Artificial Intelligence

Lecture 2, Chapter 3

Solving Problems by Searching

Supta Richard Philip

Department of CS
AIUB

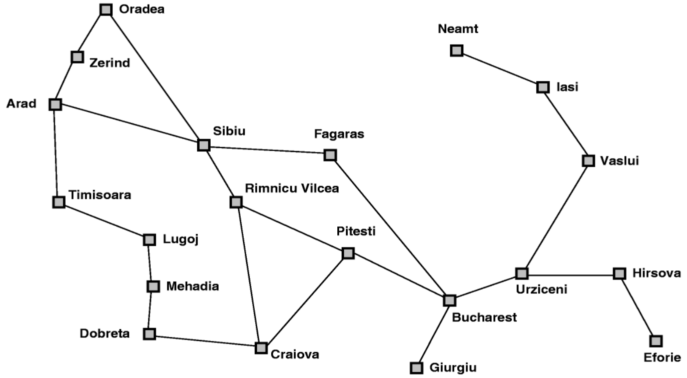
AIUB, November 2025

Table of Contents

Table of Contents

Problem Solving Agent

- A problem-solving agent is one kind of goal-based agent.
- Let us consider the route-finding problem in the cities of Romania.
- This problem can be solved by searching.



Problem formulation

- Intelligent agents are supposed to act in such a way that the environment goes through a sequence of states that **maximizes the performance measure**.
- We will consider **a goal to be a set of states** - just those states in which the goal is satisfied.
- **Actions** can be viewed as causing **transitions between states**.
- Problem formulation is the process of deciding what actions and states to consider.

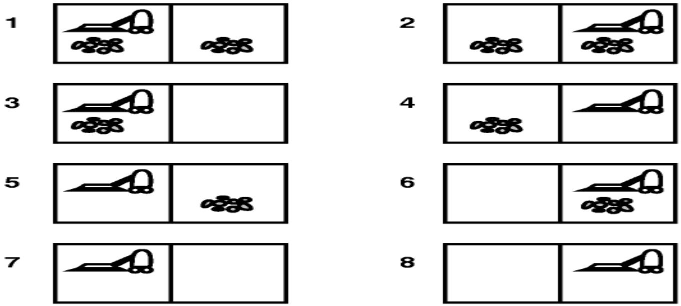
Table of Contents

Search Problem formulation

A search problem consists of:

- A state space
- A successor function(with action, cost)
- A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Example- Vacuum World



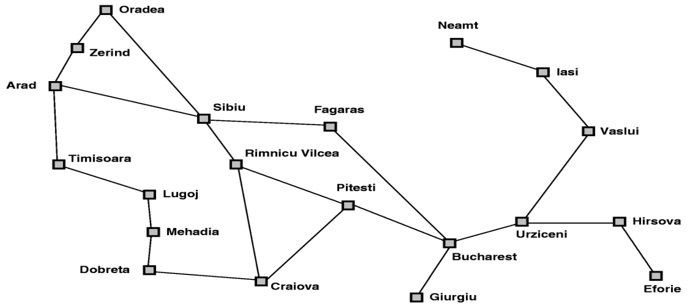
- In this case there are eight possible world states.
- There are three possible actions: left, right, and suck.
- The goal is to clean up all the dirt, i.e., the goal is equivalent to the set of states 7,8.

Search Problem formulation for vacuum world

This can be formulated as a problem as follows:

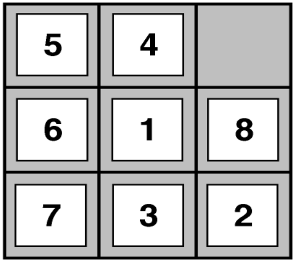
- **States:** The state is determined by both the agent location and the dirt locations. Thus, there are $2.2^2 = 8$ possible world states. A larger environment with n locations has $n.2^n$ states.
- **Initial state:** Any state can be the initial state.
- **Actions:** In this simple environment, each state has just three actions: **Left, Right, and Suck**. Larger environments might also include **Up and Down**.
- **Transition model:** The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and sucking in a clean square have no effect.
- **Goal test:** This checks whether all the squares are clean.
- **Path cost:** Each step costs 1, so the path cost is the number of steps on the path.

Example - Route-Finding in Romania

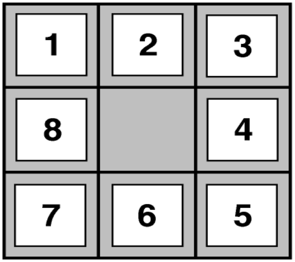


- initial states: Arad
- goal state: Bucharest
- operators: successor function $S(x)$ set of possible actions
- path cost: a function that assigns a cost to a path.

Example - The 8-puzzle problem



Start State

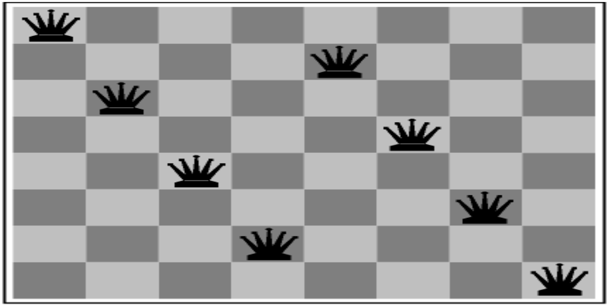


Goal State

- states: a state description specifies the location of each of the eight tiles in one of the nine squares. For efficiency, it is also useful to include a location for the blank.
- operators: blank moves left, right, up, or down.
- goal test: as in figure
- path cost: length of path

Example - The 8-queens problem

The goal of this problem is to place 8 queens on the board so that none can attack the others. The following is not a solution!



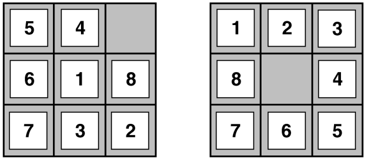
- goal test: 8 queens on board, none attacked
- path cost: irrelevant
- states: any arrangement of 0-8 queens on the board
- operators: add or remove a queen to/from any square

Search Problem formulation Exercise

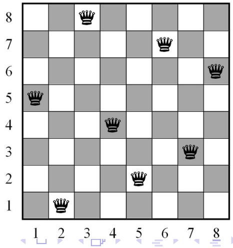
Formulate the following problem as the search problem.

- **The 8-puzzle:** The 8-puzzle has $9!/2 = 181,440$ reachable states and is easily solved. **Why?**. Please find the mathematics behind this.
- **8-queens problem:** Initial state-empty board then possible world states

$$\binom{64}{8} = \frac{64!}{8!(64-8)!}$$



Start State
Goal State



8-queens problem formulation

A better formulation

- **Initial state:** the queens are already placed on the chessboard so that each column has exactly one queen.
- **Actions:** a queen can be moved anywhere in its same column.
- **How many successors does a state have?(Branching factor)**
  - In its own column, it can be moved to any of the 7 other rows (excluding its current one).
  - There are 8 columns (1 queen per column).
  - Number of successors=8*7
- **What is the size of the state-space?**
  - Column 1: queen can be in 8 rows
  - ....
  - Column 8: 8 options
  - State space size=8^8

Table of Contents

State space graph

A state space graph is a mathematical representation of a search problem.

- Nodes are (abstracted) world configurations
- Arcs represent transitions resulting from actions
- The goal test is a set of goal nodes
- Each state occurs only once

The state space for the vacuum world.

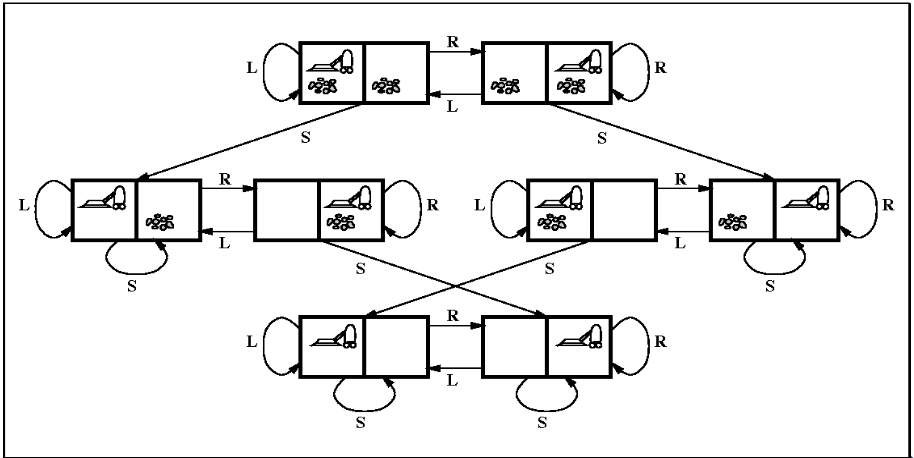


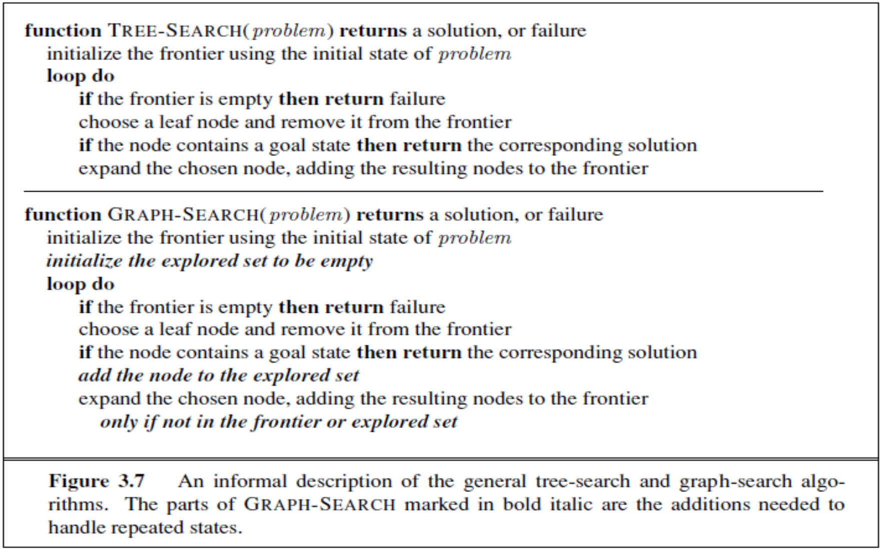
Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Table of Contents

Searching for Solutions

- We can use a **search algorithms** to find a solution to a search problem.
- In particular, we can find **a path (i.e., sequence of actions) from the start state to the goal state** in the state space graph.
- Search algorithms can be **uninformed search methods** or **informed search methods.**
- A solution is an **action sequence**, so search algorithms work by considering various possible action sequences.
- The SEARCH TREE possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions, and the nodes correspond to states in the state space of the problem.

General tree-search and graph-search algorithms

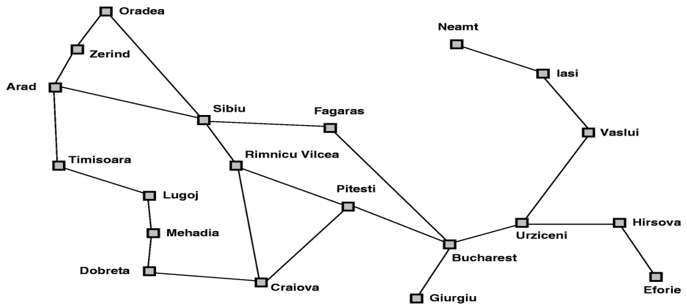


```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      add the node to the explored set
      expand the chosen node, adding the resulting nodes to the frontier
         only if not in the frontier or explored set
```

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

Route-Finding in Romania



Partial search trees for finding a route

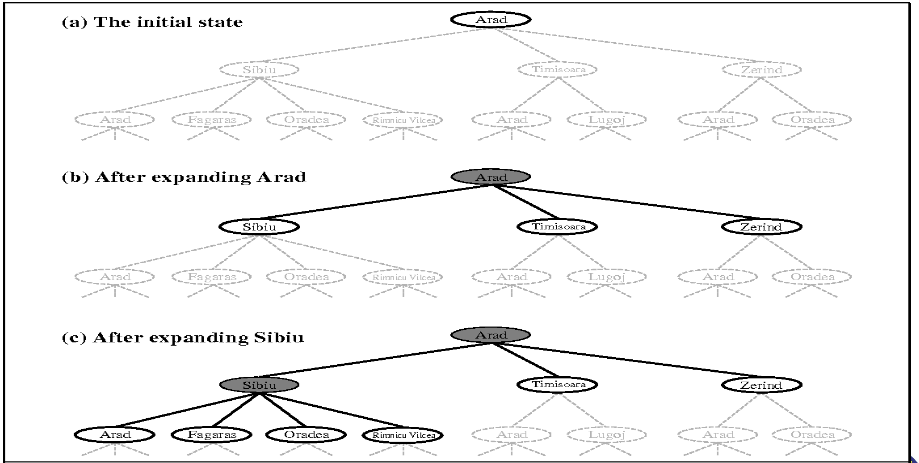


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

Searching for Solutions-cont

- By expanding the current state; that is, **applying each legal action to the current state, thereby generating a new set of states**.
- After expanding Arad, add three branches from the parent node, leading to three new child nodes
- After expanding Sibiu these six nodes is a leaf node, The set of all leaf nodes available for expansion at any given point is called the **frontier**. (Many authors call it the open list)
- The process of expanding nodes on the frontier continues until either a solution is found or there are no more states to expand.

Searching for Solutions-cont

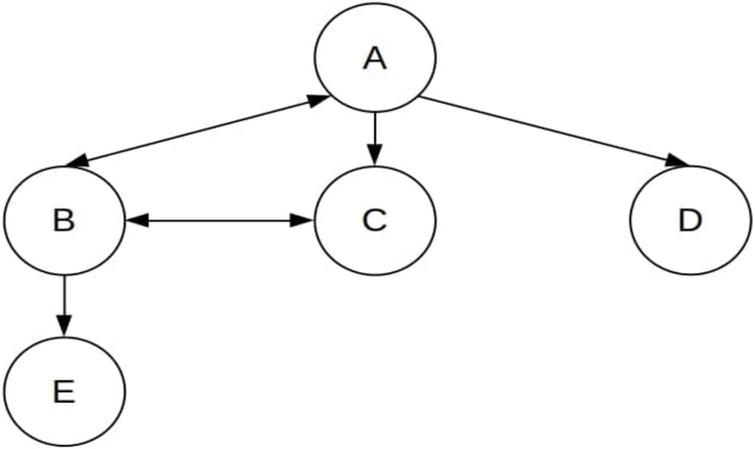
- The way to avoid exploring redundant paths is to remember, TREE-SEARCH algorithm with a data structure called the explored set (also known as the closed list), which remembers every expanded node(visited). The new algorithm, called GRAPH-SEARCH.
- Newly generated nodes that match previously generated nodes—ones in the explored set or the frontier—can be discarded instead of being added to the frontier.
- Search algorithms all share this basic structure; they vary primarily according to how they choose which state to expand next—the so-called search strategy.
- The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy.

Search strategy and Data Structure

- The appropriate data structure for the search strategy is a queue. Three common variants are:
  - the first-in, first-out or **FIFO queue, which pops the oldest element of the queue;**
  - the last-in, first-out or **LIFO queue (also known as a stack), which pops the newest element of the queue;**
  - the **priority queue**, which pops the element of the queue with the highest priority according to some ordering function.

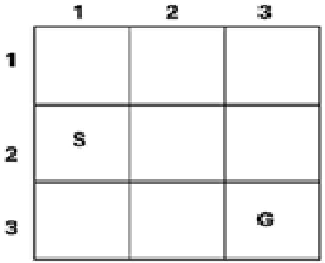
Exercise: Apply Tree Search and Graph Search for DFS and BFS

- Initial state: A
- Goal state: E



Exercise: Apply Graph Search for DFS, BFS

- Initial state: S
- Goal state: G



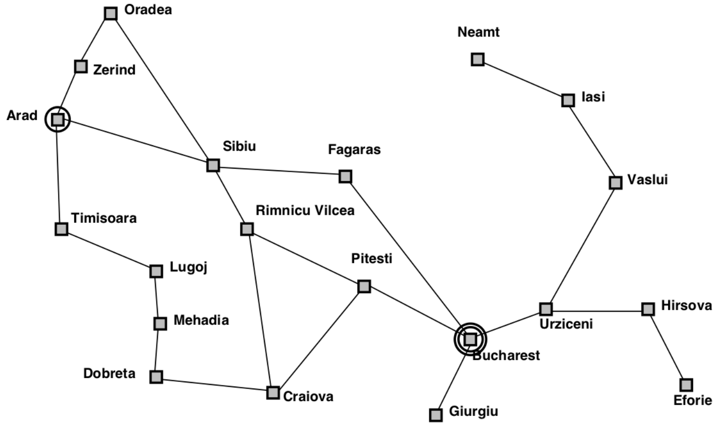
Measuring problem-solving performance

- **completeness:** is the strategy guaranteed to find a solution when there is one?
- **optimality:** does the search strategy find the highest quality solution when there are multiple solutions?
- **time complexity:** how long does it take to find a solution?
- **space complexity:** how much memory is required to perform the search?

Table of Contents

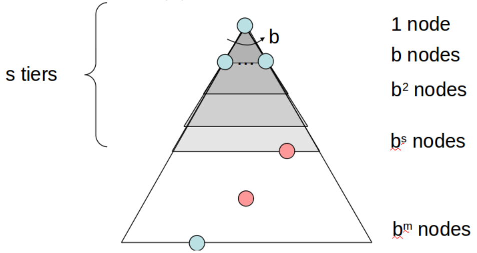
Breadth-first search

Strategy: expand shallowest node first
Implementation: The frontier is a queue of FIFO



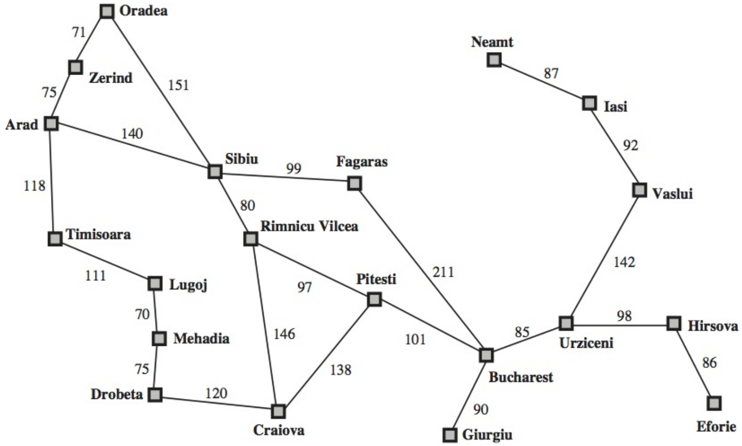
Properties of Breadth-first search

Complete: Guaranteed to find a solution if one exists?
Yes (if b is finite)
Optimal: Guaranteed to find the least cost path?
Yes (if cost $=1$ per step); not optimal in general
Time: $1+b+b^2+b^3+\ldots+b^d = O(b^d)$, i.e., exponential in d
Space: $O(b^d)$ (keeps every node in memory)
Space is the big problem; can easily generate nodes at 1MB/sec so 24hrs = 86GB.



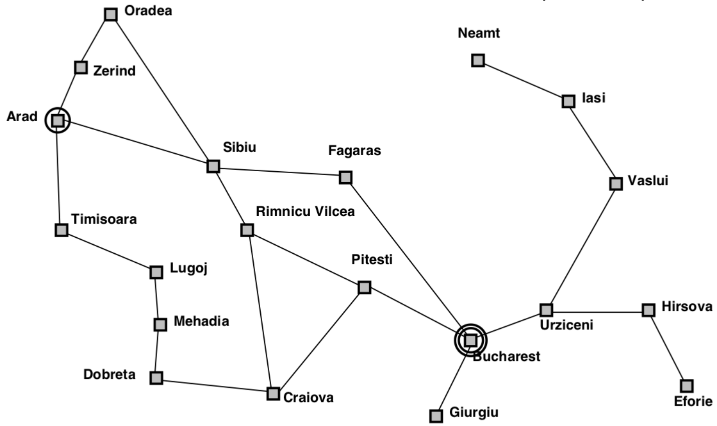
Uniform-cost search

- Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost g(n)



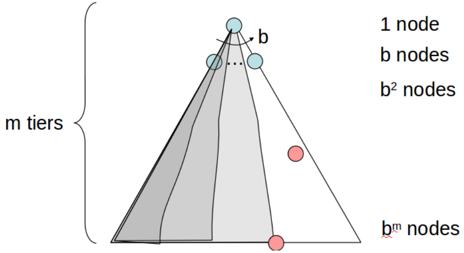
Depth-first search

Strategy: expand deepest node first
Implementation: Fringe is a LIFO queue (a stack)



Properties of Depth-first search

Complete: No. fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces
Optimal: No
Time: $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first
Space: $O(bm)$, i.e., linear space!



Depth-limited search

- depth-first search with depth limit L
- **Implementation**: Nodes at depth L have no successors

Iterative deepening search

- Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search, which finds the best depth limit.
- It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found

Bidirectional search

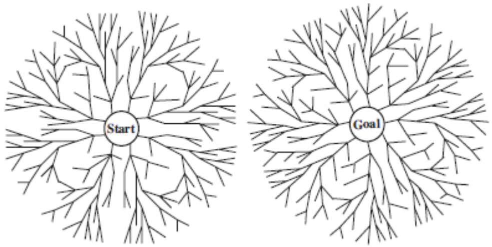
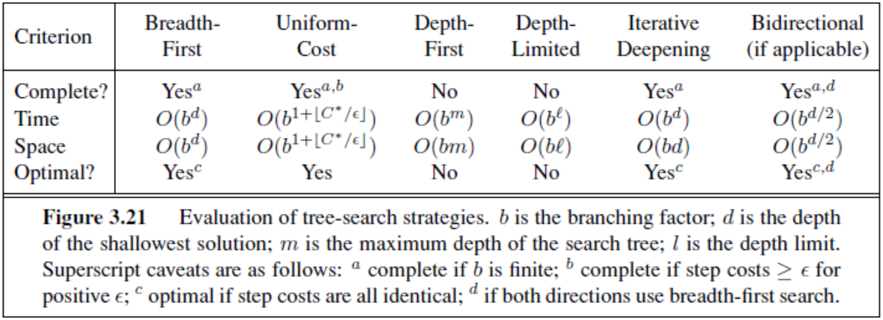


Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

Table of Contents

Comparing uninformed search



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Table of Contents

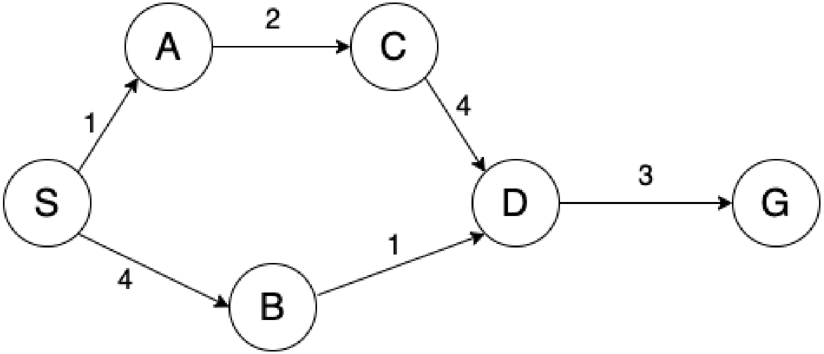
Exercise 1

- You are tasked with designing a robot navigation system where the robot can move up, down, left, or right, if there are no walls blocking its way. The goal is to reach the bottom right corner of the room. The robot can only see its immediate next squares, but not anything else. Blocks can appear randomly anywhere at the room, at any time. There is also an opposing robot whose objective is to damage your robot. Determine whether this agent's environment is single- or multi-agent, fully or partially observable, episodic or sequential, static or dynamic, and deterministic or stochastic.

Exercise 2

- Suppose that you are designing an artificially intelligent wall painting robot. The wall is divided into 144 squares in a 12 x 12 grid design. The robot can paint a single square in a singular action. It then moves the hand to the next square and paints. The job is finished when the entire wall is painted. Give a formal description of this problem as a search problem. What is the size of the state space?

Exercise 3



- Simulate BFS,DFS,UCS,DLS using the graph algorithm. Show the expand tree, explored set(visited), frontier, and path cost

Exercise 4

- Suppose that you have to design a vacuum-cleaner robot, which will operate in an apartment with 3 rooms in a row. It has the following allowed actions with the associated costs: move left (L : 5), moveright (R : 20), and clean (C : 25). Initially, only the middle room is clean, the robot is there, and the other rooms are dirty. The goal of the agent is to keep all the rooms clean. Representing the state space graph of the problem.
- Determine the plan (solution path) for the robot using the BFS, DFS, Uniform-cost search algorithm.

Table of Contents

References

- Stuart Russell and Peter Norvig. 2021. Artificial Intelligence: A Modern Approach (4th ed.). Pearson Education Inc., Boston, MA.
- Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.
- https://www.cs.cmu.edu/ 15281-s23/coursenotes/search/index.html
- https://www.baeldung.com/cs/graph-search-vs-tree-like-search