Project GridWorld
Final Report

Stavros Orfanoudakis 2015030030
Andreas Kallinteris 2017030066

March 31, 2019

1 Introduction

In the following project we are going to study some search algorithms in a 2-D grid-world like space. Specifically, our agent has to reach it's destination considering two kind of walkable tiles (grass and land) and one unwalkable obstacle tile (wall). To do that we had to implement BFS, DFS , A* and LRTA* algorithms in order to test which one behaves better.

2 Implementation

In this section we are going to explain the implementation of each separate algorithm.

2.1 DFS

DFS algorithm has been implemented as a stack , which means that our agent picks one non-discovered neighbouring node and expand ,while at the same time it puts the rest neighbouring nodes within the stack.This is repeated until the agent reach the destination. In particular, when expanding nodes, we have to make sure that the nodes ,at first are within the borders of the grid and secondly that the node is not a wall.

2.2 BFS

BFS algorithm adds all it's neighbouring non-discovered nodes into a queue and always expands the first in priority node .BFS keeps expanding considering the previous constraints, until it reaches the final destination.

2.3 A*

A* search algorithm has been implemented as following. At first we initialize an nxn "discovered" table so we can store the cost estimation of each node .Later the agent puts every neighbouring node ,that is not discovered yet or has a cost estimate lower than the previous one, in a buffer where each node has index and weight.After that, it checks every node in the buffer and returns the one with the lowest weight so the agent can move there. These actions are going to be repeated until the agent reaches the destination. When the agent reaches the destination the "discovered" table would be filled with useful values that are going to provide us (through backtracking) the optimal path.
Also it is worth mentioning ,how the cost of every node is estimated. Each weight comes from this function $f(n) = g(n) + h(n)$ where $g(n)$ is the cost from the start to the n node and $h(n)$ is the heuristic function which estimates

the distance to the destination. Regarding the heuristic function ,we tested a variety of functions and we ended up at the point where we found that the Manhattan distance is better most of the times, like bibliography suggested .

2.4 LRTA*

Learning real time A* is a local search algorithm. This algorithm is special ,because the weight of the node is determined after the agent walk there himself. This makes the problem a bit harder to solve. In our implementation we used the following LRTA* code.

function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table, indexed by state and action, initially empty
    H, a table of cost estimates indexed by state, initially empty
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state (not in H) **then** $H[s'] \leftarrow h(s')$
  if s is not null
    $result[s, a] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*-COST($s, b, result[s, b], H$)
  $a \leftarrow$ an action b in ACTIONS(s') that minimizes LRTA*-COST($s', b, result[s', b], H$)
  $s \leftarrow s'$
  return a

function LRTA*-COST(s, a, s', H) **returns** a cost estimate
  if s' is undefined **then return** $h(s)$
  else return $c(s, a, s') + H[s']$

It is worth saying ,that when there are two or more actions that minimize the LRTA* cost, we choosed the node which where closer to the destination using the same heuristic function as A* (Manhattan distance).

3 Evaluation - Results

In the following section we are going to present the results , after running each algorithm in 5 given worlds with grass cost 2 and 5.

3.1 BFS

Below we can see the **path steps** it took BFS algorithm to reach destination.

BFS	**easy**	**default**	**hard_a**	**hard_b**	**hard_c**
grass_cost : 2, 5	55	36	87	86	123

3.2 DFS

Below we can see the **path steps** it took DFS algorithm to reach destination.

DFS	easy	default	hard_a	hard_b	hard_c
grass_cost : 2, 5	48	71	118	112	198

3.3 A*

Below we can see the cost of the optimal path and the number of steps it requires, using A*.

*A**	easy	default	hard_a	hard_b	hard_c
grass_cost :2	11 (11 steps)	15 (8 steps)	20 (17 steps)	19 (16 steps)	24 (20 steps)
grass_cost : 5	11 (11 steps)	24(20 steps)	23 (23 steps)	28 (16 steps)	32 (28 steps)

3.4 LRTA*

Below we can see the cost of the path and the number of steps it requires, using LRTA*.

*LRTA**	easy	default	hard_a	hard_b	hard_c
grass_cost :2	14 (11 steps)	15 (8 steps)	36 (29 steps)	21 (16 steps)	48 (42 steps)

3.5 Results

We can see that DFS and BFS algorithms are the worst considering the path cost , but this is natural ,because these algorithms doesn't consider the weight of the nodes they process. On the other hand A* is optimal and we can prove that through the previous tests. That means that there is no better path than the one discovered by A* agent. We can also observe ,that the learning real time algorithm doesn't find the overall optimal path like A* , but it can find a very close to optimal path depending on the complexity and the size of the world it searches.

Code notes

Our code is implemented in C++ . We use the implemented worlds text files so we can load the worlds to this project. Also ,we used make-file to execute the project through terminal (command $make cr when in the project folder).Boost C++ library is required for our BFS to run (we use an implemented cyclic buffer to save time).