Report

Name: Ashutosh Mehta
Student ID: 1001709115

Description Of Missionary and Cannibal Program

State space representation:

Each state has 5-tuples -
<missionary count$_{left}$, cannibal count$_{left}$, missionary count$_{boat}$, cannibal count$_{boat}$, boat position>

Algorithm:

1. Initialize start state and goal state.
2. Initialize open list which has nodes and are considered for expansion at every iteration.
3. Initialize visited list which stores the nodes that are already explored.
4. Push start state into open list and make it current state.
5. Calculate heuristic value and generate successors of current state. Pop start state from open list and push successors to open list which are not in visited list. Also push start state into visited list.
6. Calculate heuristic value for each successor and make the successor which has minimum heuristic value as current state. Pop the current state from open list and push it to visited list.
7. If the current state is goal state then return success and print the path from start state to goal state.
8. Go to step 5 if the open list is not empty, else return failure.

How to run:

Open clisp
> (load “mc.lisp”)
> (m_c_solver)

Output:

The output shows the sequence of states from start state to the goal state. For debugging purpose, loop count is also displayed.

Name: Ashutosh Mehta
Student ID: 1001709115

Description for 8-Puzzle Program

State space representation:
Each state is represented as <t1,t2,t3,t4,t5,t6,t7,t8,t9,empty_place> where empty_place represents the position of blank tile in the list.

Evaluation Function:

h(n)= misplaced tiles
g(n)= value from start state to current state
f(n)= g(n) + h(n)

Algorithm:

1. Initialize start state and goal state.
2. Initialize open list which has nodes and are considered for expansion at every iteration.
3. Initialize closed list which stores the nodes that are already explored.
4. Push start state into open list.
5. Pop the best element, state with minimum f(n) value, from the open list using evaluation function f(n)=h(n)+g(n) and set it as current state.
6. If the current state is goal state then return success and print the path from start state to goal state.
7. Generate successor of current state and push successors to open list which are not present in closed list and push current element into closed list.
8. Go to step 5 until the open list is not empty else return failure.

Known Deficiencies:

The given heuristic works fine for the easy configurations. For complex configuration, the program will take more time to reach the goal state as the program is unable in capturing progress of the search.

How to run:

Open clisp
> (load “N_puzzle.lisp”)
> (8p_solver)

To check feasibility

> (is_feasible *start_node* *goal_node*)

Output:

The output shows the sequence of states from start state to the goal state. It also print the total number of expanded states.