = 0$, the statement $h(n) \leq h^*(n)$ is true in all cases: if the heuristic $h(n)$ is consistent, it must be admissible.

Example of an admissible heuristic that is not consistent:

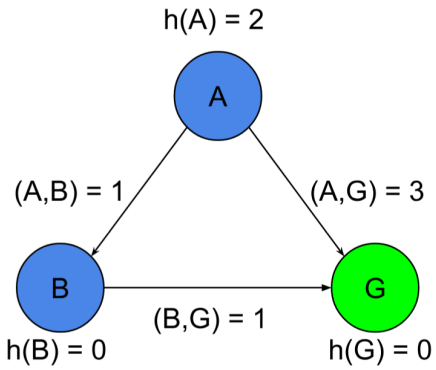

Figure 2: Example graph, with step costs and $h(n)$ values labeled

- Let A be the starting position and G be the goal
- Admissible
  - The cost of h*(A) is 2 (Path from A to B to G) and h(A)=2, therefore h(A) ≤ h*(A) holds because 2≤2 making it admissible.
  - The cost of h*(B) is 1 (Path from B to G) and h(B)=0, therefore h(B) ≤h*(B) holds because 0≤1 making it admissible
  - Both h*(A) and h*(B) are admissible
- Consistent
  - A⟶B: $h(A) \leq c(A, B) + h(B)$ equating to $2 \leq 1 + 0$ which is not consistent

- B⟶G: $h(B) \leq c(B,G) + h(G)$ equating to $1 \leq 1 + 0$ which is consistent
- A⟶B is not consistent, while B⟶G is consistent. Since A⟶B is not consistent we have successfully showed an example of an admissible heuristic that is not consistent.

6 Problem 6

The most constrained variable is more likely to cause a failure early on than other variables (all other things being equal). As a result, it is more likely to prune large parts of the search tree, reducing the amount of time spent searching.
The least constraining value rules out the fewest choices for the neighboring variables in the constraint graph. As a result, it provides the greatest chance for future variable assignments to avoid conflict and for a solution to be found.

7 Problem 7

a) The best move for the MAX player is C.

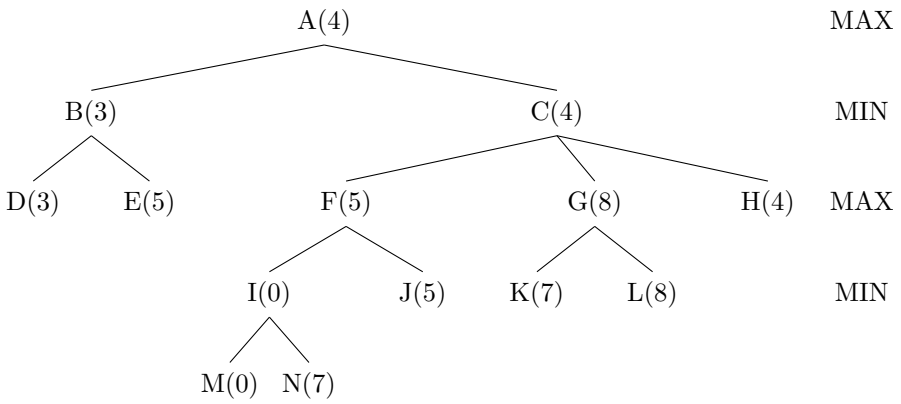

Figure 3: Game tree for Problem 7

b)

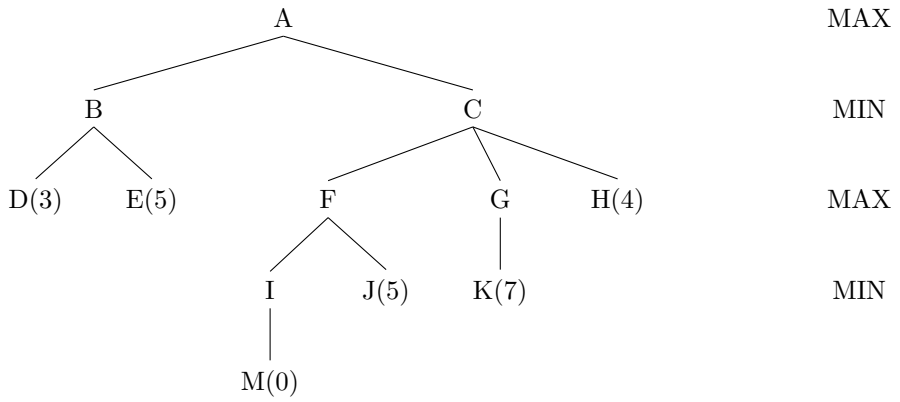

Figure 4: Figure 3 with left-to-right alpha-beta pruning applied

c) The results of right-to-left alpha-beta pruning are different than left-to-right alpha-beta pruning because simply put taking different paths results in a different criteria. Let us take a subtree with C as the root node, when we do left-to-right we have to find the value of F by going through its children and its children's children, eventually coming to the conclusion of F = 5. This means after C's first child it will only want a path that is less than 5 because C = MIN(F,G,H). Now let us do right-to-left, we immediately find out that H=4 meaning C will only want a path that is less than 4 because once again C = MIN(F,G,H). Just like that after only 1 child in both iterations the parameters we are trying to fill are already different, in addition you also have to account for the fact that each pruning visit branches in different orders.

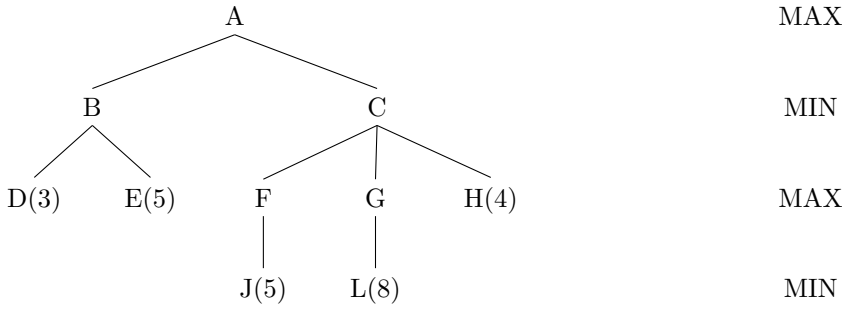

Figure 5: Figure 3 with right-to-left alpha-beta pruning applied

8 Problem 8

a) $h(n) = \min\{h_1(n), h_2(n)\}$
$h(n)$ is admissible if $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost of the short-

est path from n to the goal. Since h_1 and h_2 are admissible, $h_1(n) \leq h^*(n)$ and $h_2(n) \leq h^*(n)$. This means that $h(n) \leq h^*(n)$ since $\min\{h_1(n), h_2(n)\}$ either equals $h_1(n)$ or $h_2(n)$, making $h(n)$ admissible.
For $h(n)$ to be consistent, $h_{\min}(n) \leq c(n, a, n') + h_{\min}(n')$ must be true, where $h_{\min}$ is the heuristic chosen by $h(n) = \min\{h_1(n), h_2(n)\}$. In other words, $h_{\min}$ either equals h_1 or h_2. Since we know that $h_1(n) \leq c(n, a, n') + h_1(n')$ and $h_2(n) \leq c(n, a, n') + h_2(n')$, $h(n)$ is consistent.

b) $h(n) = wh_1(n) + (1 - w)h_2(n)$, where $0 \leq w \leq 1$
Due to the fact that $h_1(n)$ and $h_2(n)$ are admissible, they do not exceed $h^*(n)$. Therefore, a weighted proportion that adds up to 1 among the two will still result in a sum less than or equal to $h^*(n)$, so $h(n)$ is admissible.
By definition of being consistent, $h_1(n) \leq c(n, a, n') + h_1(n')$ and $h_2(n) \leq c(n, a, n') + h_2(n')$. Therefore $wh_1(n) + (1 - w)h_2(n) \leq w(c(n, a, n') + h_1(n')) + (1 - w)(c(n, a, n') + h_2(n'))$. This simplifies to $wh_1(n) + (1 - w)h_2(n) \leq c(n, a, n') + h(n')$, making $h(n)$ consistent.

c) $h(n) = \max\{h_1(n), h_2(n)\}$
Since h_1 and h_2 are admissible, we know that $h_1(n) \leq h^*(n)$ and $h_2(n) \leq h^*(n)$. This means that $h(n) \leq h^*(n)$ since $\max\{h_1(n), h_2(n)\}$ either equals $h_1(n)$ or $h_2(n)$, making $h(n)$ admissible.
For $h(n)$ to be consistent, $h_{\max}(n) \leq c(n, a, n') + h_{\max}(n')$ must be true, where $h_{\max}$ is the heuristic chosen by $h(n) = \max\{h_1(n), h_2(n)\}$. In other words, $h_{\max}$ either equals h_1 or h_2. Since we know that $h_1(n) \leq c(n, a, n') + h_1(n')$ and $h_2(n) \leq c(n, a, n') + h_2(n')$, $h(n)$ is consistent.

Let C* be the cost of the optimal solution path, such that A* expands all nodes with $f(n) <$ C* (and possibly some nodes with $f(n) =$ C*). In other words, A* search expands all nodes with $h(n) <$ C* - $g(n)$, so a heuristic with higher values generally leads to less expanded nodes. As a result, out of the three heuristics, (c) or $h(n) = \max\{h_1(n), h_2(n)\}$ would be the preferred heuristic for A*, as it chooses whichever component heuristic (h_1 or h_2) dominates the other (and therefore expands less nodes).

9 Problem 9

a) As long as there are no local maxima or plateaus, hill climbing would work better than simulated annealing. Due to its simplicity, hill climbing could also be better in situations where speed is an important factor.

b) Randomly guessing the state works just as well as simulated annealing when the cost function lacks a clear structure. This means the region around the global maximum appears indistinguishable from other areas of the function, making you unable to tell if you're approaching the solution. Also, the function's shape, assuming no real structure, wouldn't be able to guide you toward the correct direction to find an solution.

c) Simulated annealing is a useful technique when there are local optima and a complex landscape, i.e. a clear structure which may have a clear direction leading towards an optimal solution.

d) Instead of returning the last visited state, we could return the best valued state seen so far: given we know the value of each state we visited, we keep track of the best solution encountered, which helps make sure we don't end up returning a bad final state due to the randomness of simulated annealing.

e) Looping back onto the previous answer, one could use the extra memory to store and reuse the best states found. So when restarting, instead of selecting a random restart it can restart from the pool of best states, which would be more effective and efficient.

f) By adapting randomness in simulated annealing to gradient ascent search (instead of always moving in the direction that follows the gradient), randomness could be used to explore other directions/parts of the function. Another thing is that if gradient search doesn't seem to be making much progress or there's some sort of stagnation, a random reset could be done to help start at a different point, potentially based on past best solutions.