AI-Assignment 2

Siya B(214007906), Raashi M(215008169), Paige B(217002987)

April 2025

1. Question 1

   - (Lugoj, 244, 0, 244)
   - (Mehadia, 311, 70, 241)
   - (Drobeta, 387, 145, 242)
   - (Craiova, 425, 263, 160)
   - (Timisoara, 440, 111, 329)
   - (Pitesti, 503, 403, 100)
   - (Bucharest, 504, 504, 0)

Node	**Total Cost (f)**	**Path Cost (g)**	**Heuristic (h)**
Lugoj	244	0	244
Mehadia	311	70	241
Drobeta	387	145	242
Craiova	425	263	160
Timisoara	440	111	329
Pitesti	503	403	100
Bucharest	504	504	0

   (a) **(Lugoj, 244, 0, 244)**
   Open Nodes: Lugoj
   Expand Lugoj.

   (b) **(Timisoara, 440, 111, 329), (Mehadia, 311, 70, 241)**
   Open Nodes: Mehadia, Timisoara
   Expand Mehadia as $311 < 440$.

   (c) **(Drobeta, 387, 145, 242)**
   Lugoj is not considered as it has already been visited.
   Open Nodes: Drobeta, Timisoara
   Expand Drobeta since 387 is still less than 440 from Timisoara

   (d) **(Craiova, 425, 263, 160)**
   Open Nodes: Craiova, Timisoara
   Expand Craiova since 425 is still less than 440 from Timisoara

(e) **(Pitesti, 503, 403, 100), (Rimnicu Vilcea, 604, 411, 193)**
Open Nodes: Timisoara, Pitesti, Rimnicu Vilcea
Expand Timisoara since it has not been explored yet and 440 is less than both 503 (Pitesti's $f(n)$ and 604 (Riminicu Vilcea's $f(n)$)

(f) **(Arad, 595, 229, 366)**
Open Nodes: Pitesti, Arad, Rimnicu Vilcea
Expand Pitesti (Pitesti's f value is less than Arad's since 503 ¡ 595)

(g) **(Bucharest, 504, 504, 0)**
Open Nodes: Bucharest, Arad, Rimnicu Vilcea
Expand Bucharest (since 504 ¡ 595 (Arad's $f(n)$ value).
After expansion, the only open nodes remaining would be Rimnicu Vilcea. At this point, our goal has been reached so the algorithm stops here.

2. Question 2

Search Methods

a)

- **i) BFS:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
- **ii) DLS (with limit 3):** 1, 2, 4, 8, 4, 9, 4, 2, 5, 10, 5, 11
- **iii) Iterative Deepening Search:** 1, 1, 2, 3, 1, 2, 4, 8, 4, 2, 5, 2, 1, 3, 6, 3, 7, 1, 2, 4, 8, 4, 9, 4, 2, 5, 10, 5, 11

b) Bidirectional Search

- **i) Description:** Bi-directional search would work very well because it meets in the middle, which drastically reduces the search space.
- **ii) Branching factor:**
  - Forward: 2
  - Backward: 1
- **iii) Backward Search Path:** 11, 5, 2, 1
- **iv) Forward Search Path:** 1, 2, 5
- **v) Final Path:** 1, 2, 5, 11

3. Question 3

(a) Breadth-first search is a special case of uniform-cost search.
Answer: This statement is true. BFS is a special case of UCS where the neighboring nodes (edges) have the same cost of 1. UCS will operate like BFS when the cost to go to every child is a constant. UCS expands to the lowest path instead of the shallowest node, the only difference between BFS and UCS.

(b) Depth-first search is a special case of best-first tree search.
Answer: This statement is true. Best-first search expands the states with the lowest f-value. The best first tree will expand a child, then expand its children (since the children will always have lower f values) when the f-value is equal to the negative depth of the tree. The process is the same as the one DFS carries out.

(c) Uniform-cost search is a special case of A* search.
Answer: This statement is true. UCS is A* with $h(n) = 0$, which means A* behaves the same as UCS without heuristic guidance. If the heuristic was always zero, then A* will be the same as uniform-cost search

(d) Depth-first graph search is guaranteed to return an optimal solution.
Answer: This statement is false. DFS may find suboptimal solutions because it has a nature of going deep before considering alternative paths. It returns the first path that it finds that contains the goal, not the best path.

(e) Breadth-first graph search is guaranteed to return an optimal solution.
Answer: This statement is false. BFS only returns an optimal solution if all the costs of the edges are equal.

(f) Uniform-cost graph search is guaranteed to return an optimal solution.
Answer: This statement is true. UCS always finds the cheapest path if the function is non-negative, as it expands the least-cost node first.

(g) A* graph search is guaranteed to return an optimal solution if the heuristic is consistent.
Answer: This statement is true. A consistent heuristic ensures optimality in A* as it never overestimates the true cost to the goal.

(h) A* graph search is guaranteed to expand no more nodes than depth-first graph search if the heuristic is consistent.
Answer: This statement is false. DFS can sometimes be faster due to luck. It may explore a promising path early but lacks a guarantee of optimality.

(i) A* graph search is guaranteed to expand no more nodes than uniform-cost graph search if the heuristic is consistent.
Answer: This statement is true. A* with $h(n)$ dominates UCS in efficiency because it prioritizes nodes based on $f(n) = g(n) + h(n)$.

4. Question 4

Problem 4: Iterative Deepening vs. BFS

Advantage: Iterative deepening uses less memory compared to Breadth-First Search (BFS) since it does not store all the nodes at the current depth level.
Disadvantage: Iterative deepening can be more computationally expensive compared to BFS because it repeatedly expands nodes at shallower depths.
Runtimes: The time complexity of BFS is $O(b^d)$, where b is the branching factor and d is the depth of the shallowest goal. The time complexity of iterative deepening is $O(b^d)$, while the space complexity is $O(d)$ since only the current path is stored in memory.

5. Question 5

Problem 5: Consistent and Admissible Heuristics

Proof by Induction: A heuristic $h(n)$ is consistent if for every node n and successor n' of n, the following holds:

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

Base Case: For the goal node G, $h(G) = 0$ is consistent since there are no successors. Therefore,

$$h(G) \leq c(G, G) + h(G) = 0$$

The base case holds true.

Inductive Step: Assume that the consistency condition holds for a path of length k. That is,

$$h(n_i) \leq c(n_i, n_{i+1}) + h(n_{i+1}) \quad \textbf{for all such that } 1 \leq i \leq k$$

Step 1: For a path of length $k + 1$, consider the last transition from n_k to n_{k+1}.

Step 2: By the inductive hypothesis, the consistency holds for the first k transitions:

$$h(n_i) \leq c(n_i, n_{i+1}) + h(n_{i+1}) \quad \textbf{for all such that } 1 \leq i \leq k$$

Step 3: Check the next transition $(k + 1)$:

$$h(n_k) \leq c(n_k, n_{k+1}) + h(n_{k+1})$$

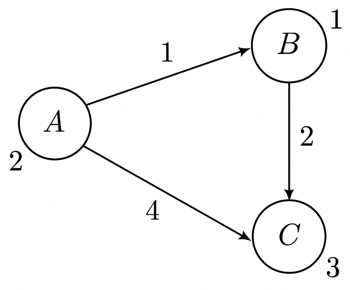
Step 4: By adding these inequalities, we obtain:

$$h(n_1) \leq \sum_{i=1}^{k} c(n_i, n_{i+1}) + h(n_{k+1})$$

Since consistency holds for each step from n_1 to n_{k+1}, the entire path of length $k+1$ must also be consistent.

Conclusion: By mathematical induction, consistency holds for any path length, and since a consistent heuristic also implies $h(n) \leq h^*(n)$, it must be admissible.

Counterexample: Consider the following graph with nodes A, B, and Goal: See Figure below



$$h(A) = 2, \quad h(B) = 1, \quad h(G) = 0$$

The heuristic is admissible as it does not overestimate the cost to reach the goal. However, it is not consistent because:

$$h(A) = 2 \not\leq c(A, B) + h(B) = 1 + 1 = 2$$

Thus, the heuristic is admissible but not consistent.

Question 6

Problem 6: CSP Heuristic

Choosing the most constrained variable first allows the algorithm to reduce the branching factor early on by assigning the variable that has the fewest possible values. Selecting the least constraining value minimizes conflicts for the remaining unassigned variables, thereby promoting future flexibility in assignments.

1 Problem 7: Min-Max Pruning

a. The best move for the MAX player to make using the Min-Max procedure is to move to node C. Moving to C will yield the result 4, while moving to node B will yield 3.

b. See figure below, left to right pruning

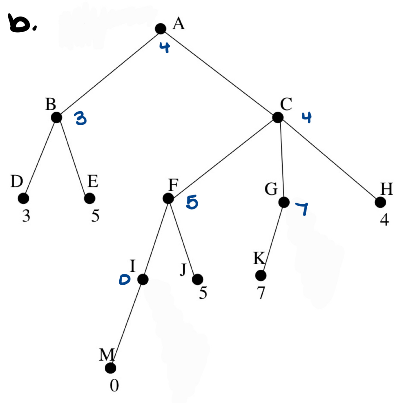


Figure 1: Left to Right Pruning

c. See figure below, right to left pruning; Different pruning occurs because

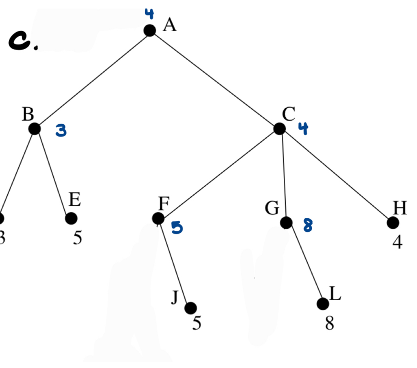


Figure 2: Right to Left Pruning

starting from right to left instead of left to right will yield different beta and alpha values. You're discovering the nodes and their values in a different order and because of that, you take on alpha/beta values at certain points later or earlier than moving in the opposite direction. Therefore since the sequence is different, the pattern of discovery is also different. However, this doesn't change the final answer; the number of nodes you expand to is just different.

2 Problem 8: Admissible & Consistent Heuristics

a. This heuristic is both admissible and consistent. It is admissible because by the definition of an admissible heuristic, where the true cost is never overestimated, because we are always finding the minimum of the two heuristics, we are never overestimating the true cost. The heuristic is not always consistent because using the inequality expression that proves consistency (a heuristic h(n) is consistent if, for every node n and any successor n' of n, the following inequality holds: $h(n) \leq c(n, a, n') + h(n')$); $h(n) = \min\{h_1(n), h_2(n)\} \leq \min\{c(n, n\prime) + h_1(n\prime),\, c(n, n\prime) + h_2(n\prime)\} = c(n, n\prime) + \min\{h_1(n\prime),\, h_2(n\prime)\}$.

b. Because each heuristic on its own is less than the true cost, a weighted combination of the two should be less than (or at most) the true cost. Therefore, the function is admissible. Since h_1 and h_2 are each consistent, you can show that a weighted sum of two consistent heuristics is also consistent. Using the previous inequality statement that proves consistency (from the previous question) if you move from n to $n\prime$ and that costs $c(n, n\prime)$, then $h_1(n) \leq c(n, n\prime) + h_1(n\prime)$, $h_2(n) \leq c(n, n\prime) + h_2(n\prime)$, and by combining them with weights w and 1-w, the inequality still holds. Therefore, the weighted combination is consistent.

c. If both heuristics for this function are admissible, that implies that they're at most the value of the true cost. Because of that, no matter which max value you take, it'll still be less than the true cost, therefore making the function admissible. If $h_1(n)$ happens to be larger than $h_2(n)$, it will be the active value at n, but then we only need to ensure the same consistency inequality that already holds for h_1. If h_1 and h_2 are both consistent, picking whichever is larger at each node still obeys a relatively unvarying pattern from node to node. If you move to a neighbor n', the cost plus $h_1 n'$ (or the maximum of the two at n') still satisfies the consistency condition. Hence $\max\{h_1, h_2\}$ is consistent.

We think that it would make the most sense to use the max function for A*; It gives a higher estimate than the minimum or a weighted average, and a higher heuristic value while staying admissible, leading to a better guided search, and not having to check as many nodes. Overall, this would lead to better optimality and a shorter search time (or rather, a reduction in the number of nodes that need to be expanded to).

3 Problem 9: Hill Climbing & Simulated Annealing

a. Hill climbing is better to use over simulated annealing when there are very few to one peak/local maxima that additionally do not become traps (you get stuck), or are easier to avoid/unlikely to get stuck in. The use of random jumps wouldn't help because you would be escaping from local maxima/minima that either aren't a big threat, or do not exist.

b. In situations with no local improvements, you can make random jumps instead of using simulated annealing. For example, if the states within the environment do not correlate and you can't gauge how you'll be able to move uphill, you might as well move randomly (this is also because you don't gain anything from using SA instead). You can also randomly jump if the environment is very flat, again because local moves do not lead to any improvements.

c. SA is most useful in scenarios where there are a lot of local maxima or minima and we want to find the global one, but the chance of getting stuck at one of these local areas is high. The random jumps would help to get out of these areas and put us onto the correct path to finding the global max/min (it is also worth noting that SA is good for environments where the max/min are not extremely random and very complicated). This is because accepting these random jumps (even occasional bad ones) still helps to escape local max/min, overall moving uphill.

d. To make improvements, we could keep track of the best state encountered thus far, and return to that state if we don't find a better one at the end of the process. This way, once we reach a termination state and have gone through as many states as possible, we can know that we found the best one that was possible to obtain during the process.

e. We could keep a list that maintains a list of the best states, and all states in general to prevent returning to a state that we already visited, but also keeping track of the best state to return to if needed. Additionally, at any point if we become stuck, we can jump back to a high-value state that was saved, and start working back from that state again (make local moves from there instead).

f. A possible way to adjust the randomness in simulated annealing to gradient ascent search is to introduce random deviations to the gradient ascent direction on each iteration. To achieve this, first update the current state and then add a random component to the gradient vector. This random component would be affected by an acceptance probability and a decreasing temperature to manage how big or how frequent these random deviations can be, and when (and how often) you accept worse solutions.