Tree Search & Graph Search

18 October 2020 23:28

Objectives:

- Graph search algorithms
- Basic tree search

Tree Search

In the vacuum example, there were 2 squares, and a binary dirt option – dirt or no dirt

It was simple to compute the state space diagram for this scenario.

In real life, e.g. a self-driving car on a street, there are going to be thousands of variables and thus an intractable number of potential states.

How do we deal with this?

Algorithm

- Offline, simulated exploration of the state space
- Starting with a start state, expand one of the explored states by generating its successors, e.g. find neighbours by considering possible actions to build a search tree.

```
function TREE-SEARCH(problem,strategy)
    returns solution, or failure
  initialise search tree using initial state of problem
  loop do
    if no candidates for expansion then return failure
    choose leaf node for expansion according to strategy
    if node contains a goal state
        then return corresponding solution
    else expand node and add resulting nodes to
        search tree
  end
```

Evaluating Search Strategies

- Completeness - does the algorithm always find a solution?
- Optimality – does the algorithm always find a least-cost solution?
- Time complexity – what is the maximum branching factor, depth of least cost solution, maximum depth of state space?
- Space complexity – what is the maximum number of nodes in memory?

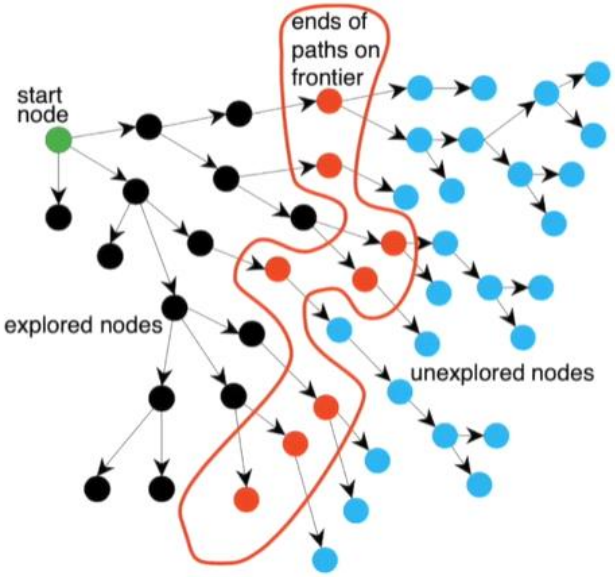
Graph Search

With a tree search, state spaces with loops give rise to repeated states that cause inefficiencies and infinite loops

Graph search is a practical way of exploring the state space that can account for such repetitions

1. Given a graph, incrementally explore paths from the start nodes

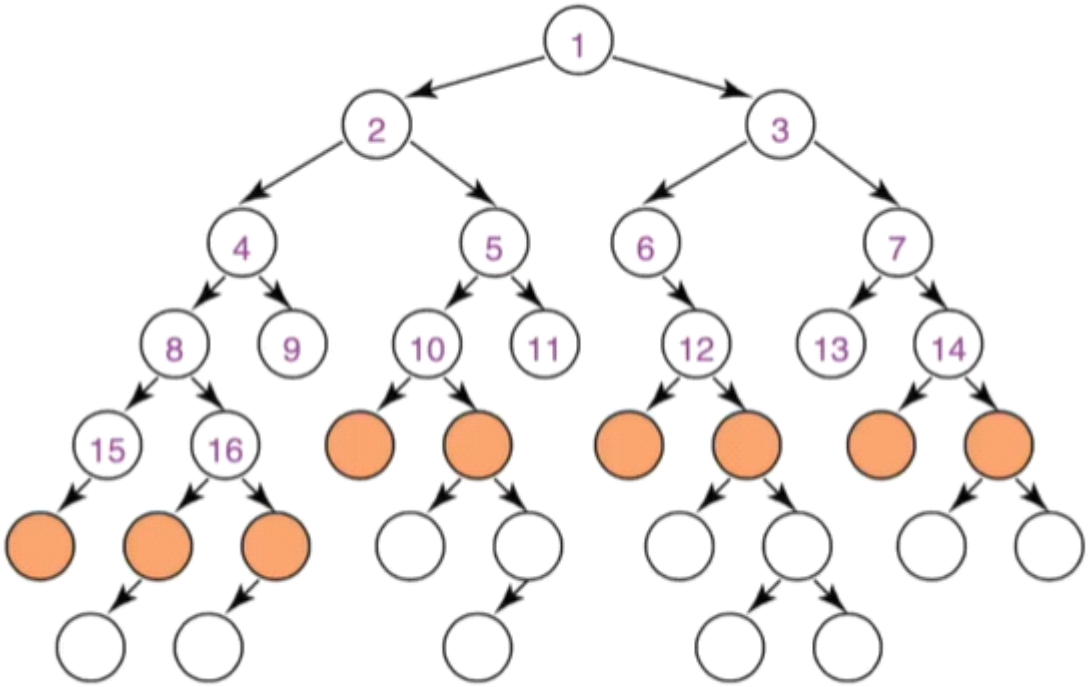
2. Maintain a frontier of paths
3. As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered
4. The way in which the frontier is expanded is defined by the search strategy



Starts with a set of nodes
We check if we're at the goal
We start with a frontier – at the start, we just have the start node
We check the frontier, and look for a goal. If none found, add next nodes to frontier and repeat.

Breadth-First Graph Search

Treat frontier as a queue
Select earliest elements added to frontier



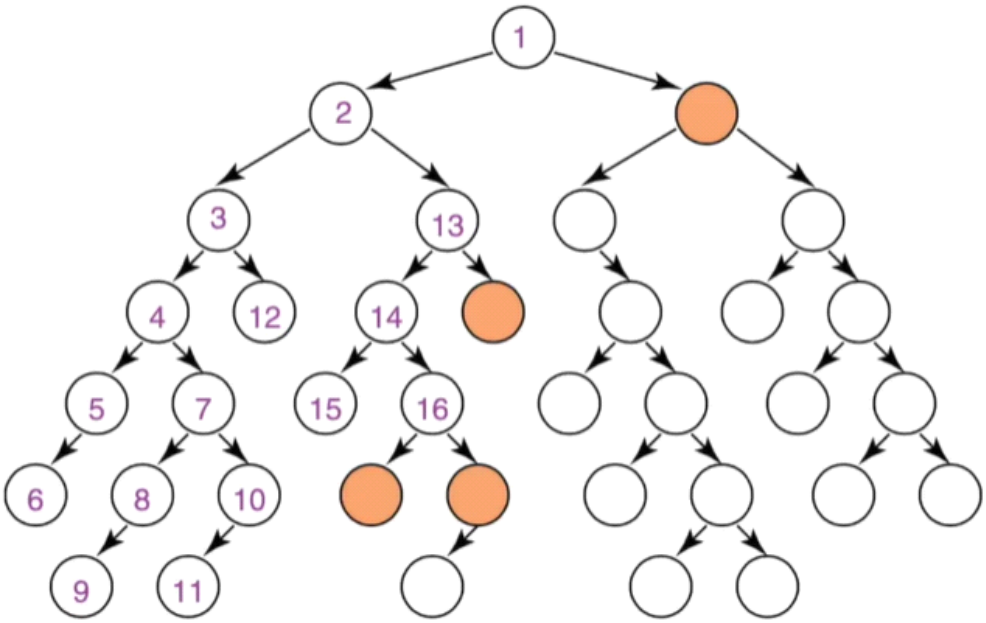
- If branching factor for all nodes is finite, bf graph search will find a solution
- Time complexity is exponential in path length - branching factor the power of n where n is the path length
- The space complexity is exponential in path length b^n
- The search is unconstrained by the goal

Depth-First Graph Search

Treat the frontier as a stack
Select the last element added to the frontier

If the list of paths on the frontier is p1, p2, p3...

- P1 is selected and the paths that extend p1 are added to the front of the stack in front of p2
- P2 is only selected when all of the paths from p1 have been explored



Complexity of a Depth-First Graph Search

- Not guaranteed to halt if we have cycles of infinite graphs
- The space complexity is linear in the number of arcs from the start of the current node
- If the graph is a finite tree, with a forward branching factor <= b and all paths from the start having at most k arcs, worst case time complexity is O(b^k)
- The search is unconstrained by the goal

Lowest-Cost-First Search

- Sometimes there are costs associated with arcs
- The cost of a path is the sum of the costs of its arcs
- An optimal solution is one with the minimum cost
- At each stage, the lowest-cost-first search selects a path on the frontier with the lowest cost
- The frontier is a priority queue ordered by path cost
- The first path to a goal is a least-cost path to a goal node

- When arcs costs are equal then breadth-first search

Summary of Uninformed Search Strategies

Strategy	Frontier Selection	Complete	Halts	Space
Breadth-first	First node added	Yes	No	Exp
Depth-first	Last node added	No	No	Linear
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp

Complete — guaranteed to find a solution if there is one (for graphs with finite number of neighbours, even on infinite graphs)
Halts — on finite graph (perhaps with cycles)
Space — as a function of the length of current path.