Real World Problems
Uninformed Search Methods

REAL-WORLD PROBLEMS

Route Finding

- Defined in terms of Specified Locations & transition along link between them.

Applications

- Automated Travel Advisory System
- Airline Travel Planning
- Car Systems

REAL-WORLD PROBLEMS

- Route-finding problems
  - GPS-based navigation systems, Google maps
- Touring problems
  - TSP problem
- VLSI layout problems
- Robot navigation problems
- Automatic assembly sequencing
- Internet searching
- Searching paths in metabolic networks in bioinformatics

REAL-WORLD PROBLEMS

Let us consider airline travel planning problems that is solved by an automated travel advisory system. The objective is to arrive to a destination with the minimum cost. Each state comprises airport location and the current time. Let us formulate the problem.

Initial state: Starting point of the journey

Goal state: Final destination

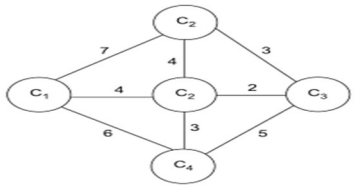
Operators/Actions: Flight from current location to reach the next location in such a way so that there is enough time difference in the next flight and in transfer within the airport if there is a connecting flight.

REAL-WORLD PROBLEMS

Path cost: Total of travel fare, travel time, waiting time, time of the day, type of airline and can include special package, mileage awards, etc.

Most often, the commercial travel systems use a problem formulation of this kind, with many additional complications such as backup reservations on alternative flights. This is done to the extent that these are justified by the cost and likelihood of the original plan.

Travelling Salesman Problem



The problem can be formulated as

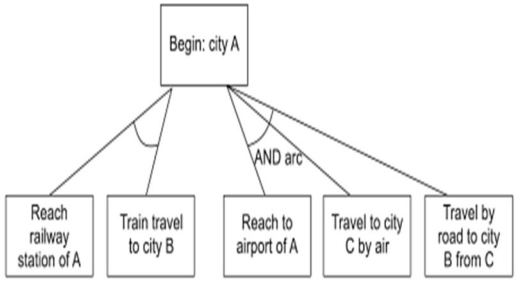
Initial state: The starting city for the tour is C_1.

Goal state: The complete trip ends at C_1.

Operator: Travel from one city to another with the constraint that the city is not visited before.

Path cost: Travelling time

Problem Reduction Methods



Search Methods

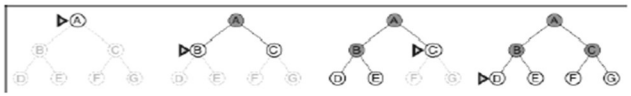
- Uninformed (blind) search strategies use only the information available in the problem definition

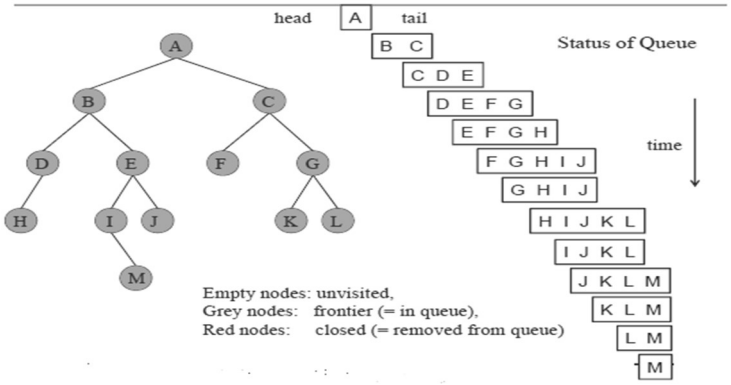
- Breadth-first search (BFS)
- Uniform-cost search
- Depth-first search (DFS)
- Depth-limited search
- Iterative deepening search

Breadth-First Search (BFS)

- The root node is expanded first
- Then all successors of the root node are expanded
- Then all their successors
- … and so on

- In general, all the nodes of a given depth are expanded before any node of the next depth is expanded.
- Uses a standard queue as data structure





Uniform-Cost Search (UCS)

- Modification to BFS generates Uniform-cost search, which works with any step-cost function (edge weights/costs):
- UCS expands the node *n* with lowest summed path cost *g*(*n*).
- To do this, the frontier is stored as a priority queue. (Sorted list data structure, better heap data structure).
- The goal test is applied to a node when selected for expansion (not when it is generated).
- Also a test is added if a better node is found to a node on the frontier.

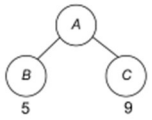
Uniform-Cost Search (UCS)

Let us discuss how the uniform cost search proceeds step-wise.

Step 1: Start from *A* — Expanded none

Step 2: Frontier — Expanded none

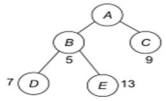
Step 3: Frontier: *B* and *C* — Expanded *A*



Select the lowest, i.e., *B*

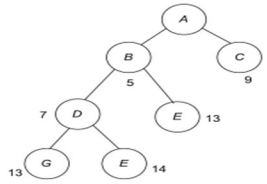
Uniform-Cost Search (UCS)

Step 4: Frontier: *D*, *C*, *E* (priority queue) Expanded *A*, *B*



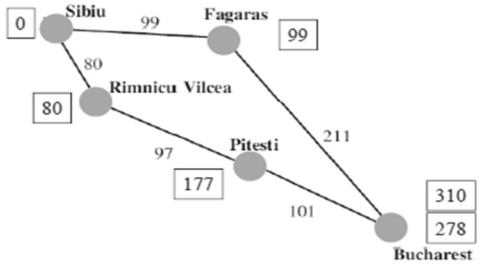
Note: Remember the costs are added from the root till the node in consideration. Select the lowest, i.e., *D*.

Step 5: Frontier (*C*, *E*, *E*, *G*) Expanded *A*, *B*, *D*



In step 5, we have considered *E*, as its parent is different. Since we have reached the goal state, from the tree, we can see that we have found a path. The path is *A-B-D-G*.

Uniform-Cost Search (UCS)

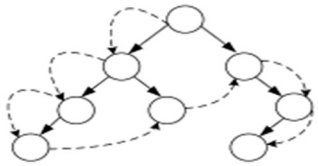


- Uniform-cost search is similar to Dijkstra's algorithm!
- It requires that all step costs are non-negative
- It may get stuck if there is a path with an infinite sequence of zero cost steps.
- Otherwise it is complete

Depth First Search (DFS)

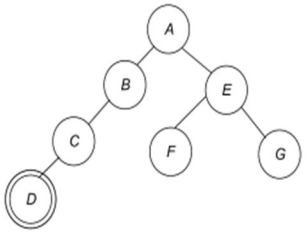
1. If initial state is the final state
2. Do it till you get success or failure.
   Generate a successor, if no more successors can be generated → failure
   Go to step 1 with initial state to be the current generated successor.

As described, the search starts with the initial node or state say root. It proceeds with its successor and follows the path ahead till either is knows that it has come to a solution or has to go back and seek new solution as shown in the figure.



Depth First Search (DFS)

With the start node of *A* and the goal state of *D*, we have the DFS approach carried out as follows:



Sample tree input for DFS.

Depth First Search (DFS)

	Stack	
Step 1:	$A \leftarrow$ top $B \leftarrow$ top E	Pop A, is it goal? No, push its children on the stack. A's successor is pushed.
Step 2:	$B \leftarrow$ top E $C \leftarrow$ top E	Pop B, is it goal? No, push its children on the stack. B's successor is pushed.
Step 3:	$C \leftarrow$ top $D \leftarrow$ top E	Pop C, is it goal? No, push its children on the stack. C's successor is pushed.
Step 4:	$D \leftarrow$ top E	Pop D, is it goal?? YES!

Depth First Search (DFS)

Consider the second case, where we want to reach to F.

The initial working is same as the way we had to reach to D. The example below shows that after having reached till D state, the next step needs backtracking and hence it starts with E. So, we will start from that point. Thus, the stack contents are as follows:

	Stack	
Step 1:	$E \leftarrow$ top	After having traversed the steps till D, stack will have E. So, pop E, is it goal? No, push its children.
Step 2:	$F \leftarrow$ top G	E's successor is added.
Step 3:	$F \leftarrow$ top G $G \leftarrow$ top	Pop F, is it goal?? YES!

Pop the contents. Path found! So, the path A-E-F is found.

Advantages of Depth First Search and Breadth First Search

Advantages of Depth First Search	*Advantages of Breadth First Search*
1. Simple to implement 2. Needs relatively small memory for storing the state-space	1. Guaranteed to find a solution (if one exists) 2. Depending on the problem, can be guaranteed to find an optimal solution

Disadvantages of Depth First Search and Breadth First Search

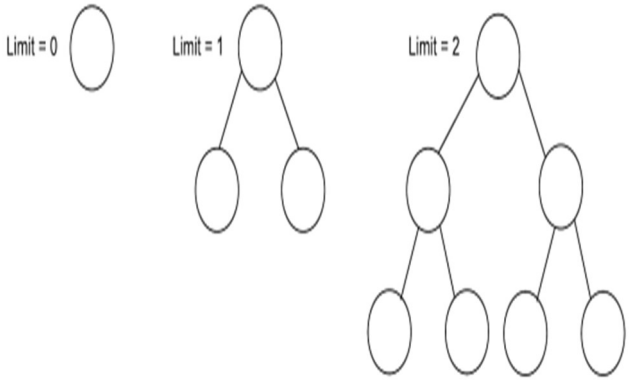
Disadvantages of Depth First Search	*Disadvantages of Breadth First Search*
1. Cannot find solution in all cases, and hence, can sometimes fail to find a solution 2. Not guaranteed to find an optimal solution can take a lot of time to find a solution	1. Memory management along with allocation is the key factor, and hence, more complex to implement 2. Needs a lot of memory for storing the state space if the search space has a high branching factor

Depth Limited Search (DLS)

The following algorithm summarises the working of depth limited search:

1. Set the depth limit to the maxdepth to search.
2. Initial node = current node
   If initial node = goal, return
3. If depth(initial node) > depth limit
   return
   else
   Expand(initial node)
   Save(successors) using stack
   Go to step 2

Iterative Deepening Search (IDS)



The algorithm for IDS is as follows:

1. Set depth-limit ← 0
2. Do
   Solution = DLS(depth-limit, initial node)
   If(solution= goal state) then return
   else
   Depth-limit = depth-limit + 1
   continue

Comparison of Uninformed Search Techniques

Search techniques	*Complete*	*Optimal*	*Time complexity*	*Space complexity*
Breadth first search	Yes	Yes	$O(b^d)$	$O(b^d)$
Uniform cost	Yes	Yes	$O(b^d)$	$O(b^d)$
Depth first	No	No	$O(b^d)$	$O(bd)$
Depth limited	No	No	$O(b^l)$	$O(bl)$
Iterative deepening	Yes	Yes	$O(b^d)$	$O(bd)$
Bi-directional	Yes	Yes	$O(b^{d/2})$	$O(b^{d/2})$