Name: Ochieng' Meshack Otieno
Adm.No: SCT212-0078/2023
Course: Computer Technology

Artificial Intelligence: Search in Pacman Assignment

Proofs required from Pacman Assignment.

Question 7 (10 points): Prove that consistency implies admissibility *and* give an example of an admissible heuristic that is not consistent (problem 3.29 from the text).

Proof:

Take h(n), where h is a heuristic, and n is the node. In this way, a heuristic h can be admissible or consistent, relative to every node n.

A h(n) is admissible if every node n, h(n) is less than or equal to h*(n), where h*(n) is the True Cost from n to the goal in the maze. Therefore, this can be expressed as:

$$h(n) <= h^*(n)$$

Let us assume that h is consistent, n to be any node and an optimal path to the goal g, where:

$$n = n_1, n_2, n_3, \ldots, n_k = g$$

and cost is:

$$h^*(n) = \sum^{k-1}, \text{ with } i=0.$$

By consistency, for each step i, we have:

$$h(n_i) <= c(n_i, n_{i+1}) + h(n_{i+1})$$

Since h(g)=0 for the goal, we get:

$$h(n) = h(n_0) <= \sum^{k-1} c(n_i, n_{i+1}) = h^*(n)$$

In this way, it is evident that h is admissible, with h(n)<= h*(n) for all n. Therefore, consistency implies admissibility.

Example:

Let h(S) = 2 and h(A) = 0.

To check for admissibility:

$h(S) = 2 <= h^*(S) = 2$. Hence this is admissible.

$h(A) = 0 <= h^*(A) = 1$. Hence, admissible

With the two being admissible, h is admissible. The consistency of S→A is 0→2<=1, which makes it false. Hence, because the consistency condition fails for S→A, h is not consistent.

Question 9 (5 points) Show that an inadmissible heuristic can lead to A* returning a suboptimal example, by giving an example search tree and heuristic where this occurs. Aim for as small a tree as possible, and explain your example clearly.

Proof:

Let S→A: cost 1

Let S→B: cost 2

Let A→G1: cost 1

Let B→G2: cost 1

True Cost to Goal will be:

Path S-A-G1: Total cost = 2

Path S-B-G2: total cost = 3

Inadmissible heuristic cost will be:

h(S) = 4

h(A) = 0

h(B) = 0

h(G1) = 0

By starting at A:

f(S) = g(S) + h(S) = 0 + 4 = 4

In this way, S can be expanded:

g(A)=1, f(A)=1 + 0 = 1

g(B) = 2, f(B) = 2 + 0 = 2

A is then expanded, with the lowest f-value:

Generation of G1: g(G1) = 2, f(G1) = 2 + 0

Hence, A* terminates and returns S-A-G1, with cost 2**, because heuristic inadmissibility h(S)=4 is overestimates the true cost to goal.

The suboptimal path: A* is realized to be returning S-A-G1 (which is cost 2) instead of continuing to find potentially better paths.

Question 10 (10 points) Give a detailed example (by drawing the states and labeling the edges with costs) where negative edge costs could lead to the following paradoxical result: a finite state space with an optimal solution that does not reach the goal in finite time. Be sure to explain why your example has this property.

Proof:

The proof starts by defining the State Space:
S (start), A, B, G (goal).

The edge cost (negative cycles) are:
S→A: cost 1

A→G: cost 1

S→B: cost 1

B→G: cost 1

A→S: cost -3 (negative)

B→S: cost -3 (negative)

The Paradox:

Optimal path Cost is negative infinity. This happens when the start is at S, and various paths are accounted for:

Start at S

Path 1: S→A→G: total cost = 1 + 1 = 2

Path 2: S→B→G: total cost = 1 + 1 = 2

A better path would be:

S→A→S→A→S→A→…→G

The S→A→S loop costs: 1 + (-3) = -2

If this is done k times, the total cost would reach: G = 2 + K*(-2).

Thus, k→∞(infinity), total cost→∞(infinity).

Why this is Paradoxical:

1. The Finite State Space is just 4 in total.
2. Optimal solution does exist, with the path having the lowest cost being -∞.
3. To achieve -∞, the path to be taken must be infinite: S→A→S loops.
4. Therefore, optimal solution never actually reaches G in finite time.

Question 11 (5 points) Suppose you are asked to apply search to a symbol problem. You are told that the solution is a sequence of at most five symbols, but the symbols are drawn from an alphabet which contains thousands of symbols. Formulate this problem as a search problem. If depth-first and breadth-first were your own choices, which would you apply? Explain. How might you change the algorithm?

Proof:

Search Problem Formulation.

State Space: All sequences of symbols from the alphabet $\sum$, where $|\sum| = 1000+$.

For States: All sequence of length 0 to 5 symbols.

Initial state starts at empty sequence of length 0.

Any valid solution sequence will satisfy problem constrains.

Append any symbol from $\sum$ to the current sequence.

Uniform cost (1 per symbol) or problem-specific cost function.

Representation of State

State = current sequence of symbols.

Maximum depth = 5 (sequence of length <= 5)

The Branching Factor is very high.

Algorithm Choice

I would choose Breadth-First Search algorithm because:

1. Memory Explosion is avoided; hence it is memory efficient. Level 0 to level 5 will be feasible.
2. With or without memory, DFS will use it resourcefully even as it approaches depth 5.
3. Early termination can stop at first solution found.

How I might change the algorithm:

I would use iterative deepening to improve the algorithm because of the following:

1. The completeness: guaranteed to find solution if it exists within depth 5.
2. It is optimal since it finds the shortest solution first.
3. It is memory efficient.
4. It won't get stuck in infinite branches.

Question 12 (5 points) Modify iterative deepening search so that it finds the lowest cost, as opposed to shortest, path. What information do you need? Prove that your algorithm finds the optimal solution.

Proof:

To modify iterative deepening to find the lowest cost path, we need to from depth-limited to cost-limited search. This is why:

This is proof by Optimality:

We start with monotonic Cost Increase: where the cost bound = 0 and incrementally increase it.

For any cost bound, the algorithm explores all paths with total cost <= C.

- For the solution to be optimal:
- Let C* be the cost of the optimal solution
- For all cost bound < C, no solution exists.
- When cost bound = C*, the optimal solution if found
- No solution with cost < C* exists.

Non-negative costs ensure progress. With non-negative cost, increasing cost bound eventually reaches C*. The algorithm cannot “skip over” the optimal solution in place.

There is also systemic exploration where the algorithm explores all paths in order of increasing total cost. It finds the first solution, which ends up being the lowest cost.

Information Needed:

Cost information from the problem

Non-negative costs

Cost function properties

Modified Algorithm components

Key Implementation details.