**Session 3: Best-First search in Graph representation of Problems**

1. **OBJECTIVES**

* To be able to understand Greedy Best-First and A\* search algorithms;
* To be able to implement Greedy Best-First and A\* search algorithms in Prolog and in Python.

1. **Demonstration of Useful Resources**
2. **Greedy Best-First Search**

Consider a problem instance given in the following graph.

80

25

26

0

55

45

36

35

34

22

28

31

47

42

32

30

27

17

20

|  |  |  |
| --- | --- | --- |
| Node | Neighbor | Distance |
| i | a | 35 |
| i | b | 45 |
| a | c | 20 |
| … | … | … |

|  |  |
| --- | --- |
| Node | h(Node) |
| i | 80 |
| a | 55 |
| b | 42 |
| … | … |

i - Initial state (source) g - Goal state (destination) h - heuristic function (straight line distance)

**Basic idea and Major steps of the algorithm:**

1. A node is selected for expansion based on an evaluation function, f(n), which is taken f(n) = h(n.
2. A Priority Queue (PQ), which contains nodes in ascending order of h-values, is maintained.
3. A Possible Path (PP) is maintained that contains nodes currently supposed to be in the solution.
4. A tree of visited nodes along with their children is also maintained which helps to update PQ and PP.
5. The process begins by placing the source node in the empty PQ, and initiating a tree by placing that node as its root.
6. The process terminates when the destination node is placed in the PQ, and consequently, selected for visit.
7. The 1st node from the PQ is selected repeatedly, and each time the tree, the PQ and the PP are updated:
8. The node in the tree is marked visited and its neighbors from the graph are added to the tree as its children, while no repeated node is allowed in the tree;
9. The node itself is deleted from the PQ, but its children are added to the PQ.
10. The PP is straightened up to the root from the selected node.

Sample representation of tree, PQ and PP in Prolog:

pq([node(b, 42), node(a, 55)]).

…

pp([i, b, e, g]).

t\_node(i, nil). t\_node(a, i).

t\_node(b, i). t\_node(d, b).

…

1. **A\* Search: Minimization of the total estimated solution cost**

**Distinguishing features:**

* Evaluation function,

f(n) = g(n) + h(n), where

g(n) = an actual path cost from initial node to node n,

h(n) = estimated cost of the cheapest path from n to the goal.

* Generates all neighbors (repeatedly, if a path is there), and puts into PQ.
* Suboptimal solutions are avoided.

Sample representation of tree, PQ and PP in Prolog:

pq([node(g, 17, 10, 97),

node(d, 4, 2, 98),

node(e, 5, 2, 101),

node(g, 13, 9, 104),

… ].

tr\_node(i, 0, nil, 80).

tr\_node(a, 1, 0, 90).

tr\_node(b, 2, 0, 87).

tr\_node(i, 3, 2, 170).

tr\_node(d, 4, 2, 98).

tr\_node(e, 5, 2, 101).

…

pp([i, a, d, g]).

1. **Lab Exercise**
2. Explore thoroughly the supplementary material provided for this session.
3. Run and analyze the codes demonstrated in this session.
4. Write a Python program that reads the file created as demonstrated into a dictionary taking ‘name’ as the key and a list consisting of ‘dept’ and ‘cgpa’ as the value for each line. Make changes in some ‘cgpa’ and then write back the whole file.
5. Implement in generic ways (as multi-modular and interactive systems) the Greedy Best-First and A\* search algorithms in Prolog and in Python.