Uniformed/Blind Search

In order to solve search problems, we need to define **states**, **initial state**, **actions (operators)**, **goal state (target)** and **utility (cost)**.

To represent the problem in the computer we use **states** because they are a representation of the current situation. So, in a problem we have a **state space** which refers to all the possible situations and a **solution** which refers to the **goal state** (to reach this solution, we need **strategies**).

To move from one state to another we use **operators**, these are actions that let us move from one state to another. We may be in a case where from one state we use an operator, and it generates a **not allowed** state.

So uniformed/blind search is about searching when we do not have additional information apart from the definition of the problem. We can only generate successor states and check if one of those is the definition.

In almost all cases the search is performed in a graph search, so from this graph, a **search tree** is made. In this tree the nodes correspond to search states, the root node corresponds to the initial search state, actions expand the current search node generating children by applying a function in the current node, there's a goal state and the utility is a cost of the path from root node to the current one. Although the graph is finite, the search tree can be infinite, this is due to the cycles that the graph can have.

We have two main structures, **opened-list** in which the nodes are saved with generated states pending expansion, and **closed-list** in which the states already expanded or visited are saved (only in some algorithms). The nodes can be **generated** (have appeared in opened-list), **generated but not expanded** (appear in opened-list) and **expanded** (appear in the closed-list).

The performance of an algorithm can be affected if we delete duplicate states, the complexity of the search problem can increase exponentially due to cycles.

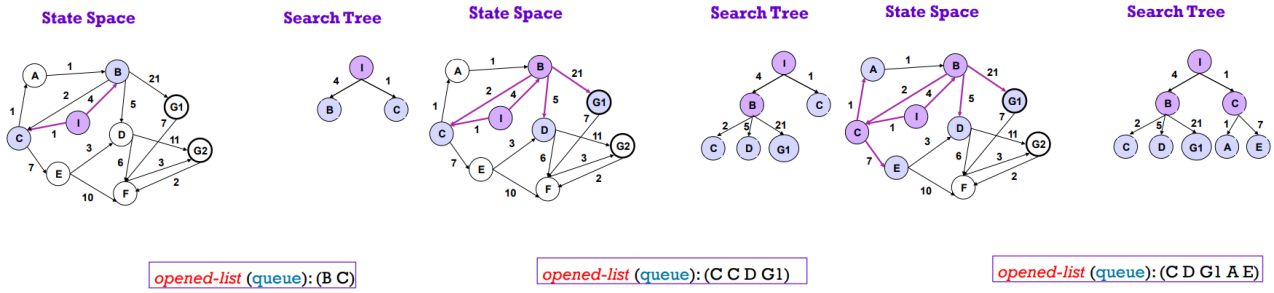
So, when we use an algorithm for searching, we should take into account these characteristics:

- Completeness. Does it guarantee to find a solution?
- Optimality. Does it find the cheapest solution?
- Cost of the solution. This is given by the utility function.
- Computational cost of the search:
    - Temporal. How long does it take to find a solution?
    - Spatial. How much memory is used?

BREADTH-FIRST SEARCH

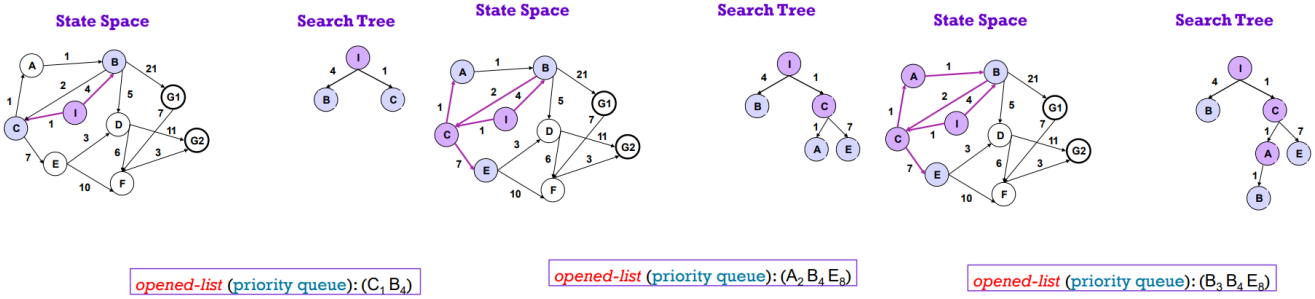
The algorithm is **complete**, it is **optimal** only if the cost is a non-decreasing function of the node depth, the time complexity is **exponential** $O(b^d)$ (being b the **branching factor** – number of children of a node, and d a path to the solution) and the spatial complexity is $O(b^d)$.

If we don't eliminate repeated states, solutions are not lost, although they provoke inefficiency. If there is not a solution, the algorithm might not finish.



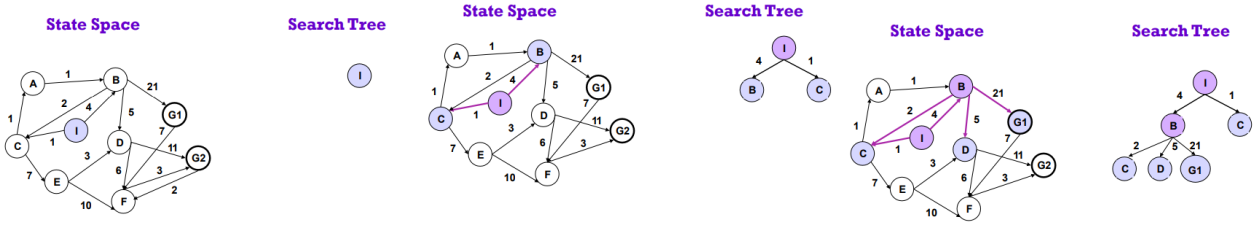
UNIFORM COST SEARCH

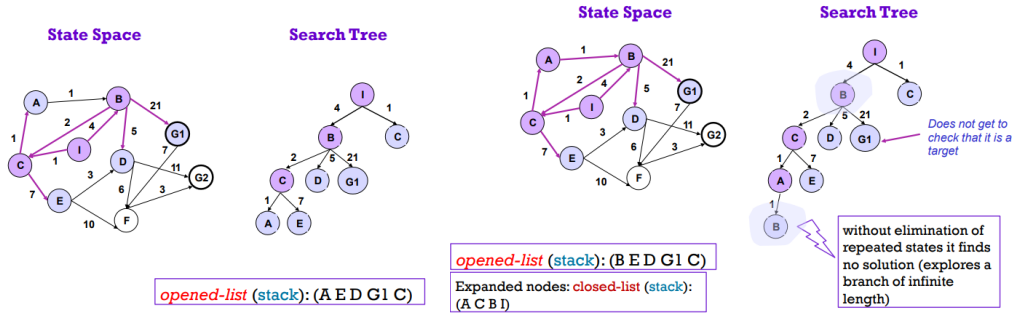
The algorithm is **complete** if there are not infinite paths, it is **optimal** if the path-cost successor(n) ≥ path-cost(n), this is satisfied if all operators have a cost ≥ 0, the complexity in space and time is $O(b^d)$. If the algorithm is not optimal then the worst case is $O(b^{\lceil C*/\varepsilon \rceil})$ being C* the cost of the path with the optimal solution and ε the minimum cost.

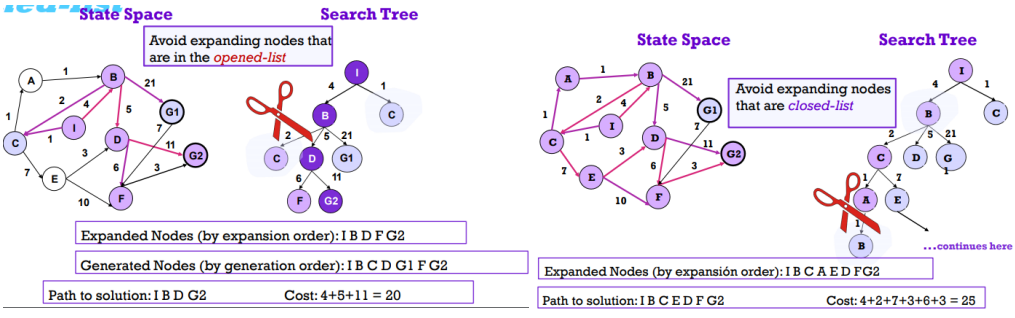


DEPTH-FIRST SEARCH

The algorithm is **not complete** because it can explore infinite paths, it is **not optimal** because finding the solution is not guarantee, the space complexity is $O(b * m)$ where b is the branching factor and m the maximum tree depth, and the time complexity is $O(b^m)$.







DEPTH-LIMITED SEARCH

The algorithm is **complete** only if L ≥ d, being L the limit height and d the depth of the shallowest solution. It is **not optimal** the is no guarantee that it finds a solution, the temporal complexity is **exponential** $O(b^L)$ and the spatial complexity is **linear** $O(b * L)$.

ITERATIVE DEEPING SEARCH

The L is set in an iterative way. The algorithm is **complete**, it is **optimal** only if the path cost is a non-decreasing function of the node depth, the time complexity is $O(b^d)$ and the spatial complexity is $O(b * d)$.

BIDIRECTIONAL SEARCH

Two simultaneous searches are executed, one from the initial state and another one from the target state. The algorithm is **optimal** and **complete** only if the path cost is a non-decreasing function of the node depth, the time complexity is $O(b^{d/2})$ and the spatial complexity is $O(b^{d/2})$.

Blind Search	Complete	Optimal	Time Efficiency (worst case)	Space Efficiency (worst case)
Breadth-first	yes	yes cost ∝ depth	$O(b^d)$	$O(b^d)$
Uniform Cost	If there are no paths of finite cost and infinite length	yes operators cost > 0	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Depth-first	No	No	$O(b^m)$	O(b*m)
Limited depth	yes L ≥ d	No	$O(b^L)$	O(b*L)
Iterative deepening	yes	yes cost ∝ depth	$O(b^d)$	O(b*d)
Bidirectional Breadth-search	yes (width)	yes cost ∝ depth	$O(b^{d/2})$	$O(b^{d/2})$