
# Artificial Intelligence

## Lecture Nine: Heuristic Search

By: Dr. Zied O. Ahmed

# LECTURE EIGHT: HEURISTIC SEARCH

## 9.1 INFORMED (HEURISTIC) SEARCH STRATEGIES

This section shows how an **informed search strategy**—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy

In state space search, *heuristics* are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution.

AI problem solvers employ heuristics in two basic situations:

1. A problem may not have an exact solution because of inherent ambiguities in the problem statement or available data. Medical diagnosis is an example of this.

2. A problem may have an exact solution, but the computational cost of finding it may be prohibitive.

A *heuristic* is only an informed guess of the next step to be taken in solving a problem. It is often based on experience or intuition.

A heuristic can lead a search algorithm to a suboptimal solution or fail to find any solution at all.

## 9.2 Hill-Climbing Search Algorithm

The simplest way to implement heuristic search is through a procedure called *hill-climbing*.

Hill-climbing strategies expand the current state of the search and evaluate its children.

The "best" child is selected for further expansion; neither its siblings nor its parent are retained.

Hill climbing is named for the strategy that might be used by an eager, but blind mountain climber: go uphill along the steepest possible path until you can go no farther up. Because it keeps no history, the algorithm cannot recover from failures of its strategy.

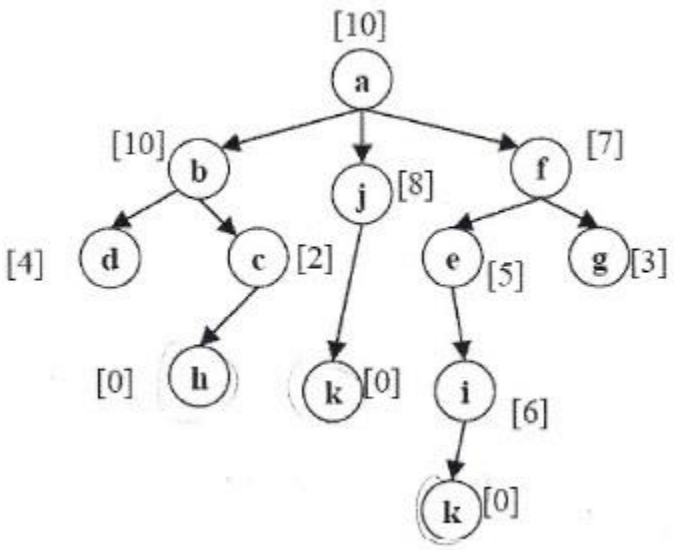
The idea here is that, you don't keep the big list of states around you just keep track of the one state you are considering, and the path that got you there from the initial state. At every state you choose the state leads you closer to the goal (according to the heuristic estimate), and continue from there.

The algorithm is followed as:

Step-1: Generates the best possible solutions

Step-2: Evaluate itself to see whether the best possible solution is the well-expected solution or not.

Step-3: If it suits the solution or the solution is found, then it quits; otherwise, it goes back to Step-1




There are basically 3 types of Hill climbing algorithms

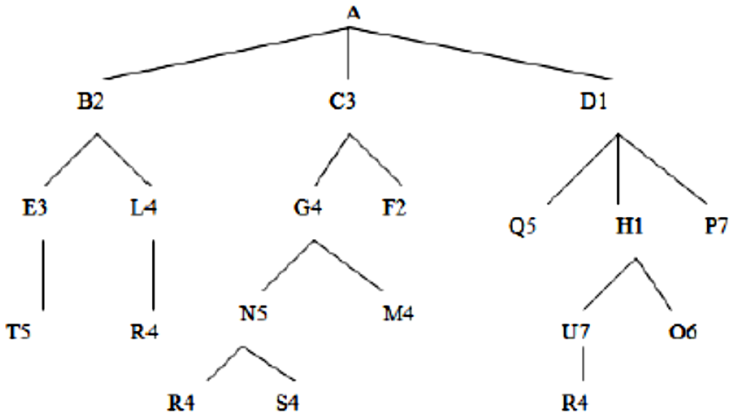
- Simple Hill Climbing:
- Steepest Ascent Hill Climbing
- Stochastic Hill Climbing

Hill climbing is an optimization technique, and problems with optimization techniques are to be trapped in local optima instead of global optima (the desired state). In hill climbing, basically, there are three main problems.

- Local maxima
- Plateau
- Ridges

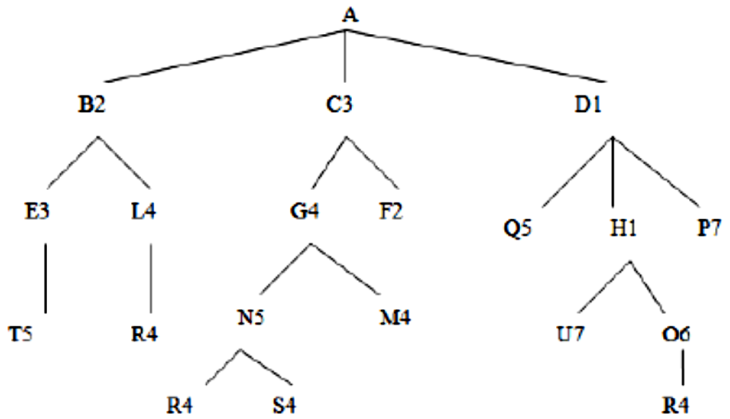
A major problem of hill-climbing strategies is their tendency to become stuck at *local maxima*. If they reach a state that has a better evaluation than any of its children, the algorithm falters. If this state is not a goal, but just a local maximum, the algorithm may fail to find the best solution.

## A trace of Hill Climbing Algorithm
## LOCAL MAXIMA



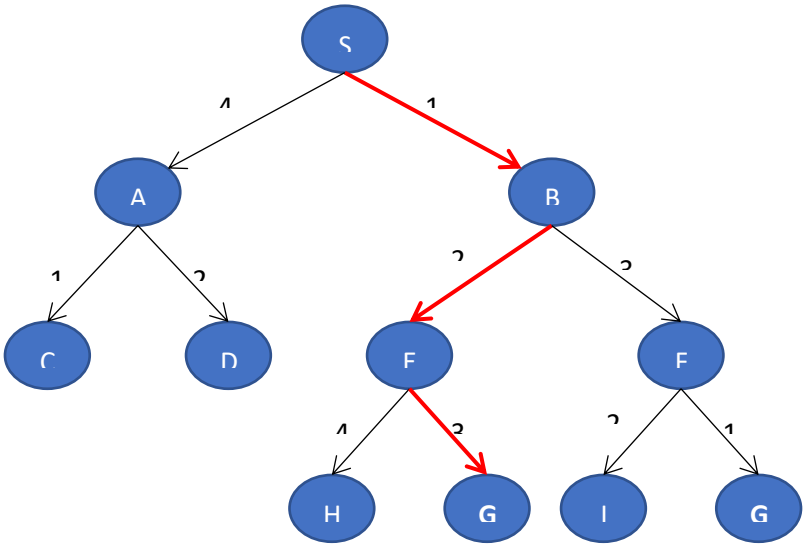

| | | |
|---|---|---|
| Open=[A] | Closed=[] | A |
| Open=[C3,B2,D1] | Closed=[A] | C3 |
| Open=[G4,F2] | Closed=[A, C3] | G4 |
| Open=[N5,M4] | Closed=[A,C3,G4] | N5 |
| Open=[R4,S4] | closed=[A,C3,G4,N5] | R4 |

# A trace of Hill Climbing Algorithm
# LOCAL MINIMA



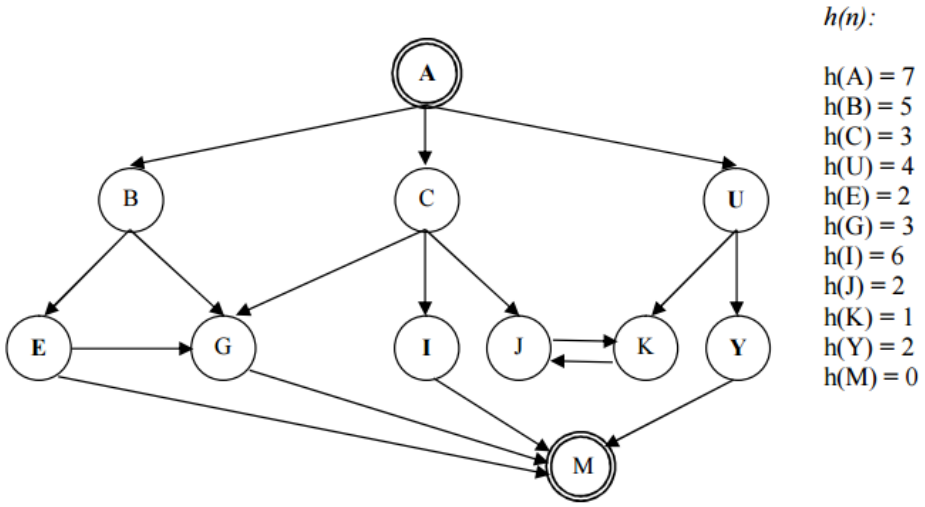

| | | |
|---|---|---|
| Open=[A] | Closed=[] | A |
| Open=[D1,B2,C3] | Closed=[A] | D1 |
| Open=[H1,Q5,P7] | Closed=[A, D1] | H1 |
| Open=[O6,U7] | Closed=[A,D1,H1] | O6 |
| Open=[R4] | closed=[A,D1,H1,O6] | R4 |
| | Closed=[A,D1,H1,O6,R4] | |

Examples:




**S → B → E → G1**

**Cost = 1+2+3=6**




## 9.3 The Best-First Search Algorithm

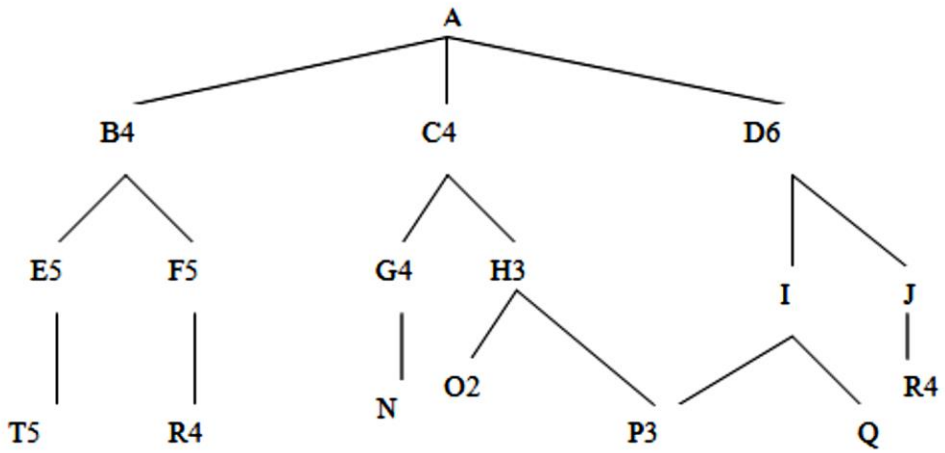
Best First search is a way of combining the advantages of both depth-first and breadth-first search in to a single method.

In Best-First search, the search space is evaluated according to a heuristic function. Nodes yet to be evaluated are kept on an OPEN list and those that have already been evaluated are stored on a CLOSED list. The OPEN list is represented as a priority queue, such that un visited nodes can be queued in order of their evaluation function. The evaluation function f(n) is made from only the heuristic function (h(n)) as: f(n)=h(n).

Best First Search Algorithm

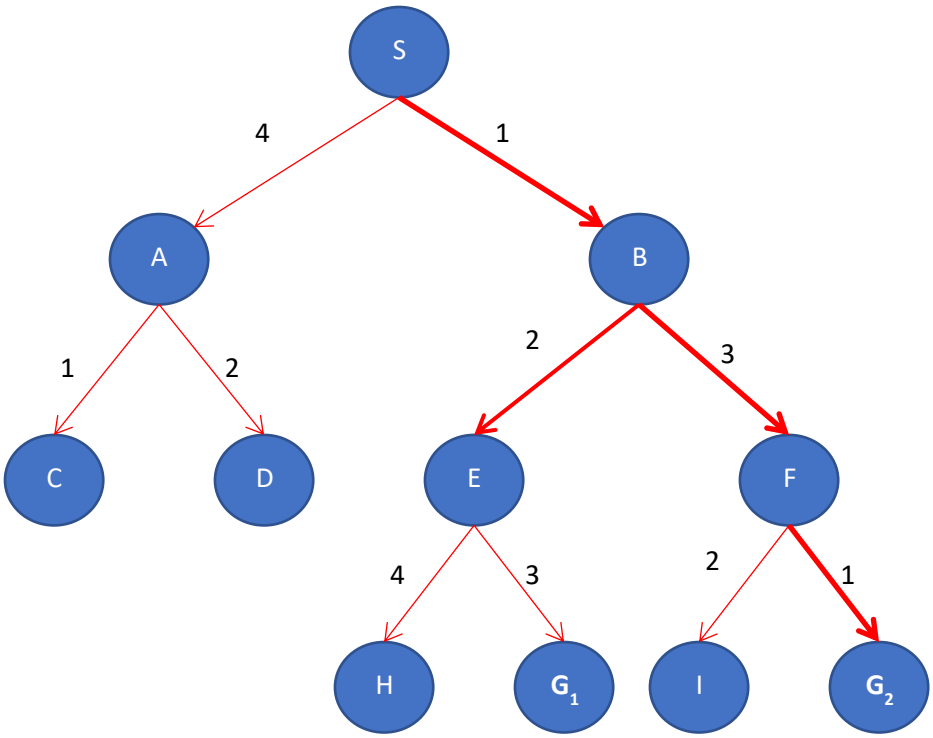
- Create 2 empty lists: **OPEN** and **CLOSED**
- Start from the initial node (say N) and put it in the 'ordered' **OPEN** list
- Repeat the next steps until **GOAL** node is reached
  1. If **OPEN** list is empty, then **EXIT** the loop returning 'False'
  2. Select the first/top node (say N) in the **OPEN** list and move it to the **CLOSED** list. Also capture the information of the parent node
  3. If N is a **GOAL** node, then move the node to the **CLOSED** list and exit the loop returning 'True'. The solution can be found by backtracking the path
  4. If N is not the **GOAL** node, expand node N to generate the 'immediate'

next nodes linked to node N and add all those to the **OPEN** list
5. Reorder the nodes in the **OPEN** list in ascending order according to an evaluation function f(n)




| | |
|---|---|
| Open=[A5] | Closed=[] |
| Open=[B4,C4,D6] | Closed=[A5] |
| Open=[C4,E5,F5,D6] | Closed=[B4,A5] |
| Open=[H3,G4,E5,F5,D6] | Closed=[C4,B4,A5] |
| Open=[O2,P3,G4,E5,F5,D6] | Closed=[H3,C4,B4,A5] |
| Open=[P3,G4,E5,F5,D6] | Closed=[O2,H3,C4,B4,A5] |
| Open=[G4,E5,F5,D6] | Closed=[P3,O2,H3,C4,B4,A5] |

**The solution path is**: A5 -B4 -C4 -H3 –O2-P3



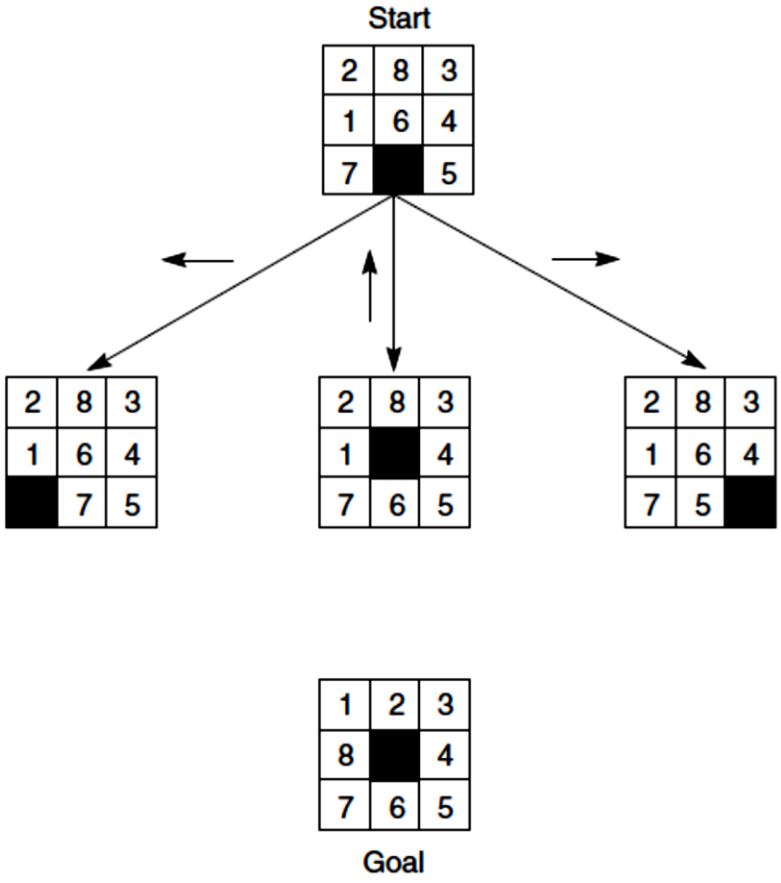

| | |
|---|---|
| open=[S0]; | closed=[ ] |
| open=[B1 , A4]; | closed=[S0] |
| open=[E3 , A4, F4 ]; | closed=[S0 , B1] |
| open=[A4 , F4 , G16 , , H7]; | closed=[S0 , B1 , E3] |
| open=[F4 , C5 , G16, D6 , H7]; | closed=[S0 , B1 , E3 , A4] |
| open=[C5 , G25 , G16 , I6 , D6 , H7]; | closed=[S0 , B1 , E3 , A4 , F4] |
| open=[G25 , G16 , I6 , D6 , H7]; | closed=[S0 , B1 , E3 , A4 , F4 , C5] |

**The solution path is**: [$S_0$ , $B_1$ , $E_3$ , $A_4$ , $F_4$ , $C_5$ , $G2_5$]

Cost = 1+3+1=5

## 9.4 Implementing Heuristic Evaluation Functions

We next evaluate the performance of several different heuristics for solving the 8-puzzle. Figure below shows a start and goal state for the 8-puzzle, along with the first three states generated in the search.




The simplest heuristic counts the tiles out of place in each state when compared with the goal. This is intuitively appealing, because it would seem that, all else being equal, the state that had fewest tiles out of place is probably closer to the desired goal and would be the best to examine next.

A "*better*" heuristic would sum all the distances by which the tiles are out of place, one for each square a tile must be moved to reach its position in the goal state.

| | | | |
|---|---|---|---|
| 2 8 3<br>1 6 4<br>■ 7 5 | **5** | **6** | **0** |
| 2 8 3<br>1 ■ 4<br>7 6 5 | **3** | **4** | **0** |
| 2 8 3<br>1 6 4<br>7 5 ■ | **5** | **6** | **0** |
| | Tiles out of place | Sum of distances out of place | 2 x the number of direct tile reversals |

1 2 3
8 ■ 4
7 6 5

Goal