Robot MazeProblem Solution

Mahesh Devalla

September 29, 2016

1 Introduction

Mazeworld problem in Artificial Intelligence is classic problem where a robot needs to reach the goal node by navigating through a maze, which has a few obstacles in between. My approach to this problem is in three major portions as per asked in the assignment.

- Given a starting node implement A^* search algorithm for a particular maze, and later compare the results with BFS and DFS search for a single robot.
- Given a starting nodes of three robots implement A^* search algorithm for a particular maze, and later compare the results with BFS and DFS searches.
- Implement A^* algorithm where for a robot, which doesn't know it's presence(blind robot) and moves to its goal by trail and error basis.

2 A* Search for Mazeworld

The A^* search algorithm is an informed search technique that takes cost into account to reach to the node, $g(n)$ and the heuristic function $h(n)$ to calculate $f(n) = g(n) + h(n)$.

Data Structures: we have to use hashmaps and priority queues to keep a note of nodes that are visited and also the priorities.

we are comparing the different algorithms here in terms of number of nodes explored, which convey the runtime complexity of the algorithm, and the space complexity of the algorithm.The implementation is represented in Listing 1, the comparison is found in Table 1.

Implementaation

```
public List<SearchNode> aStar()
  {
    resetStats();
    PriorityQueue<FindPath.SearchNode> pq = new PriorityQueue<>();
    HashMap<FindPath.SearchNode,Double> prio = new HashMap<>();
    HashMap<FindPath.SearchNode, SearchNode> hm = new HashMap<>();
    SearchNode node;
    prio.put(startNode,startNode.priority());
    hm.put(startNode, null);
    pq.add(startNode);
    while(!pq.isEmpty())
    {
      node = pq.poll();// If queue is empty then dequeue the node
      //System.out.println("node...:"+node);
      if(node.goalTest())// check if node reached the goal node or not here(6,0)
      {
        return backchain(node,hm);
```

Search Technique		Maze.txt
	Nodes explored	Space
A*	15	51
DFS	19	19
BFS	35	43

Table 1: Comparison of the various algorithm for the maze given in website(2nd maze)

```
        }
        else if(prio.containsKey(node) && (node.priority() > prio.get(node)))
        {
          continue; // Continuing till the goal node is found
        }

        incrementNodeCount();// While goal not found then increment node count.
        updateMemory(pq.size() + prio.size() + hm.size());// updating existing memory for
      printing later
        for(SearchNode nextNode: node.getSuccessors())
        {
          if(!prio.containsKey(nextNode) || nextNode.priority() < prio.get(nextNode))
          {
            prio.put(nextNode, nextNode.priority());
           hm.put(nextNode, node);
            pq.add(nextNode);

          }
        }

      }
      return null;
    }
```

Listing 1: A star Search in Java

OUTPUT OF BFS:

```
The loaded maze is:

.......
.##....
..##...
....#..
..##...
..#....
....##.

Breadth First Search:
  Nodes explored during search:  35
  Maximum space usage during search 43

 The path followed by Robot is as follows:
(3, 3 ),(2, 3 ),(1, 3 ),(1, 2 ),(1, 1 ),(1, 0 ),(2, 0 ),(3, 0 ),(3, 1 ),(4, 1 ),(5, 1 ),(6,
    1 ),(6, 0 ),

please note that coordinates start with (0,0) at left bottom. (3,3) is start and (6,0) is
    goal
```

Listing 2: BFS in Java

OUTPUT OF DFS:

```
The loaded maze is:
.......
```

```
.##....
..##...
....#..
..##...
..#....
....##.
Depth First Search:
  Nodes explored during search:  19
  Maximum space usage during search 19

 The path followed by Robot is as follows:
(3, 3 ),(2, 3 ),(1, 3 ),(1, 4 ),(0, 4 ),(0, 5 ),(0, 6 ),(1, 6 ),(2, 6 ),(3, 6 ),(4, 6 ),(5,
    6 ),(6, 6 ),(6, 5 ),(6, 4 ),(6, 3 ),(6, 2 ),(6, 1 ),(6, 0 ),

please note that coordinates start with (0,0) at left bottom. (3,3) is start and (6,0) is
    goal
```

Listing 3: DFS in Java

OUTPUT OF ASTAR:

```
The loaded maze is:
.......
.##....
..##...
....#..
..##...
..#....
....##.


 AStar Search:
  Nodes explored during search:  15
  Maximum space usage during search 51

 The path followed by Robot is as follows:
(3, 3 ),(2, 3 ),(1, 3 ),(1, 2 ),(1, 1 ),(1, 0 ),(2, 0 ),(3, 0 ),(3, 1 ),(4, 1 ),(5, 1 ),(6,
    1 ),(6, 0 ),
please note that coordinates start with (0,0) at left bottom. (3,3) is start and (6,0) is
    goal
```

Listing 4: A star Search in Java

3 Multi-Robot Mazeworld

In multi-robot-mazeworld n robots in the maze are to be reached to the goal.

Below are the few conditions for multi- robot maze

1. if we have n robots, then we represent that the state of the robot with co-ordinates (x,y) for each robot, and we should also consider the fact that only one robot move at an instance. so combining this we can tell that $2n + 1$ values required to show the state.
2. If there are k robots, and nXn maze, the total number of states can be calculated as ${}^{n^2}P_k$ and we pick k states (k robots, k turns) from n^2 possible turns, and in a particular order.
3. If $n >> k$ and w represents the number of walls, the number of states respectively without having any collisions could be given by ${}^{n^2-w}P_k$. For this to calculate the number of states with collisions, we would have to calculate 23.
4. Without there being many walls, a straightforward BFS would be hard, since the branching factor would be 10, and the number of states at each level would increase exponentially.

5. Out of many heuristic functions for the multi robot problem the sum of all the manhattan distances would be an good option. This can produce shortest distance if there are no obstacles from the start to the end of every robot.

3.1 Implementation

The states of the robots can be gievn in array of $2n + 1$ elements, where n in the number of robots. So the state space would look something like this - $[x_0, y_0, x_1, y_1, ..., x_{n-1}, y_{n-1}, t]$.

```java

    public ArrayList<SearchNode> getSuccessors() {

            ArrayList<SearchNode> listOfNodes = new ArrayList<>();
            SearchNode node;
            for (int[] temp1: move)
            {
                int[] temp2  = dupCurrRobotLoc(currRobotLoc);
                int value = currRobotLoc[currRobotLoc.length -1];
                temp2[2*value] = currRobotLoc[2*value] + temp1[0];
                temp2[2*value+1] = currRobotLoc[2*value+1] + temp1[1];
                temp2[currRobotLoc.length -1] = (value+1)%numOfRobots;
                if(loadedMaze.isLegal(temp2[2*value], temp2[2*value+1]) && !collide(temp2,
    temp2[2*value], temp2[2*value+1]))
                {
                  node = new MutliRobotNode(temp2, getCost() + 1.0);
                    listOfNodes.add(node);
                }
            }
            return listOfNodes;
        }
```

Listing 5: Multi Robots movement in Mazeworld

OUTPUT OF BFS:

```
The loaded maze is:

.......
.##....
..##...
....#..
..##...
..#....
....##.



 Breadth First Search:
  Nodes explored during search:  63967
  Maximum space usage during search 77928

 The path followed by Robot is as follows:
[ Robot A moves from:(0,0)(1,6)(4,5),  Robot B moves from:(0,1)(1,6)(4,5),  Robot C moves
    from:(0,1)(2,6)(4,5),  Robot A moves from:(0,1)(2,6)(5,5),  Robot B moves from:(1,1)
    (2,6)(5,5),  Robot C moves from:(1,1)(3,6)(5,5),  Robot A moves from:(1,1)(3,6)(6,5),
    Robot B moves from:(1,0)(3,6)(6,5),  Robot C moves from:(1,0)(4,6)(6,5),  Robot A moves
    from:(1,0)(4,6)(6,4),  Robot B moves from:(2,0)(4,6)(6,4),  Robot C moves from:(2,0)
    (5,6)(6,4),  Robot A moves from:(2,0)(5,6)(6,3),  Robot B moves from:(3,0)(5,6)(6,3),
    Robot C moves from:(3,0)(6,6)(6,3),  Robot A moves from:(3,0)(6,6)(6,2),  Robot B moves
    from:(3,1)(6,6)(6,2),  Robot C moves from:(3,1)(6,5)(6,2),  Robot A moves from:(3,1)
    (6,5)(6,1),  Robot B moves from:(4,1)(6,5)(6,1),  Robot C moves from:(4,1)(6,4)(6,1),
    Robot A moves from:(4,1)(6,4)(6,0),  Robot B moves from:(5,1)(6,4)(6,0),  Robot C moves
    from:(5,1)(6,3)(6,0),  Robot A moves from:(5,1)(6,3)(6,0),  Robot B moves from:(6,1)
    (6,3)(6,0),  Robot C moves from:(6,1)(6,2)(6,0)]
```

```

please note that each state start from and moves to next node for example
Robot A moves from means it is currently in that position
(First coordinate represent "A"Robot, Second coordinate represent "B" Robot,Third coordinate
     represent "C"Robot)

Another example: Robot A moves from:(0,0)(1,6)(4,5),  Robot B moves from:(0,1)(1,6)(4,5),
Here you can see change in only first co-ordinate from (0,0) to (0,1) which signifies that
    Robot "A" moved and other two didn't move.

Starting state of (A,B,C) robots=(0,0)(1,6)(4,5)
Goal state of (A,B,C) robots=(6,1)(6,2)(6,0)

This is given as(also given in assignment website) :

start
. B . . . . .
. # # . C . .
. . # # . . .
. . . . # . .
. . # # . . .
. . # . . . .
A . . . # # .

Goals state:
. . . . . . .
. # # . . . .
. . # # . . .
. . . . # . .
. . # # . . B
. . # . . . A
. . . . # # C
```

Listing 6: BFS in MultiRobot maze

OUTPUT OF DFS:

```
The loaded maze is:
.......
.##....
..##...
....#..
..##...
..#....
....##.


 Depth First Search:
  Nodes explored during search:  1553915
  Maximum space usage during search 51

 The path followed by Robot is as follows:
[ Robot A moves from:(0,0)(1,6)(4,5),  Robot B moves from:(0,1)(1,6)(4,5),  Robot C moves
    from:(0,1)(2,6)(4,5),  Robot A moves from:(0,1)(2,6)(4,6),  Robot B moves from:(0,2)
    (2,6)(4,6),  Robot C moves from:(0,2)(3,6)(4,6),  Robot A moves from:(0,2)(3,6)(5,6),
    Robot B moves from:(0,3)(3,6)(5,6),  Robot C moves from:(0,3)(4,6)(5,6),  Robot A moves
    from:(0,3)(4,6)(6,6),  Robot B moves from:(0,4)(4,6)(6,6),  Robot C moves from:(0,4)
    (5,6)(6,6),  Robot A moves from:(0,4)(5,6)(6,5),  Robot B moves from:(0,5)(5,6)(6,5),
    Robot C moves from:(0,5)(6,6)(6,5),  Robot A moves from:(0,5)(6,6)(6,4),  Robot B moves
    from:(0,6)(6,6)(6,4),  Robot C moves from:(0,6)(6,5)(6,4),  Robot A moves from:(0,6)
    (6,5)(6,3),  Robot B moves from:(1,6)(6,5)(6,3),  Robot C moves from:(1,6)(6,6)(6,3),
    Robot A moves from:(1,6)(6,6)(6,4),  Robot B moves from:(2,6)(6,6)(6,4),  Robot C moves
    from:(2,6)(6,5)(6,4),  Robot A moves from:(2,6)(6,5)(6,3),  Robot B moves from:(3,6)
    (6,5)(6,3),  Robot C moves from:(3,6)(6,6)(6,3),  Robot A moves from:(3,6)(6,6)(6,4),
    Robot B moves from:(4,6)(6,6)(6,4),  Robot C moves from:(4,6)(6,5)(6,4),  Robot A moves
```

Search Technique		Maze.txt
	Nodes explored	Space
A*	793	4667
DFS	1553915	51
BFS	63967	77928

Table 2: Comparison of the various algorithm for the maze given in website(2nd maze) for mutli robot

```
    from:(4,6)(6,5)(6,3),  Robot B moves from:(5,6)(6,5)(6,3),  Robot C moves from:(5,6)
    (6,6)(6,3),  Robot A moves from:(5,6)(6,6)(6,4),  Robot B moves from:(5,5)(6,6)(6,4),
    Robot C moves from:(5,5)(6,5)(6,4),  Robot A moves from:(5,5)(6,5)(6,3),  Robot B moves
    from:(5,4)(6,5)(6,3),  Robot C moves from:(5,4)(6,6)(6,3),  Robot A moves from:(5,4)
    (6,6)(6,2),  Robot B moves from:(6,4)(6,6)(6,2),  Robot C moves from:(6,4)(6,5)(6,2),
    Robot A moves from:(6,4)(6,5)(6,1),  Robot B moves from:(6,3)(6,5)(6,1),  Robot C moves
    from:(6,3)(6,4)(6,1),  Robot A moves from:(6,3)(6,4)(6,0),  Robot B moves from:(6,2)
    (6,4)(6,0),  Robot C moves from:(6,2)(6,3)(6,0),  Robot A moves from:(6,2)(6,3)(6,0),
    Robot B moves from:(6,1)(6,3)(6,0),  Robot C moves from:(6,1)(6,2)(6,0)]
```

Listing 7: DFSr in MultiRobot

OUTPUT OF ASTAR:

```
The loaded maze is:
.......
.##....
..##...
....#..
..##...
..#....
....##.


 AStar Search:
  Nodes explored during search:  793
  Maximum space usage during search 4667

 The path followed by Robot is as follows:
[ Robot A moves from:(0,0)(1,6)(4,5),  Robot B moves from:(1,0)(1,6)(4,5),  Robot C moves
    from:(1,0)(2,6)(4,5),  Robot A moves from:(1,0)(2,6)(5,5),  Robot B moves from:(2,0)
    (2,6)(5,5),  Robot C moves from:(2,0)(3,6)(5,5),  Robot A moves from:(2,0)(3,6)(6,5),
    Robot B moves from:(3,0)(3,6)(6,5),  Robot C moves from:(3,0)(4,6)(6,5),  Robot A moves
    from:(3,0)(4,6)(6,4),  Robot B moves from:(3,1)(4,6)(6,4),  Robot C moves from:(3,1)
    (5,6)(6,4),  Robot A moves from:(3,1)(5,6)(6,3),  Robot B moves from:(4,1)(5,6)(6,3),
    Robot C moves from:(4,1)(5,5)(6,3),  Robot A moves from:(4,1)(5,5)(6,2),  Robot B moves
    from:(5,1)(5,5)(6,2),  Robot C moves from:(5,1)(5,4)(6,2),  Robot A moves from:(5,1)
    (5,4)(6,1),  Robot B moves from:(5,1)(5,4)(6,1),  Robot C moves from:(5,1)(5,3)(6,1),
    Robot A moves from:(5,1)(5,3)(6,0),  Robot B moves from:(6,1)(5,3)(6,0),  Robot C moves
    from:(6,1)(5,2)(6,0),  Robot A moves from:(6,1)(5,2)(6,0),  Robot B moves from:(6,1)
    (5,2)(6,0),  Robot C moves from:(6,1)(6,2)(6,0)]
```

Listing 8: A star Search in MultiRobot

4 Blind Robot in a Mazeworld

In this case the robot is not aware of its location in the maze, the ultimate goal of the robot is to do trail and errors to find out where it is exactly in the particular maze. The heuristic function if number of states can be k in a $n * n$ maze is given by n/k, which eliminates k columns or rows based upon the direction.

Note: Due to the a small mishap in the code of producing path for dfs with bigger depth my project ended abruptly., so providng the pseduo code as of now, due to the time constraints.

Search Technique		Maze.txt
	Nodes explored	Space
A*	A big number	A big number
DFS	Hanged	Hanged
BFS	A very big number	A very big number

Table 3: Comparison of the various algorithm for the maze given in website(2nd maze)

```
        public ArrayList<SearchNode> getSuccessors()
        {
            ArrayList<SearchNode> successors = new ArrayList<SearchNode>();
            {
                if(maze.isLegal(xNew, yNew)) {

                    /* Calculate all the states by storing either 1/0 in their respective
    states
                    and copy this array into another array for future use
                     if(moving east)
                     {
                     calculate valid node or not by calling isLegal(int x, int y) method and
     reduce the node count accordingly.
                     return the node for further movements
                     }
                        Repeat the same steps for all the directions
            return successors;
        }
```

Listing 9: Blind Robot Successor Method

The output of A* search in will be as follows.

```
A*:
E,W,N,E,W,N....
```

5 literature review

Standley.T standley2010finding: In this paper the author highlights the new ideas to further increase the cooperative path finding problem. Additionally, he also proposes a method known as Operator Decomposition, a technique that is used to increase single agents at a time than to move every other agent into another state, which reduces the computing. This is one of the good ways to solve the problem according to the author.

The algorithm in this paper decreases the branching factor but the overhead is: it runs in an exponential time. So, he introduces "Simple Independence Detection" algorithm. In this the path is calculated thinking that it is independent and has no other connections with other paths. We use backtracking technique if an obstacle or conflict is found.

He also quantifies that using the Independence Detection algorithm and the Operator Decomposition algorithms solved 90.45 % of problems within no time. When we use the regular A^* Search and Independence Detection have resulted in 74.98 output. So, he states that these new algorithms can be applies in many domains to solve the classic path finding related problems with much faster computing time.