Artificial Intelligence Cheatsheet

Edoardo Riggio

January 26, 2022

Artificial Intelligence - SA. 2021
Computer Science
Università della Svizzera Italiana, Lugano

Contents

1 Blind Search Algorithms

A **search algorithm** takes as input a problem space and a starting state, and tries to compute a path in the best possible way.

The strategy is to search which node to expand among the yet undiscovered nodes. In order to expand a node, we need to consider all of the nodes that are reachable in one step from the selected node.

1.1 Definitions

1.1.1 Search Tree

A search tree is a tree composed of nodes. Each node in the tree represents a step in the search algorithm. A node is composed of:

- State
- Node who generated it
- Action used to generate it
- Depth of the tree
- Cost of the path from the root

1.1.2 Travelling Salesman Problem

The goal of this problem is to visit all the cities of a graph such that the travelling cost is minimum. The travelling cost is measured by summing up all of the travelling costs from the starting city to the destination city.

1.1.3 Evaluation of Search Strategies

There are four common criteria for evaluating search strategies:

- **Completeness**

  Does the algorithm always find a solution if one exists?

- **Optimality**

  Does the algorithm guarantee the least-cost solution?

- **Time Complexity**

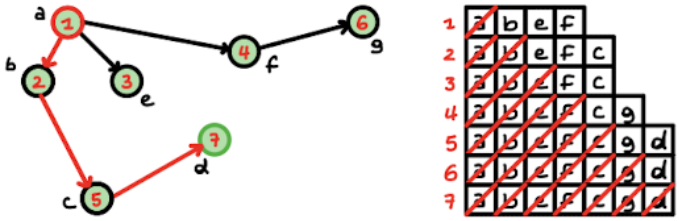
  How much time does it take for an algorithm to run?

- **Space Complexity**

  How much memory does the algorithm require?

1.2 Breath-First Search

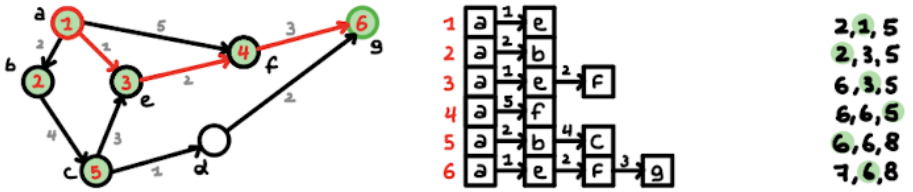
In this algorithm, the shallowest unexpanded node is expanded. In order to do so, this algorithm requires a FIFO queue. This type of queue is able to keep track of which node to expand next.



This algorithm is **complete**. It is also **optimal** in the case that all of the steps have the same cost. Furthermore, the **space and time complexities** of this algorithm are both $O(b^d)$. Where b is the branching factor, and d is the solution depth.

1.3 Uniform-Cost Search

In this algorithm, a cost is assigned to each edge of the graph. Here a node is expanded if $g(n)$ – i.e. the distance from the root to the node – is minimal.

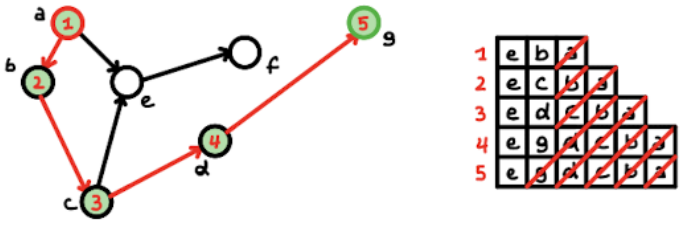


In the above image, on the far right we have the values of $g(n)$ at each step, while in the middle, we have the path chosen at each step.

This algorithm is **complete**. It is also **optimal** in the case that all costs are non-negative. Furthermore, the **space and time complexities** are both $O(b^d)$, where b is the branching factor, and d is the solution depth.

1.4 Depth-First Search

This algorithm expands the deepest unexpanded node. In order to do so, it uses a LIFO queue.



This algorithm is **not complete** in the case of an infinite state space – since it has no depth limitation. It is also **not optimal**. Furthermore, the **time complexity** of the algorithm is $O(b^m)$, while the **space complexity** is $O(b \cdot m)$. Where b is the branching factor and m is the maximum depth of the search tree.

1.5 Depth-Limited Search

In this case we have an algorithm which is identical to the DFS search. The only difference between the two is that depth-limited search is – as the name suggests – limited. This means that it will stop when it reaches a certain depth in the solution tree.

This algorithm is **complete** in the case where we know a bound to the solution depth. It is **not optimal**. Furthermore, the **time complexity** is $O(b^d)$, and the **space complexity** is $O(b \cdot d)$. Where b is the branching factor and d is the maximum depth.

1.6 Iterative Deepening Search

This is a depth-limited search algorithm, where at each step, a counter representing the tree's depth is incremented. DFS is performed at every repetition of the algorithm – when the algorithm goes back to the starting node, and it stops at the depth indicated by the counter.

This algorithm is **complete**. It is also **optimal** in the case that there are non-negative costs. Furthermore, the **time complexity** is $O(b^d)$, and the **space complexity** is $O(b \cdot d)$. Where b is the branching factor and d is the depth at which the goal is found.

1.7 Bi-Directional Search

This algorithm considers two queues, one only containing the root, and one only containing the goal. The nodes of each queue are expanded until both queues contain a common node.

This algorithm is both **complete** and **optimal**. Furthermore, the **time and space complexities** are $O(b^{\frac{d}{2}})$, where b is the branching factor, and d is the depth at which the goal is located.