Search Algorithms in Artificial Intelligence

Amirmahdi Ansaripour (Student ID: 810198358)

1 Introduction

This report is an explanation about my project which is related to search algorithms. At first, the whole structure of a search problem is discussed. In the next sections, different search methods (BFS, DFS, IDS, and A*) are explained in detail.

2 The Structure of a Search Problem

Generally, each search problem should have the following features:

- **Initial state**
- **Goal state**
- **Actions** (Possible actions that the agent may take)
- **The cost between each two states**

3 Different Methods

In this section, the searching methods implemented in the project are discussed. Some figures from the project's output are also included. (Note that based on the project, our agent (Gandalf) should reach the goal state (The Castle)).

3.1 Breadth First Search (BFS)

In this algorithm, all nodes (states) in the same layer are explored before states in deeper layers. In fact, we can imagine a queue which starts with the initial state. When each state enters the explored set, its neighbours (or children) are pushed to the end of the queue (Here, the queue is the frontier set). In the next steps, each state is popped from the beginning of the frontier set.

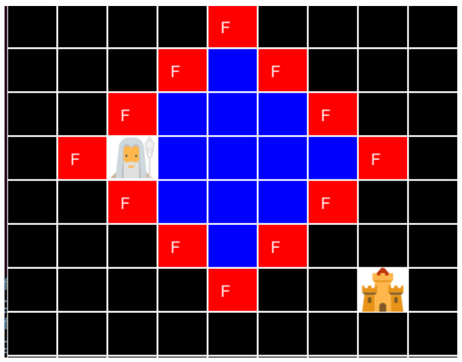


Figure 1: The frontier set (red) and explored set (blue) in BFS

Pay attention to the example in Figure 2 to see in what order each node and its children are pushed to the explored and frontier sets, respectively[1]. In addition, since each state is explored in the shallowest depth from the root, repetitive states are ignored.

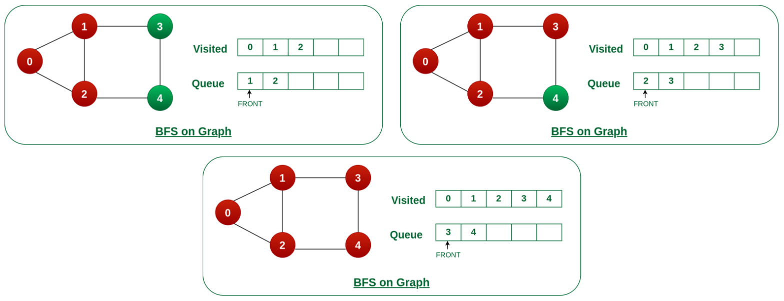


Figure 2: An example for BFS

- **Time Complexity:** $O(b^d)$

  (b is the maximum number of children that an explored node has, and d is the height in which the goal state is located)

- **Memory Complexity (number of unique nodes expanded):** $O(b^d)$

  (level-wise)

- **Completeness: yes**

- **Optimality: yes**

- **Need to analyze repetitive states: no**

[1]Images are taken from https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

3.2 Depth First Search (DFS)

In this algorithm, the path's depth increases by one in each step. Instead of using a queue (like in BFS), a stack is used when employing DFS. In each step, the top of the stack is popped and its children are pushed. Therefore, it seems logical that DFS goes down until it finds the goal or reaches leaves. When reaching the leaves, DFS backtracks and tries to find another path to the goal state [2].

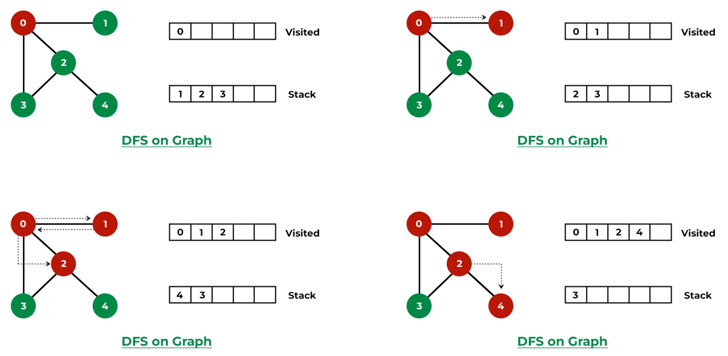


Figure 3: An example of DFS algorithm. Pay attention to the backtracking process.

Obviously, the algorithm is not optimal because it may blindly go down when a more efficient way exists. This claim can also give us a clue about the fact that DFS needs to handle repetitive nodes and replace their costs if a cheaper path is discovered.

- **Time Complexity:** $O(b^m)$

  (b is the maximum number of children that an explored node has, and m is the deepest level reached by the algorithm. It is possible for m to be infinity)

- **Memory Complexity:** $O(b.m)$

  (m is equal to the number of explored nodes, and each explored node has at most b children)

- **Completeness: yes (if $m\ != \infty$)**

- **Optimality: no**

- **Need to analyze repetitive states: yes**

[2]Images are taken from https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

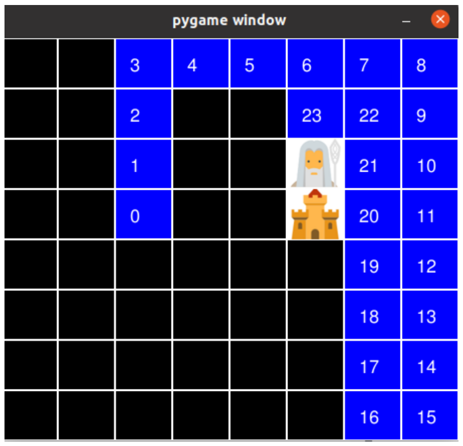
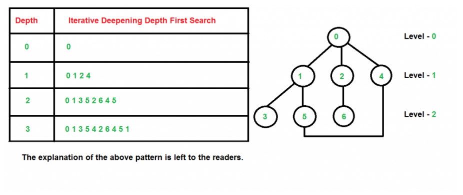


Figure 4: An example demonstrating that DFS is not optimal

3.3 Iterative Deepening Search (IDS)

The only difference that exists between DFS and IDS is that in IDS, there is a threshold for depth prohibiting the algorithm from getting down inefficiently. In fact, IDS can be viewed as a combination of BFS and DFS. In this algorithm, we start with depth one, do a DFS search, and increase the depth by one for the next step [3].



Depth	Iterative Deepening Depth First Search
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4 5
3	0 1 3 5 4 2 6 4 5 1

The explanation of the above pattern is left to the readers.

Figure 5: An example of IDS (Note how BFS and DFS are combined)

The algorithm is optimal (thus complete) provided that the threshold is set to a large number. Its time complexity is worse than those of DFS and BFS (proved below), but its memory consumption is better than that of BFS. The reason is that unique nodes are added to the explored set in a DFS manner.

- **Time Complexity:** Imagine a graph whose nodes have b children, and the goal state resides in depth d. The initial state's children are explored d times, and the initial state's grandchildren are explored $d-1$ times, and so on.

$$Time\ complexity = (d)b + (d-1)b^2 + ... + 2b^{(d-1)} + b^d = O(b^d) \quad (1)$$

[3]The image is taken from https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/

(Pay attention that although the order is the same as those of BFS and DFS, the constant coefficient related to IDS is greater.)

- **Memory Complexity:** $O(b.d)$
  (like DFS)
- **Completeness: yes**
- **Optimality: yes (both completeness and optimality are correct provided that d is a large number)**
- **Need to analyze repetitive states: no**

3.4 Heuristic Astar (briefly A*) Search

This algorithm employs an innovative approach towards reaching the goal state. Imagine that we are in state n and want to reach the goal state. We claim that:

Total cost $= g(n) + h(n) \quad (\forall n\ in\ graph)$

Where $g(n)$ is the cost from the initial state to n, and $h(n)$ is the estimated (heuristic) cost from state n to the goal. Pay attention that $g(n)$ can be easily achieved for node n: $g(n) = g(n-1) +$ cost of edge between $n-1$ to node n. As a result, the difficult part is how to estimate $h(n)$. Define $r(n)$ the real value from node n to goal:

- if $h(n) > r(n)$, then the solution will not be optimal.
- if $h(n) << r(n)$, i.e, if the estimated cost is <u>much</u> lower than the real cost, many nodes will be explored, which increases memory consumption.

One useful heuristic function is Euclidean distance. It is obvious that Euclidean distance is always smaller than Manhattan distance in a grid, so it can be used for search problems in grid areas [4].

As previously explained, if $h(n)$ is significantly lower than $r(n)$, excessive nodes are explored, which is useless. One technique to prevent such problem is called weighted A* (which has been implemented in this project). In weighted A*, we use a coefficient $\alpha > 1$ such that:

$$h(n) << r(n), \quad but \quad \alpha * h(n) <= r() \tag{2}$$

Pay attention to Figure 8 and Figure 9, and how a proper choice of α can affect the number of nodes explored. Also, by taking a close look at Figure 9, it can be seen how the direction of Gandalf (agent) is concentrated towards the goal, which is a unique feature of A* method.

[4] Image is taken from https://www.researchgate.net/figure/Euclidean-and-Manhattan-distance-comparison-3235-Optimizations-The-first-optimization_fig24_343237167

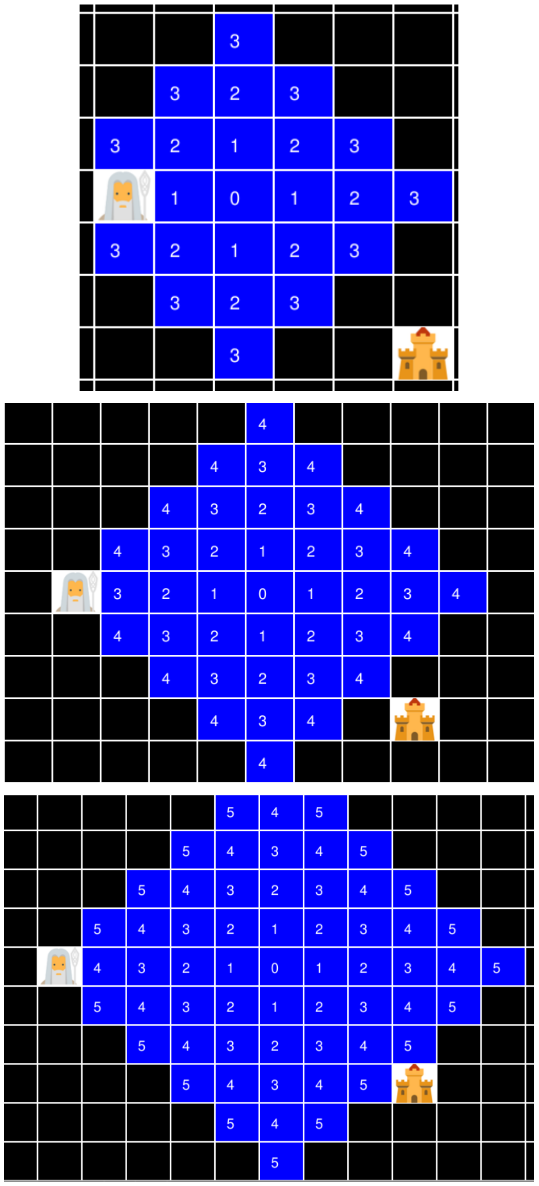


Figure 6: Different iterations in an IDS search

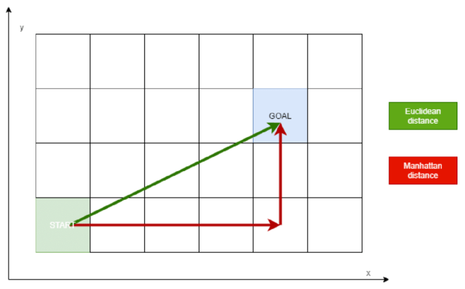


Figure 7: If Manhattan distance is the real cost in a search problem, then Euclidean distance is a proper heuristic.

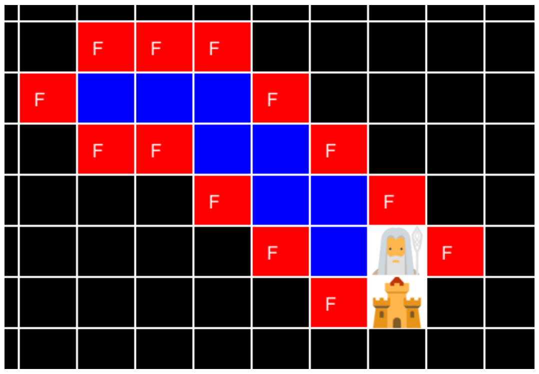


Figure 8: Using $\alpha = 1.5$

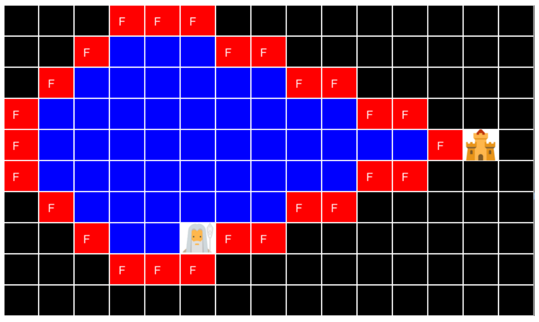


Figure 9: Using $\alpha = 0.009$

Finally, it is worthwhile to note that heuristic functions which are lower than their corresponding real costs are called admissible heuristics. In order for repetitive states not to be be handled, we should make sure that an consistent heuristic is used. Details about consistent heuristics are beyond the report's scope.

- **Time / Memory Complexity:** differs based on $h(n)$ definition.
- **Completeness: yes** (A* will finally converge to a solution)
- **Optimality: yes** (provided that $h(n)$ is admissible)
- **Need to analyze repetitive states: no** (provided that $h(n)$ is consistent)

4 Conclusion

Different searching methods have been discussed in this report. Some of their characteristics have also been mentioned. For example, time/memory complexity, optimality, completeness, and whether it is required to handle repeated states or not

5 Acknowledgement

I became familiar with the concept of search algorithms in the Artificial Intelligence Course offered by Dr.Yagoobzadeh and Dr.Fadaei in the University of Tehran, Spring 2022. I also want to thank the teacher assistants of the second assignment of this course, which was about implementing search algorithms, Sina Negarandeh and Sajjad Alizadeh.

6 Reference

1. Slides of AI Course (Dr.Yagoobzadeh and Dr.Fadaei, University of Tehran).
2. Geeks for Geeks Website (URL: https://www.geeksforgeeks.org/search-algorithms-in-ai/)