CS 182 Lecture 2: Problem Representation and Uninformed Search

Professors: Ariel Procaccia and **Stephanie Gil**

Email: sgil@seas.harvard.edu

Prof. Gil Office hours: Wednesdays 2:30-3:30p

Last Time:

Types of Problems and Environments:

- Fully vs. partially observable
- Deterministic vs. stochastic
- Episodic vs. sequential (i.e. what role does *history* play)
- Static vs. dynamic
- Discrete vs. continuous
- Single-agent vs. multiagent

Classical Planning Assumptions

- Finite state environment – set of states and actions is finite
  - Changes occur only in response to actions
  - i.e. if the actor does not act, the current state remains unchanged.
  - Does not include the possibility of actions by other actors or exogeneous events

- Determinism – no uncertainty
  - We assume that we can predict with certainty what state will be produced if action a is performed on state s
  - Excludes accidents, execution error, or nondeterministic actions (such as rolling dice)

Pros and Cons

Pros

- Simple models to build
- Simple to reason with these models

Cons

- Does not always capture the full reality
- May introduce costly errors

This Time

- Problem representation
- Uninformed search
- Reference readings: Chapter 3.3-3.4 in Russel and Norvig text
- HW 0 is posted (due 9/12 @ 11:59pm)

Problem Formulation

- The process of deciding what actions and states to consider (i.e. that are relevant) given a goal

- We must know:
  - What are the states
  - What are the start state and goal states
  - What are the actions
  - Which actions lead to which states (referred to as the successor function)

Elements of a Well-Defined Problem

- What is a problem?

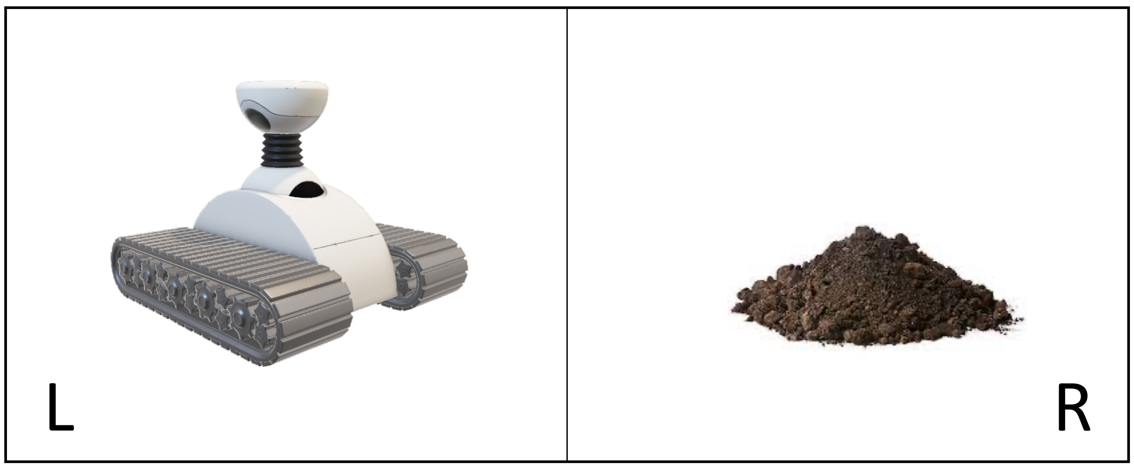
Problem

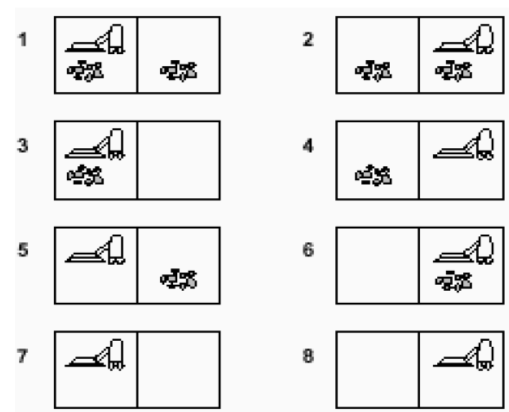
- **Initial state:** where is the agent starting?
- **Actions:** <action, successor>
- **Goal test:** is a given state the goal state?
- **Path cost:** numeric cost of each path. Step cost is c(x,a,y) for x initial state, action a, and next state y

- Solution quality?
  - Measured by the cost over the path
  - An *optimal* solution has the lowest path cost among all solutions

Example Formulation: Vacuum World

- <u>States:</u> agent is in one of two locations (L,R) which can be dirty or clean. How many states are there?



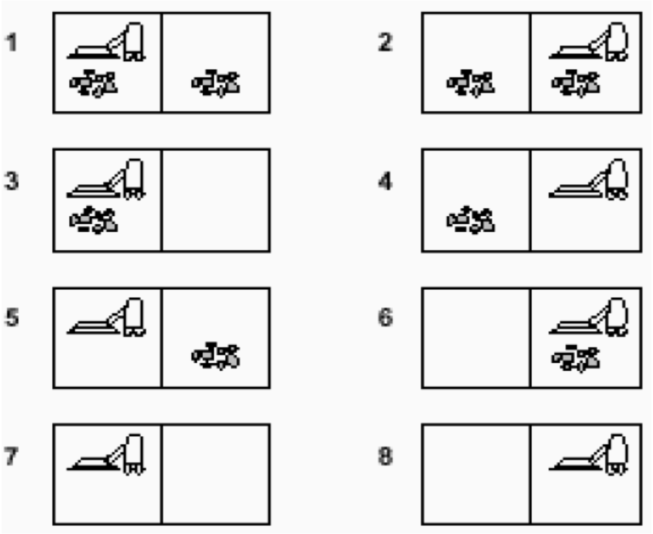


Russel and Norvig text

- <u>Actions:</u> Left/Right/Suck
- <u>Goal:</u> Clean up all the dirt. How many states are goal states?

State Space Graphs

- Must include:
  - **Nodes** – abstracted world configurations
  - **Arcs** – represent successors (action results)
  - **Start state** – beginning state
  - **Goal test** – set of goal nodes



Q1: Draw the state space graph for the Vacuum World problem?

Example Formulation: Vacuum World

- Successor Function: Generates the legal states resulting from trying the 3 actions <left,right,suck>
- Goal Test: Are all squares clean?
- Path Cost: Each step costs 1
- State Space:

State Space Graphs

A few important notes about state space graphs:

- In the state space graph each state occurs only once

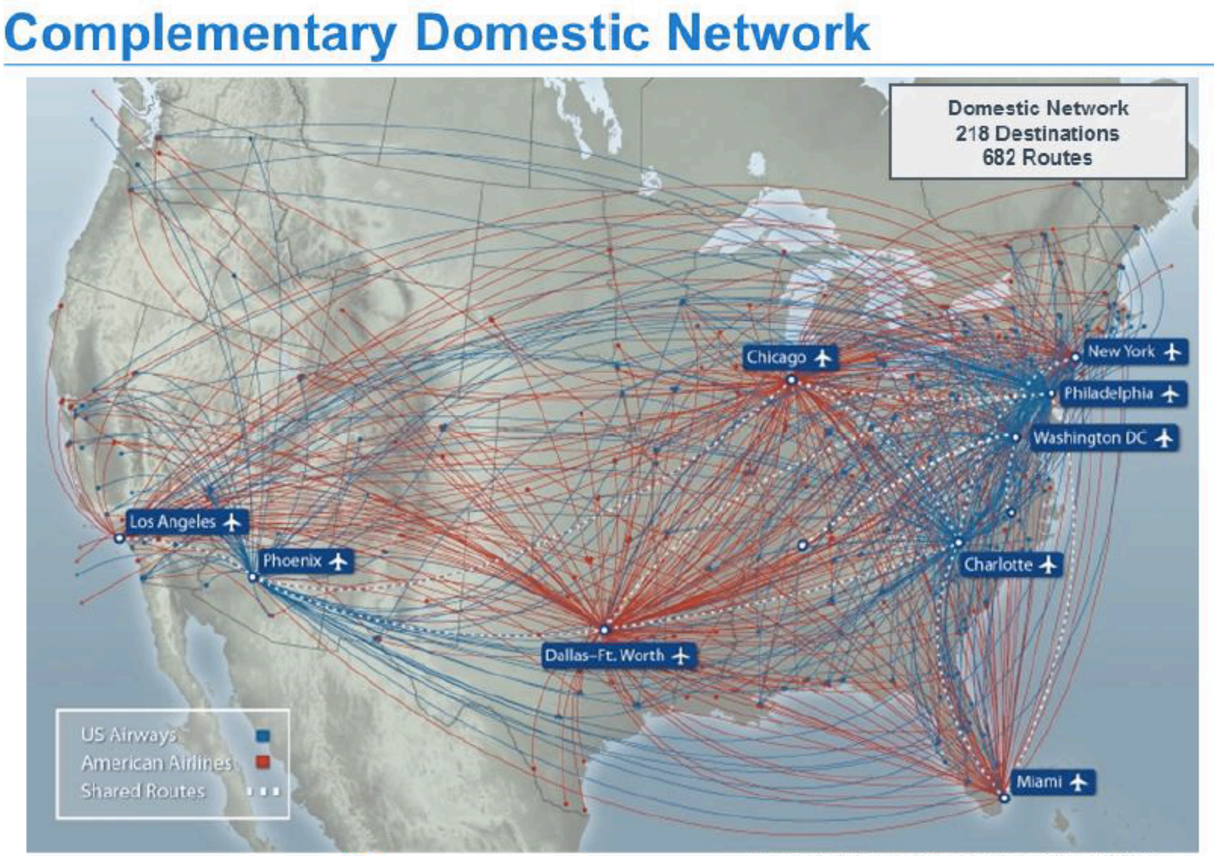
- Does this mean that there is only one path to get to each state?

- In practice, it is too big to draw, build, or fit into memory
  - Example: size of the state space for tic-tac-toe?

➢ Much of this class is about solving problems without having to explore the entire state space

Real World Problems

Route-finding problem (e.g. routing in computer networks, military operations, airline travel planning system)



Real World Problems

Route-finding problem (e.g. routing in computer networks, military operations, airline travel planning system)

- States: Locations (and current time)
- Initial state: Problem dependent
- Successor Function: Returns the state resulting from taking any scheduled flight
- Goal Test: Are we are the destination by some pre-specified time?
- Path Cost: Can be monetary, wait time, flight time, etc.

Real World Problems (cont.)

- **Touring problem** – closely related to route-finding but has an important difference: *visit each city only once*.
  - State space must include not only the current city but also previously visited cities
  - How does the state space compare to that of route-finding?

- **Traveling Salesman problem** – a touring problem where the aim is to find the shortest tour

- **Robot Navigation** – Generalization of the route-finding problem. Differences:
  - Robot can move in continuous space with an infinite set of possible actions and states
  - Robots must deal with errors in sensor readings and motor controls

Problem Formulation then What?

Now that we have the problem formulated correctly, what can we do next?

How do we generate a solution?

Search.

Automating the Solution Generation Part… Search

- Search strategy: choice of which state to expand (depends on choice of states compatible with the successor function)
- Measures of search performance
  - Completeness – is the algorithm guaranteed to find a solution when there is one?
  - Optimality – Does the strategy find the optimal solution?
  - Time Complexity – How long does it take to find a solution?
  - Space Complexity – How much memory is needed to perform the search?

Search

Repeated action in search: expand the current state (i.e. which states can the current state lead to?) and generate a new set of states

- Search steps:

1) Choose state
2) Goal test
3) Expand state

- Two termination conditions

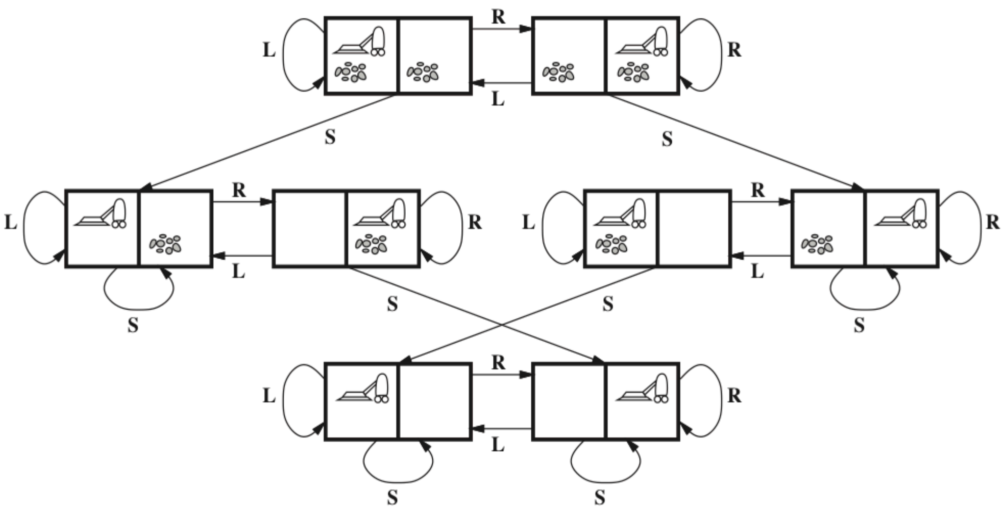
1) Solution found
2) No more states left to be expanded

Search tree

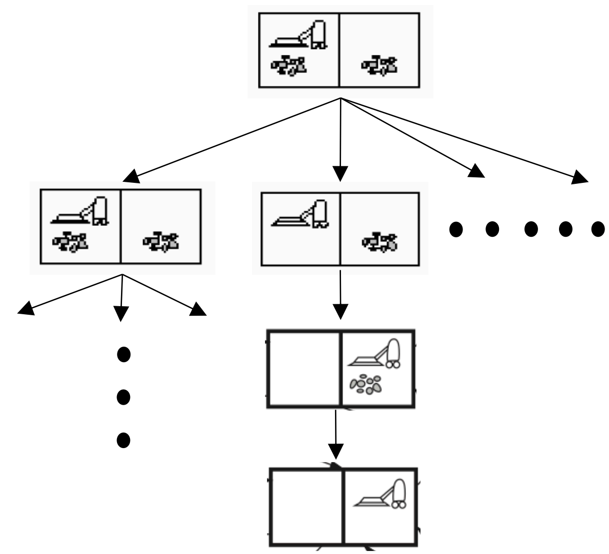
- A few definitions…
  - State
  - Parent node
  - Action
  - Path
  - Path cost $g(n)$
  - Depth d
  - Branching factor b

From a State Graph to a Search Tree

State Space Graph



Search Tree



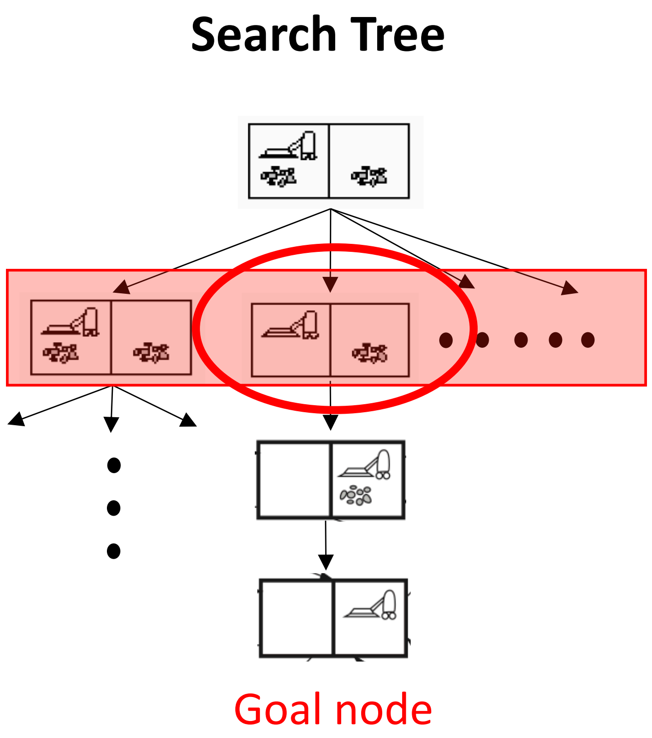
Nodes in a tree correspond to "plans" not states!

Tree Search

function TREE-SEARCH(*problem*, *strategy*)
set of frontier nodes contains the start state of *problem*
loop

- **if** there are no frontier nodes **then return** failure
- choose a frontier node for expansion using *strategy*
- **if** the node contains a goal **then return** the corresponding solution
- **else** expand the node and add the resulting nodes to the set of frontier nodes

Terminology for Tree Search



- Frontier nodes
- Node chosen for expansion (how to choose this node? *Strategy*)
- Goal node

From a State Graph to a Search Tree

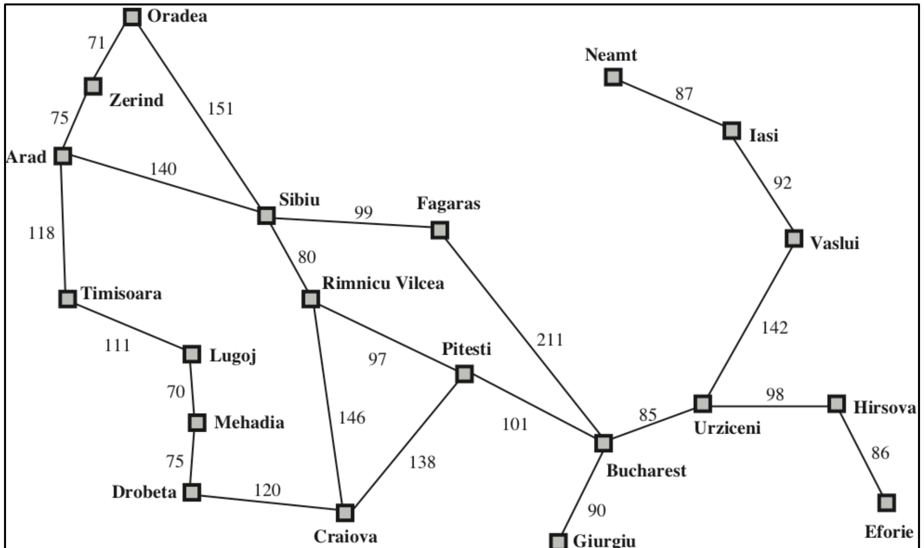


Fig 3.2 Russell and Norvig

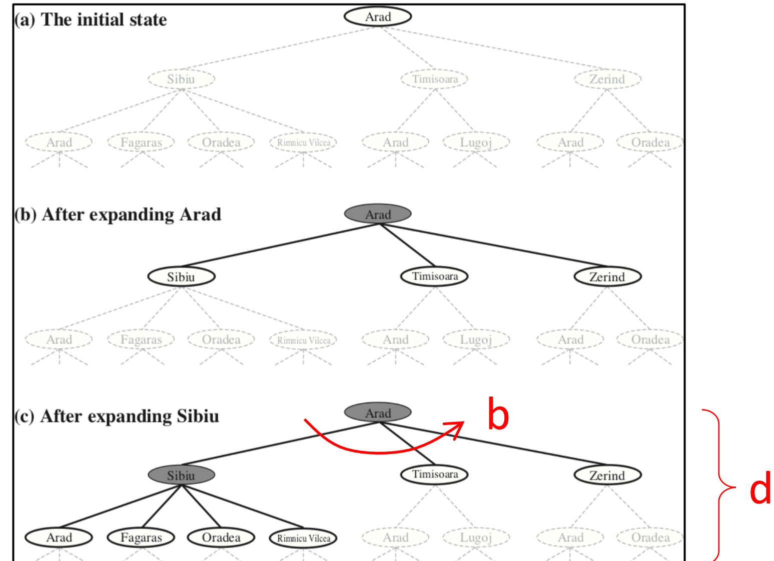


Fig 3.6 Russell and Norvig

- Complexity for uninformed search – described in terms of 3 quantities
  - Branching factor b
  - Depth d
  - Maximum length of any path in the state space m

Uninformed Search Strategies

When you have no idea of whether one “non-goal state” is better than another

Q2 (Polls Everywhere poll):

- If we have a finite # of states, must the search tree have a finite number of nodes?
- Is the # of nodes in the search tree the same as the # of states?

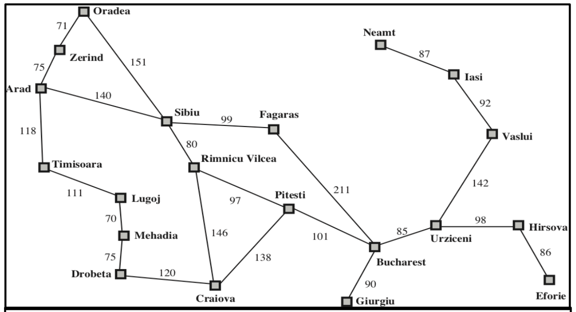
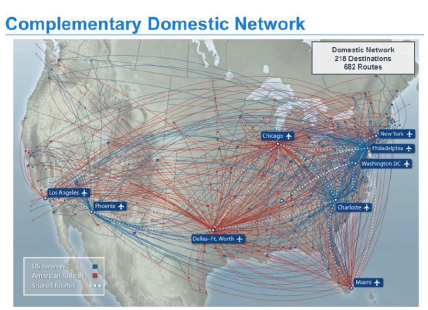
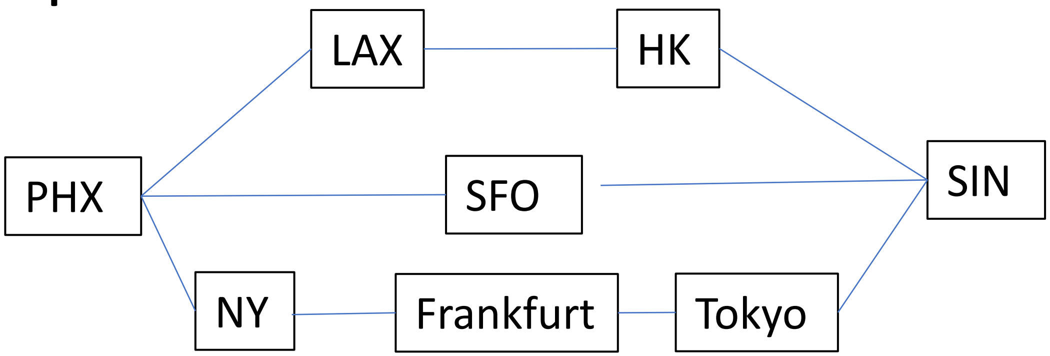


Fig 3.2 Russell and Norvig

Breadth-first Search

- FIFO queue
- All nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded
- Is BFS complete?
- Is BFS optimal?
- Airline route example…





Complexity of BFS

- Memory requirements of BFS?
- Time complexity of BFS?

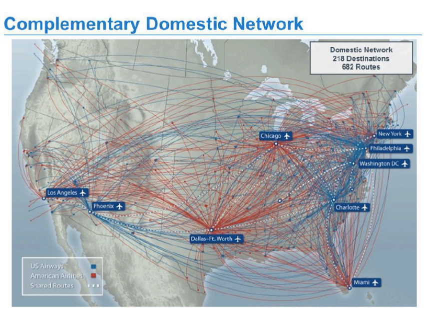
*Exponential complexity search problems cannot be solved by uninformed methods for any but the smallest problem instances

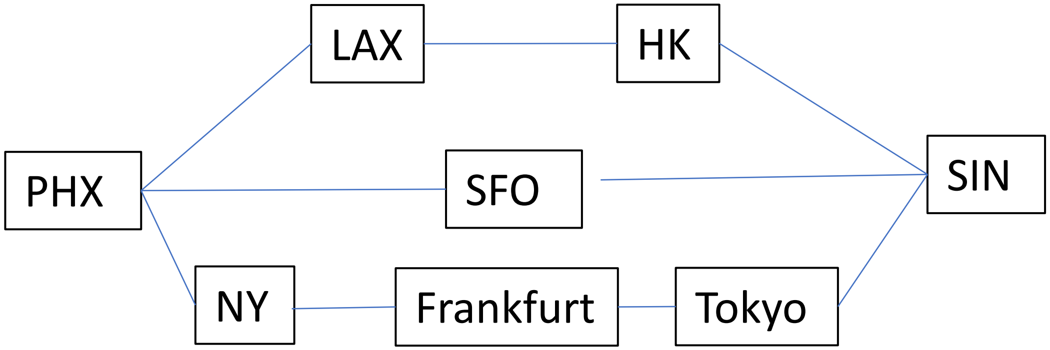
- Depth of 2, 0.11 seconds
- Depth of 4, 11 seconds
- Depth of 8, 31 hours
- Depth of 10, 129 days….

[Russell and Norvig, ch 3]

Depth-first Search

- LIFO queue
- Explore tree down to the root using a rule (such as leftmost branch first) before backing up
- Is DFS complete?
- Is DFS optimal?
- Airline route example…





Depth-first Search

- LIFO queue (a stack)

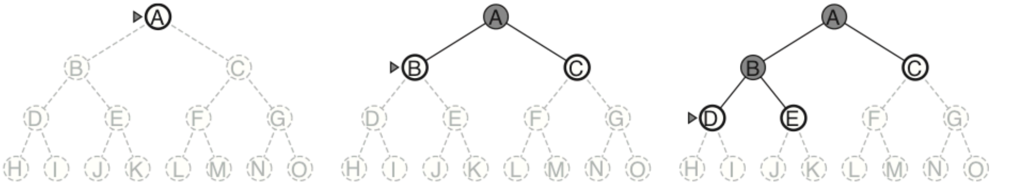


Fig 3.16 Russell and Norvig

Depth-first Search (cont.)

- Stores only a single path from the root to a lead node, along with unexpanded sibling nodes for each node on the path
- What factors does memory depend on?
- Is DFS optimal?
- Is DFS complete?
- What is the worst-case complexity? Better or worse than BFS?

Variations on BFS and DFS

- Depth-limited Search
  - When is this a good idea?
  - Definition of **diameter** of the graph: greatest length amongst the shortest path between two nodes

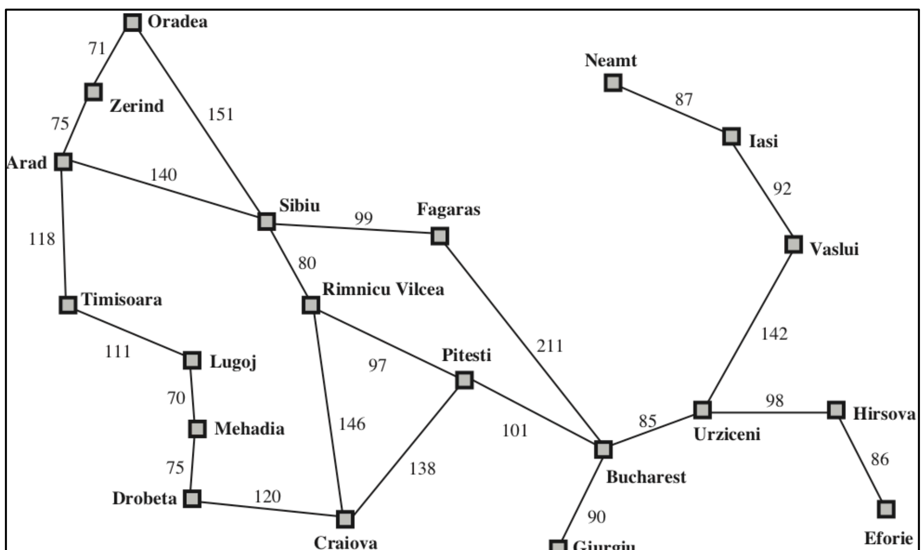


Fig 3.2 Russell and Norvig

Variations of BFS and DFS (cont)

- Iterative Deepening
  - Combines the best of DFS (modest memory req) and BFS (complete for finite b, optimal for some problems)

[Fig 3.19 Russell and Norvig]

Iterative Deepening Search

- Run DFS with depth limit l=1,2,…
- Combines the best properties of BFS and DFS
- What factors does memory depend on?
- Space complexity?
- Is IDS optimal?
- Is IDS complete?
- What is the worst-case complexity? Better or worse than BFS?

Review of Search Strategies

Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Under certain conditions	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	O(b*m)
IDS	Yes	No	$O(b^d)$	O(b*d)

The Problem of Repeated States

- We saw that desirable properties of BFS and DFS depend on finite branching factor and search tree depth

- Different ways this can happen
  - Infinite states space or action space (e.g. continuous problems)
  - Repeated states and cycles

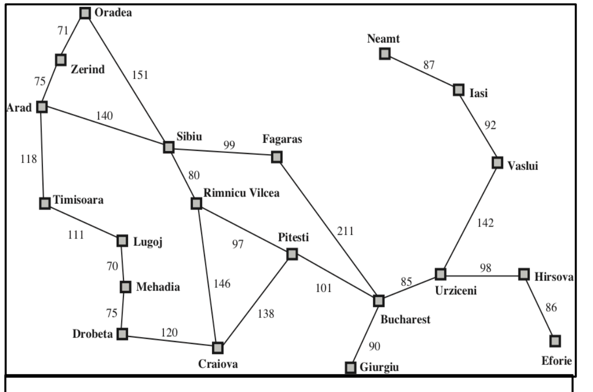


Fig 3.2 Russell and Norvig

How to Avoid Repeated States?

- Problem: expanding states that have already been encountered and expanded before
- For some problems, repeated states are unavoidable
  - Includes problems where actions are reversible
- Idea: prune or avoid repeated states

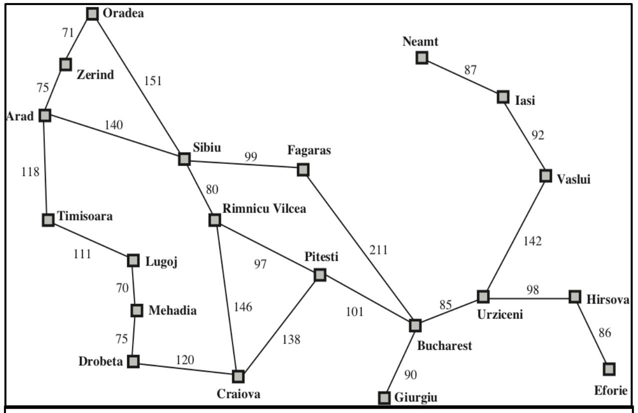
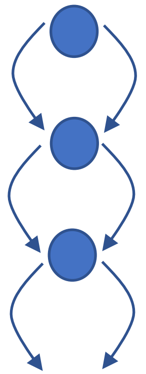


Fig 3.2 Russell and Norvig

For this problem we considered Depth-limited Search with ℓ=9 (the graph diameter) being the maximum path length

Repeated States (cont.)

- Depth-limited search may not always work – we don't always have a good candidate for ℓ
- Two (or more) distinct actions can lead to one distinct state (and not detecting this can lead to an exponential sized graph…)



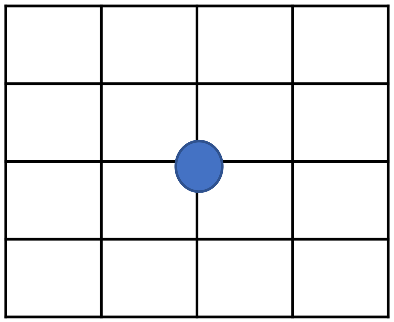


Fig 3.18 Russell and Norvig (2nd edition)

Detection of Repeated States

- Detect repeated states before expanding – if a match is found then the algorithm has discovered two paths to the same state

- Get around this by keeping visited states in memory
  - Points to a fundamental tradeoff between space and time complexity

- Closed set – every expanded node and checks current node to closed list before expanding

- Open set – the fringe of discovered by unexpanded nodes

➢ New algorithm is called graph search instead of tree search

Graph Search

function GRAPH-SEARCH(*problem*, *strategy*)

set of frontier nodes contains the start state of *problem*

loop

- **if** there are no frontier nodes **then return** failure
- choose a frontier node for expansion using *strategy*, and add its state to the explored set
- **if** the node contains a goal **then return** the corresponding solution
- **else** expand the node and add the resulting nodes to the set of frontier nodes, only if not in the explored set of visited states

Next Time…

- Informed search
  - Relevant readings for reference: Ch 3.5, 3.6, 4.1