AI ASSIGNMENT 03 :

Protyush Kumar Das and Somesh Devagekar

Question 1

1.1 What is the fundamental problem, when applying an uninformed search algorithm on graphs?

Ans: Uninformed strategies use only the information available in the problem definition, also called as blind search. These include:

Breadth First Search (BFS)
Uniform Cost Search (UCS)
Depth First Search (DFS)
Depth Limited Search (DLS)
Iterative Deepening Search (IDS)
Bidirectional Search (BS)

In graph search algorithms, DFS's main advantage is space efficiency. It is its main advantage on BFS. However, if we keep track of visited nodes, we lose this advantage, since we need to store all visited nodes in memory. The size of visited nodes increases drastically over time, and for very large/infinite graphs - might not fit in memory. DFS can get stuck in infinite loops.

1.2 How does the general graph-search algorithm address this problem?

ANS: Sometimes DFS can be in an infinite branch. An infinite branch is a branch that does not end, and also does not get us to our target node, so for DFS, we might keep expanding this branch inifinitely, and 'miss' the good branch, that leads to the target node.

We can overcome this flaw in DFS, while maintaining relatively small memory size by using a combination of DFS and BFS: Iterative Deepening DFS

1.3 Is the distinction between tree and graph-search specific to each individual search-algorithm or can all search algorithms be implemented in both flavors?

ANS:A tree is a graph, but one which among other criteria is minimally connected (only one path between any two nodes) and acyclic (i.e. no loops).

So, for searching, algorithms operating on trees can make a certain set of assumptions which allow optimizations, not possible on a generalized graph. For example, for tree traversal we know we will visit each node only once (due to the minimal connectivity), but for other graphs we need to keep track of visited nodes if you don't want to process them multiple times (as there could be multiple paths leading to the same nodes). Yes all search algorithms can be implemented on trees as well as graphs.

The main difference is in the search pattern that is used to search through the graph.

Graph search uses more memory because it stores all the cycles of the tree.

1.4 Given a problem, that is solvable by a search algorithm. How do we chose between tree and graph search?

ANS:A problem is defined by two things:

• A search space:– The set of objects among which we search for the solution.
Examples: routes between cities, or n-queens configuration
• A goal condition – Characteristics of the object we want to find in the search space?
Examples: • Path between cities A and B

protyush.das@smail.h-brs.de

AI ASSIGNMENT 03 :
Protyush Kumar Das and Somesh Devagekar

• Non-attacking n-queen configuration

In Graph Search we hold a list of explored nodes, while in Tree Search we don't. In simple words, tree does not contain cycles and where as graph can. Sometimes for example in games we have repeated states and you have to explore them again, here a graph can be used. Another aspect is tree will typically have some kind of topological sorting or a property like binary search tree which makes search so fast and easy compared to graphs. (*Ref:https://stackoverflow.com*)

Question2

2.1 What are the properties of Iterative-deepening depth-first search? Is it optimal and does it find shortest paths in your map? Explain why / why not?

Does a complete search:-It combine DFS space efficiency and BFS completeness and time complexity. It perfect is when the memory is constrained and is guaranteed to find the shortest path leading from the given start node to any goal node in the problem graph.

Required time=O(branching factor^depth-of-optimal-solution)

A tree traversal algorithm which generally operates in O(bd) space complexity where b is the branching factor and d is the depth-of-optimal-solution.

Using iterative deepening, this algorithm is run over and over again, m times at increasing depth:

$O(b^m) = b^0 + b^1 + b^2 + ... + b^m$

Thus the most significant element over time is b^m, where m is the max depth reached.

But since it stops at the depth ‘d’ hence the time complexity sums up to O(b^d)

Required memory=O(branching factor*d)-Here the node only on the current path are remembered, thus the amount of memory is linear to the length of the solution the search constructs.

Optimal solution:- In the given maps for traversal it doesn't give the optimal solution. The reason behind it is that the map contains multiple goals at several levels. Hence we cannot set a static limit to bind the iterative deepening search and it has to search the entire tree for many iterations and ends up with time complexity of O(b^m)

protyush.das@smail.h-brs.de