SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

Course Name – Introduction to Artificial Intelligence

II Year / III Semester

Unit 1 Heuristic search,Hill climbing,Best first

Heuristic Search

- Heuristic - a “rule of thumb” used to help guide search
  - often, something learned experientially and recalled when needed
- Heuristic Function - function applied to a state in a search space to indicate a likelihood of success if that state is selected
  - heuristic search methods are known as “weak methods” because of their generality and because they do not apply a great deal of knowledge
  - the methods themselves are not domain or problem specific, only the heuristic function is problem specific
- Heuristic Search –
  - given a search space, a current state and a goal state
  - generate all successor states and evaluate each with our heuristic function
  - select the move that yields the best heuristic value
- Here and in the accompanying notes, we examine various heuristic search algorithms
  - heuristic functions can be generated for a number of problems like games, but what about a planning or diagnostic situation?

Example Heuristic Function

- Simple heuristic for 8-puzzle:
  - add 1 point for each tile in the right location
  - subtract 1 point for each tile in the wrong location
- Better heuristic for 8-puzzle
  - add 1 point for each tile in the right location
  - subtract 1 point for each move to get a tile to the right location
- The first heuristic only takes into account the local tile position
  - it doesn't consider such factors as groups of tiles in proper position
  - we might differentiate between the two types of heuristics as *local* vs *global*

Goal:

1	2	3
4	5	6
7	8	

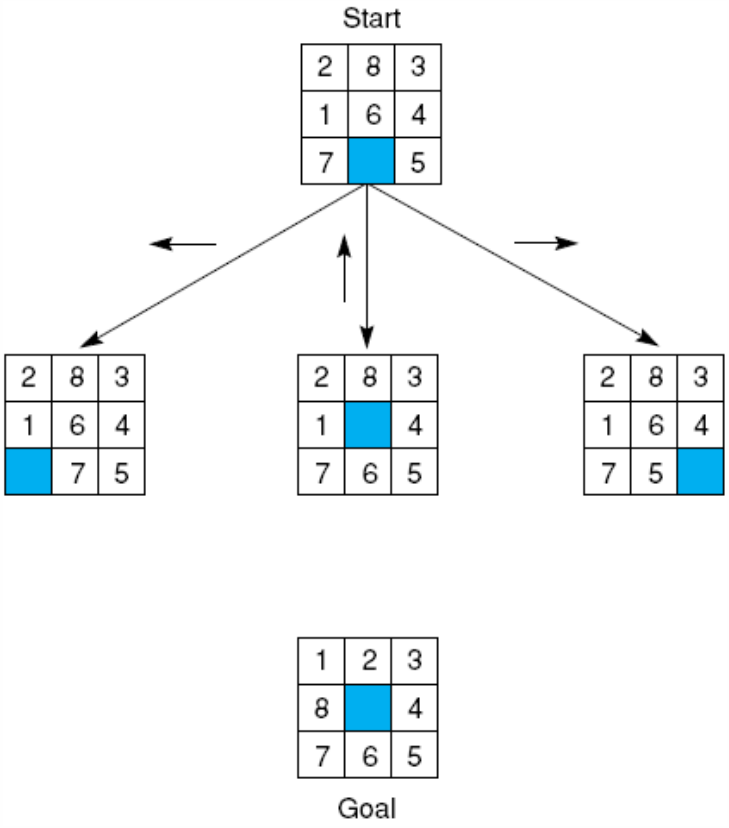
Current:

4	2	3
5	7	1
6		8

Moves:

7 down (simple: -5, better: -8)
6 right (simple: -5, better: -8)
8 left (simple: -3, better: -7)

Example Heuristics: 8 Puzzle



2 8 3 1 6 4 7 5	5	6	0
2 8 3 1 4 7 6 5	3	4	0
2 8 3 1 6 4 7 5	5	6	0
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

From the start state, which operator do we select (which state do we move into)? The first two heuristics would recommend the middle choice (in this case, we want the lowest heuristic value) while the third heuristic tells us nothing useful (at this point because too much of the puzzle is not yet solved)

Hill Climbing

- Visualize the search space as a 3-dimensional space
  - a state is located at position <x, y> where these values represent the state's variables, and its z value (height) is its heuristic worth
    - this creates a topology where you want to reach the highest point
  - in actuality, most problems have states that have more than just <x, y> values
    - so in fact, hill climbing takes place in some n+1 dimensions where n is the number of variables that define the state and the last value is the heuristic value, again, indicated as height
  - to solve a problem, pick a next state that moves you "uphill"
- Given an initial state perform the following until you reach a goal state or a deadend
  - generate all successor states
  - evaluate each state with the heuristic function
  - move to the state that is highest
- This algorithm only tries to improve during each selection, but not find the best solution

Variations of Hill Climbing

- In *simple hill climbing*, generate and evaluate states until you find one with a higher value, then immediately move on to it
- In *steepest ascent hill climbing*, generate all successor states, evaluate them, and then move to the highest value available (as long as it is greater than the current value)
  - in both of these, you can get stuck in a local maxima but not reach a global maxima
- Another idea is *simulated annealing*
  - the idea is that early in the search, we haven't invested much yet, so we can make some downhill moves
    - in the 8 puzzle, we have to be willing to "mess up" part of the solution to move other tiles into better positions
  - the heuristic worth of each state is multiplied by a probability and the probability becomes more *stable* as time goes on
    - simulated annealing is actually applied to neural networks

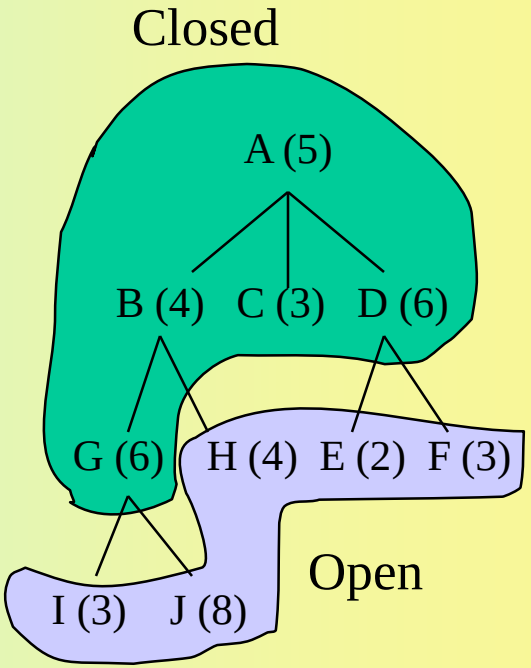
Note: we are skipping dynamic programming, a topic more appropriate for 464/564

Best-first search

- One problem with hill climbing is that you are throwing out old states when you move uphill and yet some of those old states may wind up being better than a few uphill moves
  - the best-first search algorithm uses two sets
  - open nodes (those generated but not yet selected)
  - closed nodes (already selected)
    - start with Open containing the initial state
    - while current <> goal and there are nodes left in Open do
      - set current = best node* in Open and move current to Closed
      - generate current's successors
      - add successors to Open if they are not already in Open or Closed

•- this requires searching through the list of Open nodes, or using a priority queue

Below, after exploring A's children, we select D. But E and F are not better than B, so next we select B, followed by G.

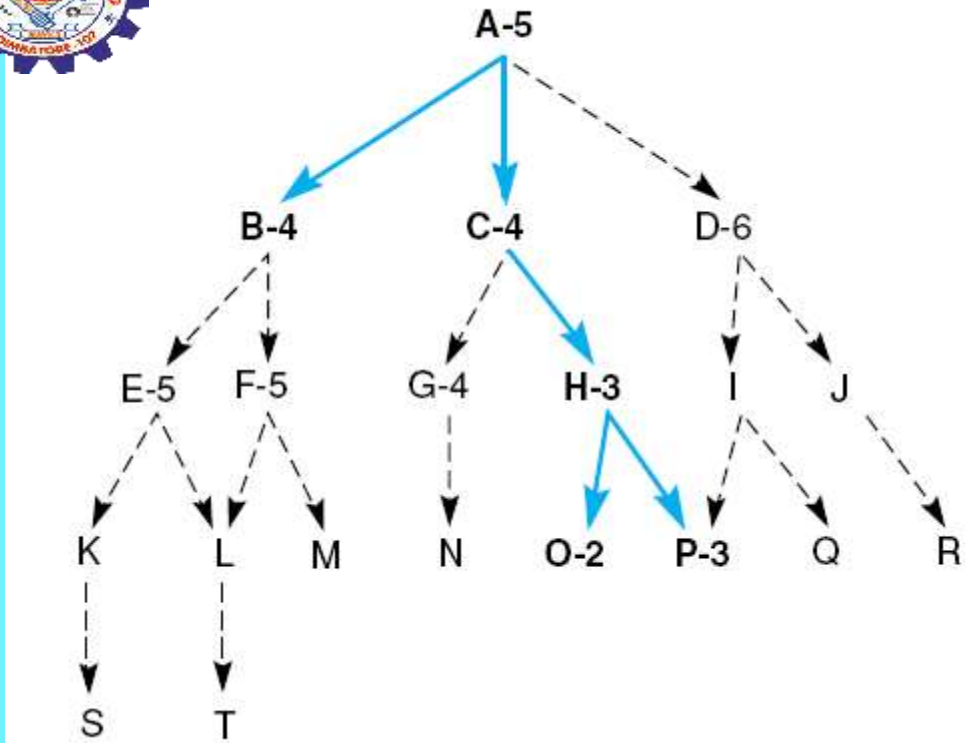


Now, our possible choices are I, J, H, E and F

Best-First Search Algorithm

```
function best_first_search;

begin
   open := [Start];                                                    % initialize
   closed := [ ];
   while open ≠ [ ] do                                                 % states remain
      begin
         remove the leftmost state from open, call it X;
         if X = goal then return the path from Start 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;
return FAIL                                                            % open is empty
end.
```



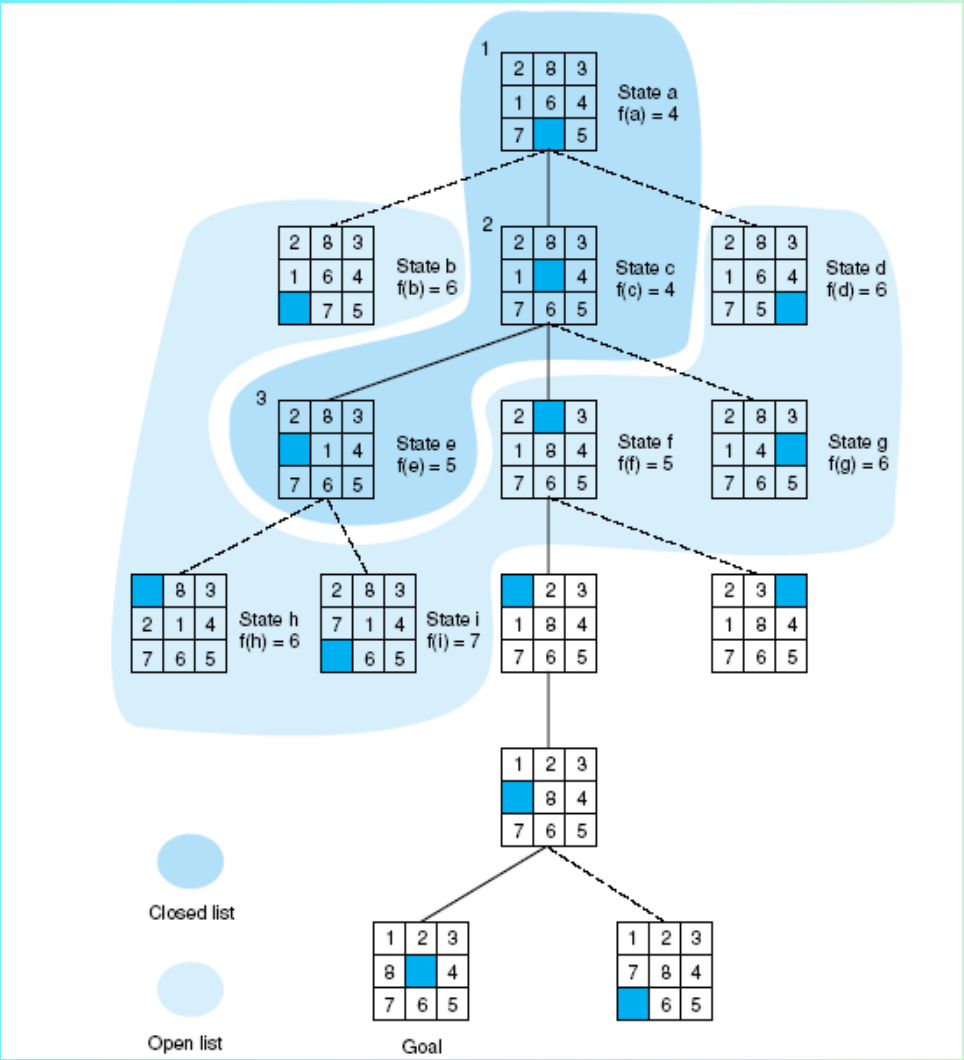
Best-first Search Example

1. open = [A5]; closed = [ ]
2. evaluate A5; open = [B4,C4,D6]; closed = [A5]
3. evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]
4. evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]
5. evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]
6. evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!

Heuristic Search and Cost

- Consider in any search problem there are several different considerations regarding how good a solution is
  - does it solve the problem adequately?
  - how much time does it take to find the solution (computational cost)?
  - how much effort does the solution take? (practical cost)
    - notice that the second and third considerations may be the same, but not always
- It will often be the case that we want to factor in the length of the path of our search as part of our selection strategy
  - we enhance our selection mechanism from finding the highest heuristic value to finding the best value f(n) = g(n) + h(n)
    - f(n) – cost of selecting state n
    - g(n) – cost of reaching state n from the start state
    - h(n) – heuristic value for state n
  - if we use this revised selection mechanism in our best-first search algorithm, it is called the the A Algorithm
- Since we want to minimize f(n), we will change our heuristic functions to give smaller values for better states
  - some of our previous functions gave higher scores for better states

Example: 8 Puzzle Redux



1. open = [a4];
   closed = [ ]
2. open = [c4, b6, d6];
   closed = [a4]
3. open = [e5, f5, b6, d6, g6];
   closed = [a4, c4]
4. open = [f5, h6, b6, d6, g6, l7];
   closed = [a4, c4, e5]
5. open = [j5, h6, b6, d6, g6, k7, l7];
   closed = [a4, c4, e5, f5]
6. open = [l5, h6, b6, d6, g6, k7, l7];
   closed = [a4, c4, e5, f5, j5]
7. open = [m5, h6, b6, d6, g6, n7, k7, l7];
   closed = [a4, c4, e5, f5, j5, l5]
8. success, m = goal!

Other Factors in Heuristic Search

- Admissibility
  - if the search algorithm is guaranteed to find a minimal path solution (if one exists) – that is, minimize practical cost, not search cost
    - a breadth-first search will find one
  - if our A Algorithm guarantees admissibility, it is known as an A* Algorithm with the selection formula f*(n) = g*(n) + h*(n) where g*(n) is the shortest path to reach n and h*(n) is the cost of finding a solution from n
    - h(n) is an estimated cost derived by a heuristic function, h*(n) may not be possible, it requires an oracle
- Informedness
  - a way to compare two or more heuristics – if one heuristic always gives you a more accurate prediction in the A* algorithm, then that heuristic is more informed
- Monotonicity – we will skip this
  - there are other search strategies covered in the notes accompanying this chapter

Constraint Satisfaction

- Many branches of a search space can be ruled out by applying constraints
- Constraint satisfaction is a form of best-first search where constraints are applied to eliminate branches
  - consider the Cryptorithmetic problem, we can rule out several possibilities for some of the letters
- After making a decision, propagate any new constraints that come into existence
  - constraint Satisfaction can also be applied to planning where a certain partial plan may exceed specified constraints and so can be eliminated

SEND
+ MORE
MONEY

M = 1 ➔
S = 8 or 9 ➔
O = 0 or 1 ➔
O = 0 …
N = E + 1 (since N != E)

Now we might try an exhaustive search from here