كلية العلـوم

قســـم علوم الذكاء الاصطناعي

المحاضرة الثالثة

المادة: Heuristic Search

المرحلة: الثانية

اسم الاستاذ: م.م هادي صلاح

Hill Climbing Algorithm

Hill climbing is a variant of the generate-and-test approach in which feedback from the test procedure is used to guide the generator in deciding which direction to move within the search space. In a pure generate-and-test procedure, the test function returns only a yes or no response. However, when the test function is augmented with a heuristic function, it provides an estimate of how close a given state is to a goal state.

This approach is particularly effective because, in many cases, the computation of the heuristic function can be performed at almost no additional cost while the test for a solution is being carried out. Hill climbing is commonly used when a good heuristic function is available for evaluating states, but when no other useful domain knowledge is accessible.

For example, suppose you are in an unfamiliar city without a map and you want to reach downtown. You simply move in the direction of the tall buildings. In this case, the heuristic function is the distance between the current location and the location of the tall buildings, and the desirable states are those in which this distance is minimized.

For each state, the evaluation function is defined as:

$$f(n) = h(n)$$

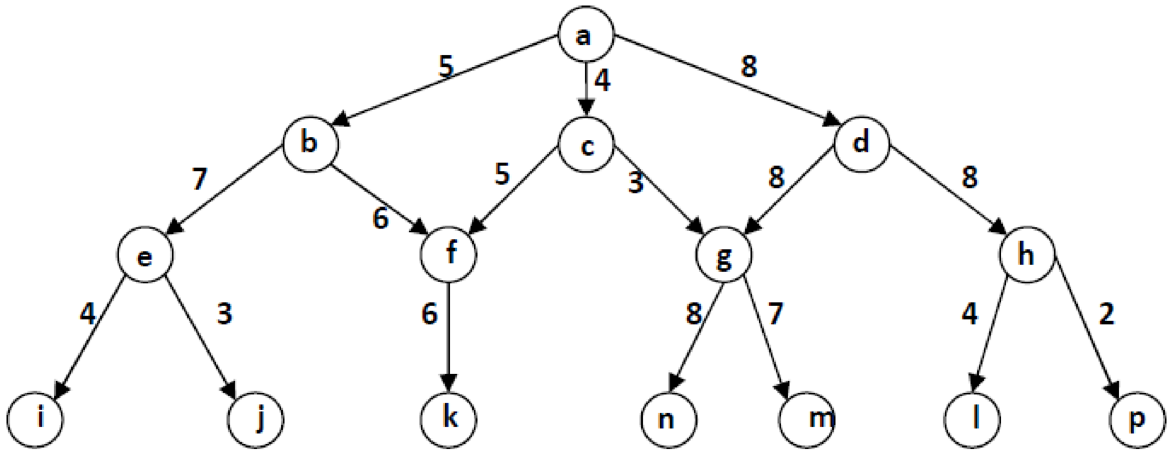
where $h(n)$ is the heuristic function that computes the heuristic value for each state n.

Hill Climbing Search Algorithm

```
Function Hill Climbing Search
Begin
Open := [Initial state];        % initialize
Closed := [ ];
CS := initial state;
Path := [initial state];
Stop := FALSE;
While Open <> [ ] do            % states remain
```

```
Begin
    If CS = Goal then
        Return Path;
    Generate all children of CS and put them into Open;
    If Open = [ ] then
        Stop := TRUE;
    Else
    Begin
        X := CS;
        For each state Y in Open do
        Begin
            Compute the heuristic value of Y, h(Y);
            If Y is better than X then
                X := Y;
        End;
        If X is better than CS then
            CS := X;
        Else
            Stop := TRUE;
    End;
End;
Return (FAIL);                    % Open is empty
End.
```

- *Consider the following problem state space then:*



Hill Climbing Search – Open and Closed Lists

Step / Iteration	Open	Closed
1	[a]	[]
2	[b5, c4, d8]	[a]
3	[c4, b5, d8]	[a]
4	[f5, g3, b5, d8]	[a, c4]
5	[g3, f5, b5, d8]	[a, c4]
6	[n8, m7, f5, b5, d8]	[a, c4, g3]
7	[m7, n8, f5, b5, d8]	[a, c4, g3]

Stop: the goal (m) is found.

Best First Search Algorithm

Now let us discuss a new heuristic method called Best First Search, which combines the advantages of both depth-first search and breadth-first search into a single search strategy.

The actual operation of the algorithm is very simple. It proceeds step by step, expanding one node at each step, until a node corresponding to a goal state is generated. At each step, the algorithm selects the most promising node among those that have been generated so far but not yet expanded.

After selecting a node, the algorithm generates its successor nodes, applies the heuristic function to each of them, and adds them to the list of open nodes after checking whether any of these nodes have been generated previously. This checking process guarantees that each node appears only once in the graph, although many nodes may still point to it as successors. The next step then begins.

For each state, the evaluation function is defined as:

$$f(n) = h(n)$$

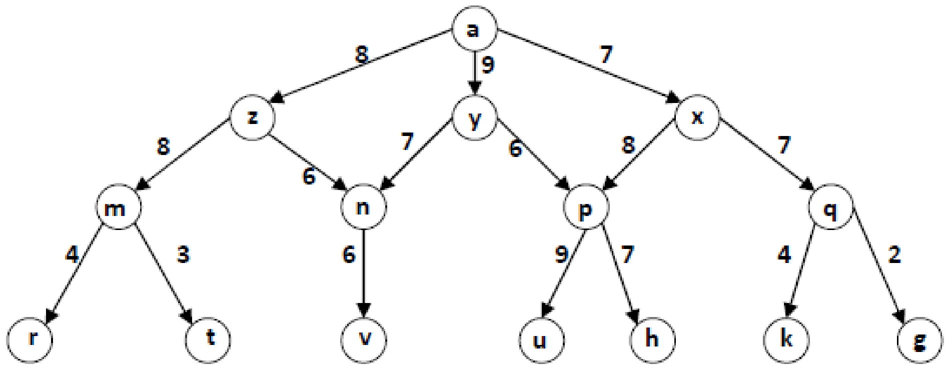
where $h(n)$ is the heuristic function that computes the heuristic value for each state n.

Best-First Search Algorithm (Pseudo-code)

```
Function Best-First Search

Begin
Open := [Initial state];         % initialize
Closed := [ ];
While Open <> [ ] do             % states remain
Begin
    Remove leftmost state from Open, call it X;
    If X = Goal then return the path from Initial to X
    Else
    Begin
        Generate children of X;
        For each child of X do Case
            The child is not on Open or Closed; Begin
                Assign the child a heuristic value;
                Add the child to Open
            End;
            The child is already on Open;
                If the child was reached by a shorter path
                Then give the state on Open the shorter path
            The child is already on Closed;
                If the child was reached by a shorter path then Begin
                    Remove the state from Closed;
                    Add the child to Open
                End;
        End;          % Case
        Put X on Closed;
        Re-order states on Open by heuristic merit (best leftmost)
    End;
End;
Return FAIL          % Open is empty
End.
```

Consider the following problem state space then:



Best First Search – Open and Closed Lists (a → k)

STEP / ITERATION	OPEN	CLOSED
1	[a]	[]
2	[z8, y9, x7]	[a]
3	[x7, z8, y9]	[a]
4	[p8, q7, z8, y9]	[a, x7]
5	[q7, p8, z8, y9]	[a, x7]
6	[k4, g2, p8, z8, y9]	[a, x7, q7]
7	[g2, k4, p8, z8, y9]	[a, x7, q7]
8	[k4, p8, z8, y9]	[a, x7, q7, g2]

The goal (k) is found.