HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

------ 📖 ------

PROJECT REPORT

AI gardening problem

Class: Introduction to Artificial Intelligence – 128519

GROUP 3

1. Đỗ Tuấn Minh – 20200390
2. Hoàng Huy Chiến – 20200084
3. Đinh Ngọc Hạnh Trang – 20204928
4. Nguyễn Kim Tuyến – 20205196
5. Tô Thái Dương – 20205180

INTRODUCTION

In the course Introduction to AI, we have studied about several kinds of Intelligent Agents such as simple reflex agents, model-based reflex agents, goal-based agents, utility-based agents, knowledge-based agents. Among them, utility-based agents, consider search cost and the path cost in the progress of trying to reach the goal. This agent is the topic we choose for our programming project.

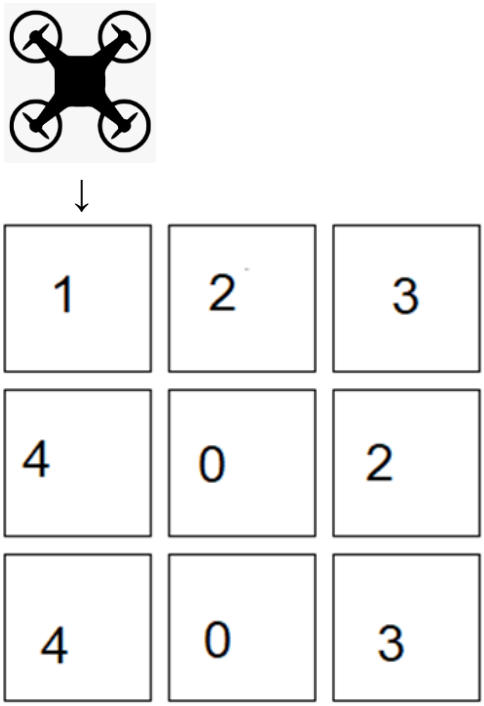
In this report, we present the AI gardening problem and propose some methods for solving this problem. We will also concentrate on explain in details how each algorithm works with some helpful analyses.

Contents

1. Presentation of the subject

There is a mxn square garden. We denote (x,y) is the location of one square. Each square contains exactly one of the following:

- Scarecrow (Denote 0)
- Warehouse (Only at (0,0)) (Denote 1)
- Newly planted seed (Denote 2)
- Live tree (Denote 3)
- Dead Tree (Denote 4)



An AI machine, starts at the warehouse ((0,0)), has following functionalities:

- If AI machine gets to the newly planted seed, it will water it.
- If AI machine gets to the live tree, it will pick all the fruit and bring it back to the warehouse.
- If AI machine gets to the dead tree, it will remove the dead tree, bring it back to the warehouse, then take a new seed to the previous spot, plant and water it (plant and water at the same time).
- The AI machine can not go pass the scarecrow.

The AI machine can perform the following actions:

- Move forward.
- Turn left.
- Turn right.

- ❖ Remove dead tree.
- ❖ Put the dead tree back to the ware house.
- ❖ Water the seed (plant and water will be count as one action).
- ❖ Pick the fruit.
- ❖ Bring the fruit back to the warehouse.

The goal is to:

- ❖ Water all the newly planted seed.
- ❖ Bring all the fruits back to the warehouse.
- ❖ Bring all the dead trees back to the warehouse.
- ❖ Plant and water all the new tree in the dead tree spot.

With a minimum number of actions:

- ❖ If not possible, print out “Can not find the path”.

2. Description of the problem

2.1 Detailed description of the problem

- ❖ The environment is known, partially observable, deterministic, dynamic and discrete. Thus, the solution is always a fixed sequence of actions.
- ❖ There are two main parts in this problem to deal with:
    - Searching: to find the shortest path in the matrix (mxn square garden).
    - Traversing: to visit and do the task in every cell (except the scarecrow).

=> We have to combine at least two algorithms.

- ❖ As the enviroment is partially observable, so the state is changing every time the AI machine explore new cells which have not been explored yet, so we will denote -1 to the cell which the AI machine has not explored yet.

2.2 Problem formulation

- ❖ States: m*n matrix with value -1, 0, 1, 2, 3, 4 (number 1, which is warehouse, can only be at (0,0)).
- ❖ Initial state: m*n matrix with value -1 (except at (0,0) it will be 1).
- ❖ Actions: forward, turn left, turn right, water (if only it goes to 2), pick the fruit (if only it goes to 3), bring all the fruit back to the warehouse (if only it goes to 3), remove dead tree (if only it goes to 4), bring dead

tree back to the warehouse (if only it goes to 4), plant and water (if only it goes to 4)

- ❖ Transition model: Expected effects, except:
  - Going to cell 0 (scarecrow).
  - Moving out of the matrix m*n given.
  - Doing the actions which is other cell type's task (For example, you can not pick the fruit when you visit a newly planted seed).
- ❖ Goal test: This checks whether:
  - All the newly planted seeds are watered.
  - All the fruits, dead trees are brought to the warehouse.
  - Plant and water every new seed which is brought to the dead tree's removal spot.
- ❖ Path cost: each action cost 1, so the path cost is number of actions taken .

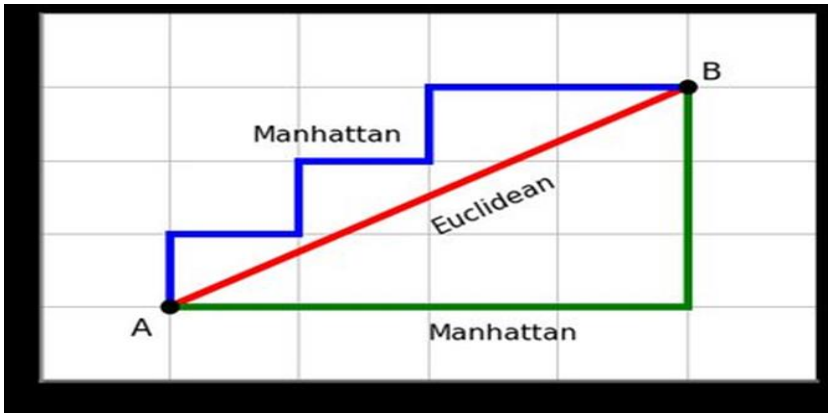
3. Algorithm selection

For searching the shortest path, we apply two algorithms: A* searching algorithm and Breadth first search algorithm (BFS).

For traversing, we apply Depth first search algorithm.

The reasons for choosing these algorithms:

- BFS is one of the best algorithms to find the shortest path in matrix problem. To find the shortest path, all you have to do is start from the source and perform a BFS and stop when you found your destination. The only additional thing you need to do is have an array previous[] which will store the previous node for every node visited.
- A* searching algorithm is also one of the best algorithms to find the shortest path in matrix problem. This algorithm is complete and optimal. Besides, the heuristic function in this problem is quite clear , which is the Manhattan distance (the sum of the absolute differences between the two vectors).



- Depth first search algorithm (DFS) consumes less memory and is more efficient than breadth first search (BFS) when it comes to traversing through the matrix. This problem does not cause any loop so DFS is enough to perform the solution. DFS visits every single nodes separately so we can propose any action at that node easier.
- Combining A* search(or BFS) algorithm with DFS algorithm, we can find the shortest path and traverse through every single cell in the matrix.

3.1 Algorithm for searching:

3.1.1 A* Search Algorithm

We apply the A* search into this problem:

- ❖ Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. So here we use A* Search Algorithm.
- ❖ The heuristic function in this algorithm is the Manhattan distance (the sum of the absolute

Input: A graph *G(V,E)* with source node *start* and goal node *end*.
Output: Least cost path from *start* to *end*.
Steps:

```
Initialise
    open_list = { start }                    /* Listof nodes to be traversed*/
    closed_list = { }                        /*List of already traversed nodes*/
    g(start) = 0                             /*Cost from source node  to a node*/
    h(start) = heuristic_function (start,end)  /*Estimated cost from node to goal node*/
    f(start) = g(start)+h(start)             /* Total cost from source to goal node*/

while open_list is not empty
    m = Node on top of open_list, with least f
    if m = = end
        return
    remove m from open _list
    add m to closed_list
    for each n in  child(m)
        if n in closed_list
            continue
        cost= g(m)+ distance(m,n)
        if n in open_list and cost < g (n)
            remove n from open_list as new path is better
        if n in closed_list and cost < g(n)
            remove n from closed_list
        if n not in open_list and n not in closed_list
            add n to open_list
            g(n) = cost
            h(n) = heuristic_function (n,end)
            f(n) = g(n)+h(n)
return failure
```

differences between the two vectors) of the point considered.(For example, the Manhattan distance between (x1,y1) and (x2,y2) is |x1 - x2| + |y1 - y2|)

- ❖ The reason we use Manhattan distance:
  - It is admissible.
  - Since we can only move the blocks 1 at a time and in only one of 4 directions (right, left, top, bottom), the optimal scenario for each block is that it has a clear, unobstructed path to its goal state.

3.1.2 Breadth First Search(BFS)

We apply the BFS into this problem:

- Consider a square grid having many obstacles and we are given a starting cell and a target cell. There are many ways to traverse graphs. BFS is the most commonly used approach, so we apply this algorithm.
- Let's first start out with search - and search for a target node. Besides the target node, we'll need a start node as well. The expected output is a path that leads us from the start node to the target node.
- When we're reconstructing the path (if it is found), we're going backwards from the target node, through it's parents, retracing all the way to the start node. Additionally, we might want to reverse the path for our own intuition of going from the start_node towards the target_node.

```
BFS (G, s, e)
//Where G is the graph,s is the source node, e is the target node
let Q be queue.
Q.enqueue( s )
//Inserting s in queue until all its neighbour vertices are marked.
mark s as visited.

let P be dict
//P to store parents of node
let path be queue
//path to store the way
while ( Q is not empty)
    s = Q.dequeue()
    if s is target node:
        path_found = true
        break
    //Found the target node
    for all neighbours w of v in Graph G
        if w is not visited
        Q.enqueue( w )
        //Stores w in Q to further visit its neighbour
        mark w as visited.
        P = current_node
if path_found:
    path.append(target_node)
    while P is not none:
        path.append(P([target_node[0], target_node[1]))
        target_node = P([target_node[0], target_node[1])
    //Traversing
    path.reverse
return path
```

3.2 Algorithm for traversing :Depth First Search Algorithm (DFS)

We also apply depth first search algorithm in our problem:

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.
- As the environment is partially observable, which means the AI gardening can only observe its 4 neighbours (the one which shares a side to its current cell) so every time we DFS a cell, we can store its value (0, 1, 2, 3, 4) inside the matrix_discover (the matrix that the AI machine has seen so far).

```
DFS(s){
    //Update the matrix the AI machine sees
    If visited(s)= true:
        return
    visited(s)= true
    If s is obstacle:
        return
    //logic
    //If dfs 2 vertexes not adjacent to each other continuosly then find the shortest path between them, update the path and action needed
    //Process type of the vertex, do the task required, update the path and action needed
    // Print the percept sequence and action at each step
    for v in neighbor(s) do
        If visited(v)= false then{
            DFS(v)
        }
}

for every v in graph do
    visited(s)= false
for every v in graph do
    DFS(v)
```

4. Algorithms implementation in the propose problem

4.1 Main difficulty:

During the work, we have faced some difficulties:

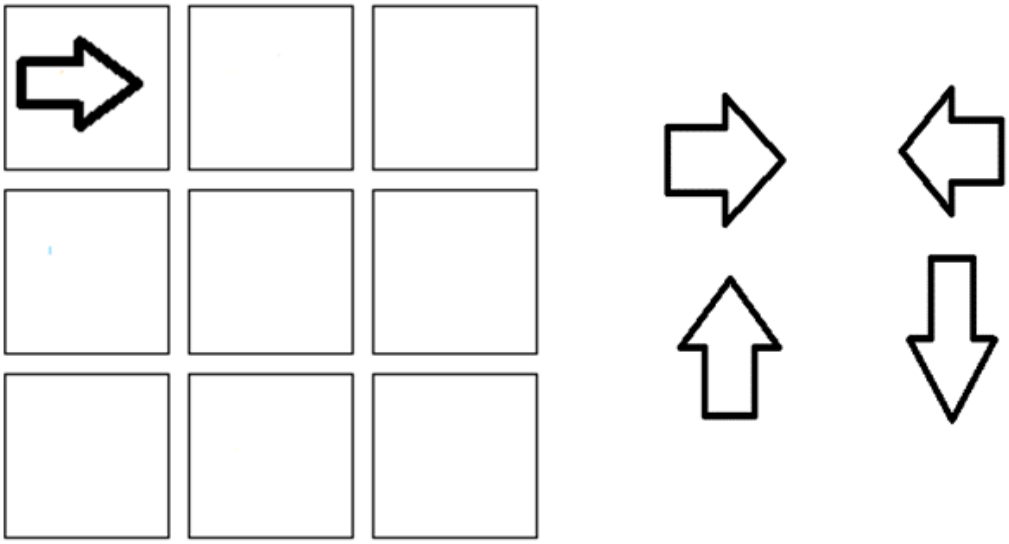
- For the BFS, as the matrix we apply this algorithm is changing every time the AI machine visits a new cell. That means we have to build a function to update the graph again every time, which is time consuming.
- For the A* Search algorithm, as the matrix we apply this algorithm is changing every time the AI machine visits a new cell. That means it has to calculate the heuristics again every time, which is time consuming.
- For DFS algorithm, there are different types of cells (live tree, dead tree, newly planted seed, scarecrow, warehouse) so we have to consider each case with a different approach. Each case considers

different kinds of actions (for example when we reach a newly planted seed, the action we need to do is ‘plant’). As there are some cells in the matrix that we can not go pass (scarecrow) so we can not present the matrix as one tree. So when we apply DFS algorithm, there is a chance that there is two cells which the AI gardening machine goes to continuously but they are not next to each other (due to the dfs code implementation).That is when we need to use A* search algorithm to find the path between these two cells. (Because the AI gardening machine can only go to its adjacent cell (the cell which shares one side) in each step).

- As the AI gardening machine can go forward, turn left, turn right, at each step, when we want the machine to move to the next cell, we need to consider more actions than usual. For example, when the machine is at (0,0), and heads right (towards (0,1)), if we want to move to (1,0), we have to do the list of actions: ‘Turn right’, ‘Forward’. We have to consider its previous actions before making any new actions.
- The speed execution of A* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute h (n). So we believe it has some complexity problems. So we need to make some alters to make it more efficient.

4.2 Propose a solution to deal with the difficulties

- We use function updategraph() to update the graph when the AI machine go to new cell. Because when AI machine go to new cell, the graph which we will use to apply BFS will change.
- We use Manhattan distance as the heuristic function. Because this heuristic is one of the most effective heuristics when it comes to finding shortest path in the matrix.
- Next, we have to handle the movement action (forward, turn left, turn right) when the AI gardening machine moves to a different cell. As we mentioned above , we need to consider the previous step to make any neext move. So we add a list State which stores the direction(left,right,up,down) the AI gardening machine heads at any certain time in the program. Then, we initialize the first state to be ‘Right’ (at (0,0) and head towards (1,1)).



Pic : First init and four types of states

So now we know which way the AI gardening machine heads, it is easy to find the movement action (see function movementActionAppend). After we move to a new cell, update the state of the AI gardening machine.

Every time we apply DFS to a cell in a matrix , we need to see which kind of cell it is (scarecrow,newly planted seed, live tree, dead tree) to do its following task.We use the function taskActionAppend to fully complete this action.

4.3 Algorithm improvement by the solution

- ❖ A* Search Algorithm
  - It costs much less time.
- ❖ DFS Algorithm
  - With two simple functions we use above, we can now see all the actions the machine needs to take to finish this problem.

5. Results comparison of the algorithm used for solving the problem

5.1Providing quantitative performance indicators

- ❖ Total number of test cases: 100.
- ❖ Number of test cases for which the algorithms can successfully find a path without breaking: 57.
- ❖ Percentage of problem completion for each algorithm (including both problems which do not exist any solution)

- A* search + DFS: 100%.
- DFS + BFS: 100%.

❖ Time and space complexities of each test cases observed in practice given in this table:
Data experiment.xlsx

5.2 Explaining these results

❖ If any of our algorithms completes searching (including cases where it successfully touches every cell and cases where it is stuck by obstacles), it will return the time taken and the number of steps taken in cases it successfully touches every cell or "Can not find the path" in cases where it is stuck by obstacles.

❖ For 2 ways of solving this problem, we both use DFS to traverse from one node to other, the only difference came from choosing the searching algorithm: A* search or BFS. In general, for the small size test cases, the execution time between these two do not show any noticeable differences. But for some larger size problems, the BFS's time for searching might be over 3 times slower than A* search. This is reasonable since the A* search is more well developed, so it should be more effective. However, the total steps of 2 algorithms are quite similar for most cases.

❖ A* search and BFS have the same space complexity in theory, which is O(b^d) as they store all generated nodes in the memory.

Criterion	**A* search**	**BFS**	**DFS**
Complete?	Yes	Yes	No
Time Complexity	(Depends on the heuristic)	O(b^d)	O(b^m)
Space Complexity	O(b^d)	O(b^d)	O(bm)
Optimal?	No	Yes	No

6. Conclusion and further discussions

- In conclusion, we believe that we manage to solve this problem quite well, with not so much time given. We have created a simple AI gardening machine which can have a very good application to life. However, there is still much work to do left to make this more efficient. Because as you could see, both of the algorithms are really time consuming.
- During the project, we have learnt a lot of things: A* search algorithms, BFS algorithms, DFS algorithms. We have learnt how these algorithms work, how to apply, combine them, how to choose the right algorithms to deal with the problem, how to analyse data, time and space complexity. This is the first time our team have had a team-work project, which we believe is very essential for our future. Thanks to the team, we have finished this project.
- Further discussions: In the future, we hope that we can continue to develop this project to make it more efficient (For example, put multiple warehouses in the matrix and the AI machine can visit the nearest warehouse when in need). We hope that one day we can make it into a real machine that can help people.

7. List of tasks

7.1 Programming tasks:

- Implementing BFS Algorithms: Hoàng Huy Chiến (75%), Nguyễn Kim Tuyến (25%).
- Implementing A* Search Algorithms: Đỗ Tuấn Minh.
- Implementing DFS Algorithms: Đỗ Tuấn Minh.
- Generating random cases for the problem : Tô Thái Dương (50%), Đinh Ngọc Hạnh Trang (50%).

7.2 Analytic tasks:

- Nguyễn Kim Tuyến : proposing our subject, write about BFS in the report, standardization of reports.
- Hoàng Huy Chiến : proposing our subject, making PDF to presentation.
- Đỗ Tuấn Minh: proposing our subject, choosing algorithms, idea to solve this problem, write 1,2,3.1.2,3.2,4,6,7,8 (except every thing about BFS) in the report
- Tô Thái Dương: proposing our subject, generating data sets, analyzing experiments results, making demo video.

- Đinh Ngọc Hạnh Trang: proposing our subject, generating data sets, analyzing experiments result, write 5 in the report.

8. List of bibliographic reference

- *A* Search: Concept, Algorithm, Implementation, Advantages, Disadvantages,* Source: https://www.brainkart.com/article/A--Search--Concept,-Algorithm,-Implementation,-Advantages,-Disadvantages_8883/

- *Depth First Search,* Source: https://en.wikipedia.org/wiki/Depth-first_search