


Uninformed Search Strategy for State Space Search Solving

[ALGORITHM](#)[ARTIFICIAL INTELLIGENCE](#)[BEGINNER](#)

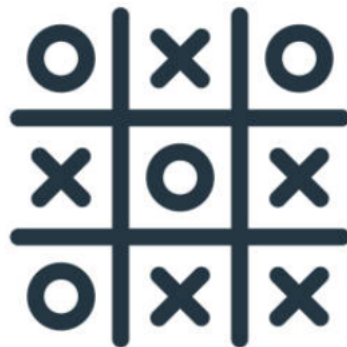
This article was published as a part of the [Data Science Blogathon](#).

Introduction

We saw the importance of PEAS representation in the article [here](#). PEAS stands for Performance evaluation, Environment, Actuators, and Sensors. Agents understand their environment through sensors and perform relevant actions using Actuators. The search strategies of the agent decide this “relevant” action. This helps the agent choose the right move to reach the destination. Hence, It is important to compute the path for the agent to serve its purpose. AI uses different Search Strategies to compute the path for the agent.

 Initial state and goal state of 8 puzzle. | [Download Scientific Diagram](#)

Why is Search Strategy Important?



A simple Tic-Tac-Toe game can have 362880(9!) different states possible, and only a few of them lead us to victory. Similarly, the famous Missionaries and Cannibals problem also has many possible states. Do I even need to say about Chess? Lmao.

Search Strategies are structured ways to solve issues. Rational or Problem-solving agents in AI usually use these search strategies to solve a given problem and provide the best solution.

Search Strategies are majorly classified into two:

- **Uninformed Search**
- **Informed Search**

Uninformed Search Strategy

Uninformed Search Strategies are search strategies that work in a brute force approach. Uninformed Search Strategy does not have any knowledge except what is defined in the problem's constraints. They only work by blindly traversing through the different states and finding the path to the destination.

The 4 parameters on which we will evaluate the strategies are:

- **Completeness:** If a search strategy can identify the path to the goal of our agent, then only we can call it complete.
- **Time Complexity:** Measure of time.
- **Space Complexity:** Maximum storage for worst-case conditions.
- **Optimal solution:** The term "optimal solution" refers to a solution for an algorithm that is the best solution possible (lowest route cost).

The 6 Uninformed Search strategies that we will cover today are,

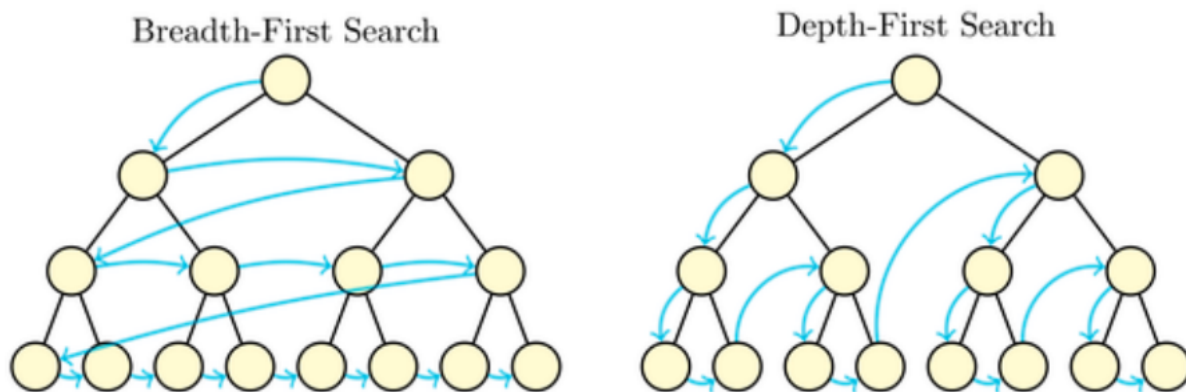
- **Breadth-first Search**

- **Depth-first Search**
- **Uniform cost search**
- **Depth-limited Search**
- **Iterative deepening depth-first search**
- **Bidirectional Search**

Breadth-First Search

The most popular search method for traversing a tree or graph is breadth-first search, which follows the level order traversing method. The breadth-first search algorithm performs level-by-level searches in a tree or graph. The BFS algorithm begins its search at the tree's root node and grows every child node at the current level before moving on to nodes at the next level. We use a queue to implement BFS.

The BFS promises to provide a solution (if it exists). But this increases the time and space complexity when the goal states are situated very deep in a tree/graph. A BFS solution by me to the Missionaries and Cannibals problem is here.



Depth-First Search

A recursive approach for exploring a tree or graph is called depth-first search. The depth-first search begins at the root node, travels down each path until the depth of the tree by expanding one of its children, and then moves on to the next path. We use Stack to implement DFS.

DFS just needs to store a stack of the nodes on the path from the root node to the current node, hence it uses relatively little memory compared to other strategies. Since DFS goes for the depths first, it sometimes enters an infinite loop. Hence, DFS doesn't ensure a complete solution.

Uniform-cost Search

Uniform-cost search can be used when the graphs have weighted vertices. It helps in finding the goal in one of the most optimal ways for a utility-based agent. The idea is to expand the cheapest node every time. This is done so by maintaining a frontier i.e. A priority queue ordered by the path cost (lower states are given higher priority).

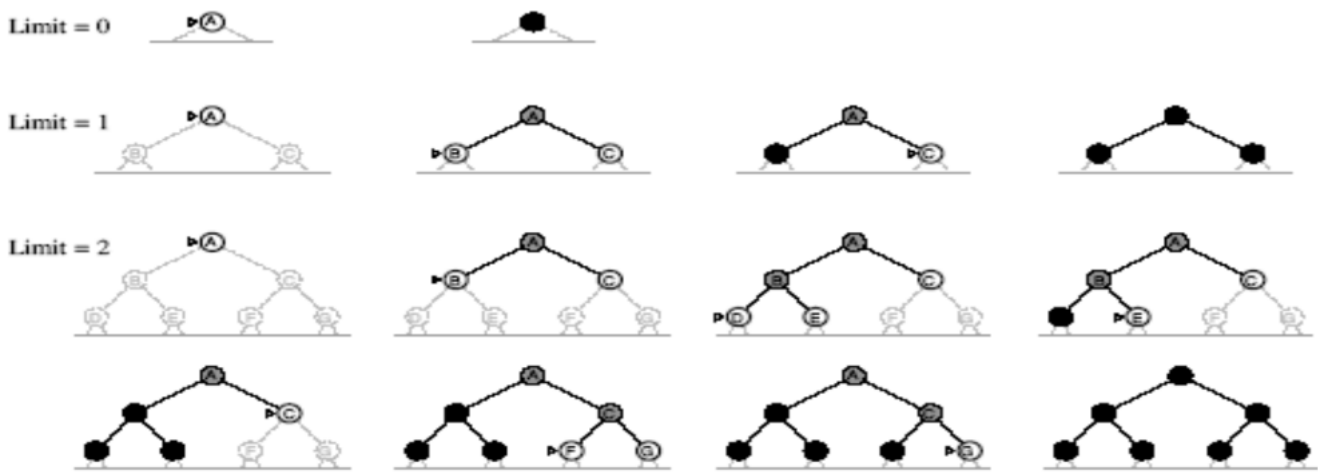
Any graph or tree that requires the optimal cost can be solved with it. This strategy sometimes loses itself in an infinite loop as it focuses more only on path cost.

Depth limited Search

Depth Limited Search is similar to Depth First Search, but it is predetermined with a depth limit (say l). The nodes below the depth l are used as if they have no successors i.e., leaf nodes. This helps us in solving the infinite path problem of DFS. But this, too, can't assure a complete solution. It may fail when our goal state lies in one of the levels below l . It recursively moves to traverse the depth of the graph until the limit is reached. If it reaches the limit, it cutoffs and backtracks toward the other nodes for searching.

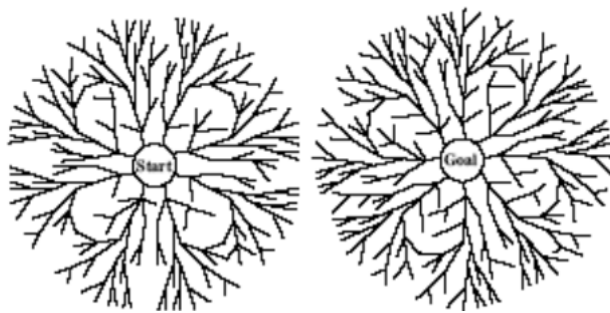
Iterative deepening depth-first search

This is one of the widely used strategies as it looks to combine the benefits of both the DFS and the BFS. This is beneficial for cases when the search space is too large and the solution's depth is unknown. It starts with depth limit 0 of the tree and explores all its nodes before increasing the limit by 1 every time. This is done until one of the goals is found. Like DFS, its memory requirements are modest. Like BFS, it promises a complete solution and an optimal one too, when the path cost is a non-decreasing function of the depth of the node. For a graph/tree with branching factor 10 and depth 5, the Breadth-first search traverses 111,110 nodes in its worst case. At the same time, the iterative Deepening Depth-first search traverses 123,450 nodes.



Bidirectional Search

Bidirectional Search is a two-way search. i.e., one forward search from the initial state and the other backward search from the goal state. We hope the two searches may meet in the middle of one of the states. The notion of using this search is that $b^{(d/2)} + b^{(d/2)}$ is much less than b^d . Replace the goal test and determine whether the frontiers of the two searches intersect, if they do, a solution has been found. A schematic view of a bidirectional search is about to succeed when a branch from the start node meets a branch from the goal node.



Conclusion

The above comparison chart compares the different uninformed search strategies based on the 4 parameters. This helps in understanding and correctly choosing our strategy for our agent to solve the problem we are working for. We can conclude that based on the requirements and expectations from the agent, we can choose a strategy and work accordingly.

What if I told you we could explore to experience and use the heuristics to get better strategies? Let's see in the next article.

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

Article Url - <https://www.analyticsvidhya.com/blog/2022/09/uninformed-search-strategy-for-state-space-search-solving/>

NEHAAL PANDEY

