

Searching

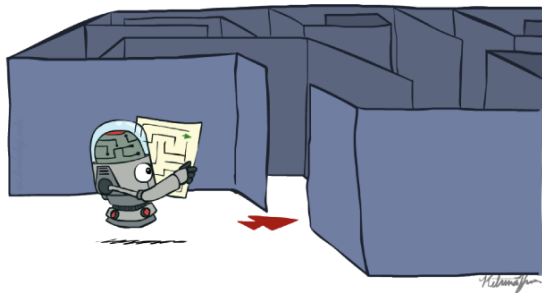
Artificial Intelligence @ Allegheny College

Janyl Jumadinova

September 6–10, 2021

Ref: “Artificial Intelligence: A Modern Approach” textbook

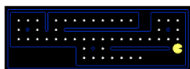
Agents that Plan Ahead



Problem Solving via Search: building goal-based agents that can plan ahead to solve problems.

- Model world with state space.
- Setting up state spaces.

Planning Agents



Planning agents:

Ask “what if?”

Decisions based on (hypothesized) consequences of actions.

Must have a model of how the world evolves in response to actions.

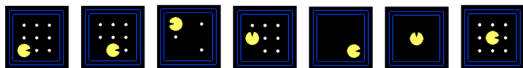
Must formulate a goal (test).

Consider how the world **WOULD BE**.

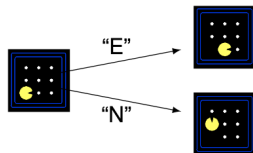
Search Problems

A **search problem** consists of:

A state space



A successor function
(with actions, costs)

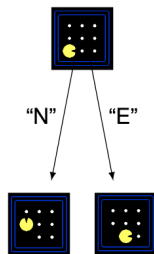


A start state and a goal test

A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Search Trees

This is now / start



Possible futures.

A search tree:

A “what if” tree of plans and their outcomes

The start state is the root node

Children correspond to successors

Nodes show states, but correspond to PLANS that achieve those states

For most problems, we can never actually build the whole tree

General-purpose Search Methods

Uninformed Search Algorithms

Search algorithms which explore the search space without having any information about the problem other than its definition.

General-purpose Search Methods

Uninformed Search Algorithms

Search algorithms which explore the search space without having any information about the problem other than its definition.

Informed Search Algorithms

Search algorithms which leverage any information (heuristics, path cost) on the problem to search through the search space to find the solution efficiently.

Uninformed Search Methods

- Breadth-First Search
- Depth-First Search
- Depth Limited Search
- Iterative Deepening Search

Uninformed Search Methods

- Breadth-First Search
- Depth-First Search
- Depth Limited Search
- Iterative Deepening Search

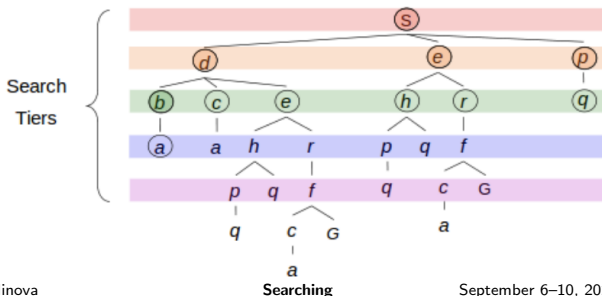
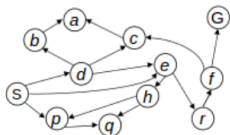
Fringe: data structure used to store all the possible states (nodes) that you can go from the current states.

Breadth-First Search

Explores all of the neighbor nodes at the current depth prior to moving on to the nodes at the next depth level.

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



Breadth-First Search

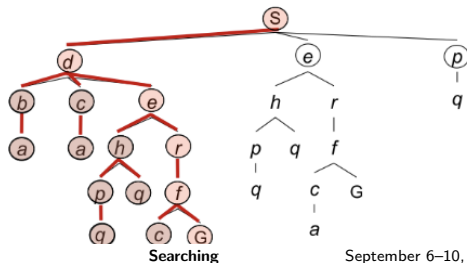
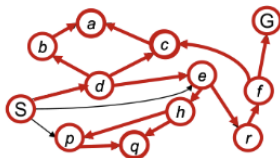
```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure  
  node  $\leftarrow$  NODE(problem.INITIAL)  
  if problem.IS-GOAL(node.STATE) then return node  
  frontier  $\leftarrow$  a FIFO queue, with node as an element  
  reached  $\leftarrow$  {problem.INITIAL}  
  while not IS-EMPTY(frontier) do  
    node  $\leftarrow$  POP(frontier)  
    for each child in EXPAND(problem, node) do  
      s  $\leftarrow$  child.STATE  
      if problem.IS-GOAL(s) then return child  
      if s is not in reached then  
        add s to reached  
        add child to frontier  
  return failure
```

Depth-First Search

Starts at the root node and explores as far as possible along each branch before backtracking.

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack



Depth-Limited Search

Idea: DFS with pre-determined limit to avoid the problem of infinite path of DFS.

- If the depth limit is reached, assume there are no successor nodes.
- It is memory efficient.
- It can be incomplete.

Iterative Deepening Search

Idea: get DFS's space advantage with BFS's time

- Run a DFS with depth limit 1. If no solution ...
- Run a DFS with depth limit 2. If no solution ...
- Run a DFS with depth limit 3 ...

Informed Search Methods

- Best First Search
- A* (“A star”)
- Recursive Best First Search

Best First Search

Best-First Search:

Nodes are selected for expansion based on an evaluation function, $f(n)$.
Traditionally, f is a cost measure.

Best First Search

Best-First Search:

Nodes are selected for expansion based on an evaluation function, $f(n)$. Traditionally, f is a cost measure.

- Use an evaluation function, $f(n)$, for each node (estimate of "desirability").
- Always choose the node from fringe that has the lowest f value.

Heuristic Search

Idea: Informed search by using problem-specific knowledge.

Heuristic Search

Idea: Informed search by using problem-specific knowledge.

Heuristic:

Problem specific knowledge that (tries to) lead the search algorithm faster towards a goal state. Often implemented via heuristic function $h(n)$.

- **Heuristic search** is an attempt to search the most promising paths first. Uses heuristics, or rules of thumb, to find the best node to expand next.

Example: Romania

On holiday in Romania; currently in Arad.

Flight leaves tomorrow from Bucharest

Formulate goal: be in Bucharest

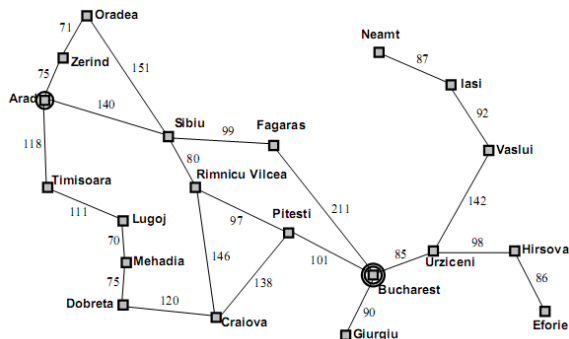
Formulate problem:

states: various cities

actions: drive between cities

Find solution: sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

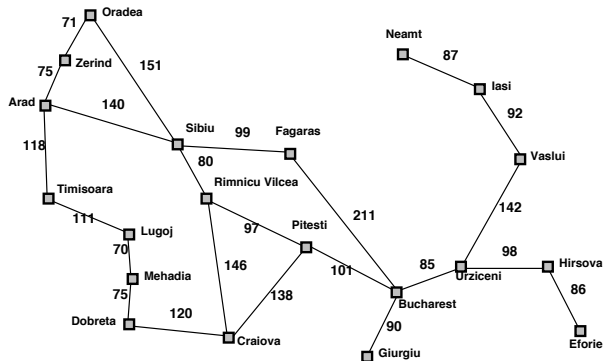
Example: Romania



Can use Tree Search Algorithms (BFS, DFS)

Special cases: greedy search, A^* search

Romania with step costs in km



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search

Idea:

avoid expanding paths that are already expensive

A* search

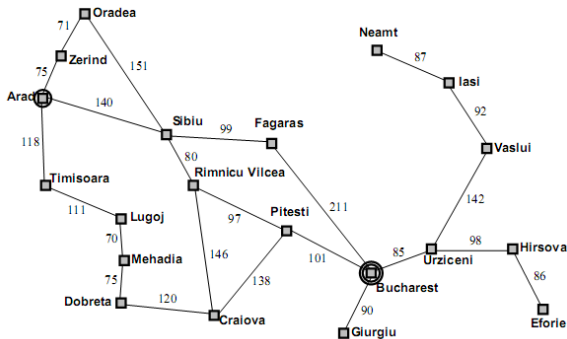
Idea:

avoid expanding paths that are already expensive

- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost to goal from n
- $f(n)$ = estimated total cost of path through n to goal

Romania with step costs

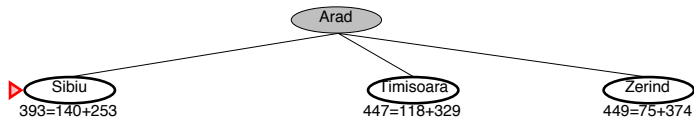
Example: Romania



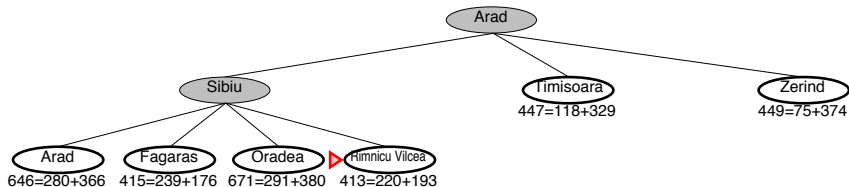
A* Search

Arad
366=0+366

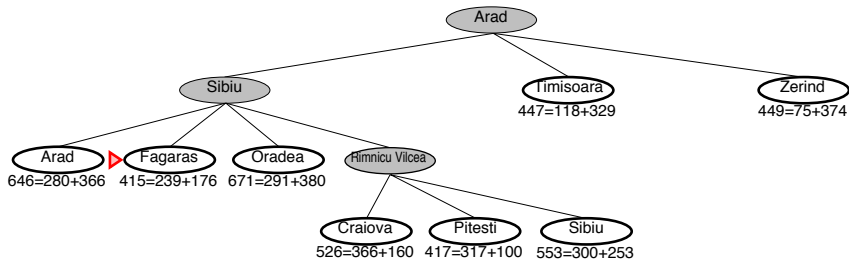
A* Search



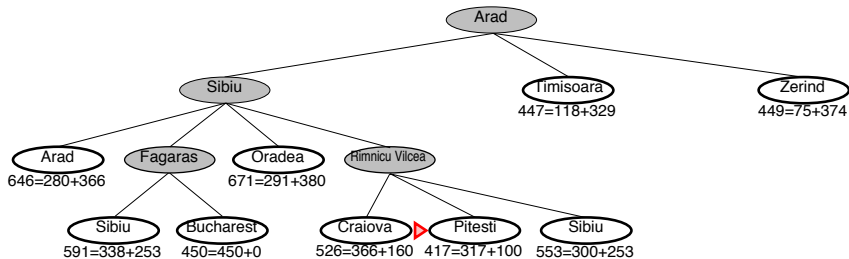
A* Search



A* Search



A* Search



A* Search

