CMU 15-781

Lecture 2: Uninformed Search

Teacher:
Gianni A. Di Caro

Search Problems

- A search problem has:
  - **States** (configurations)
  - **Start state** and **goal states**
  - **Actions** available to the agent in each state
  - **Transition model:** the state resulting from doing action a in state s *(Successor function)*
  - **Cost model:** step costs $c(s, a, s') \geq 0$, path costs (additive)

Find one (the optimal) sequence of actions (path) from start to a goal state

Example: Pancakes

Discrete Mathematics 27 (1979) 47–57.

BOUNDS FOR SORTING BY PREFIX REVERSAL

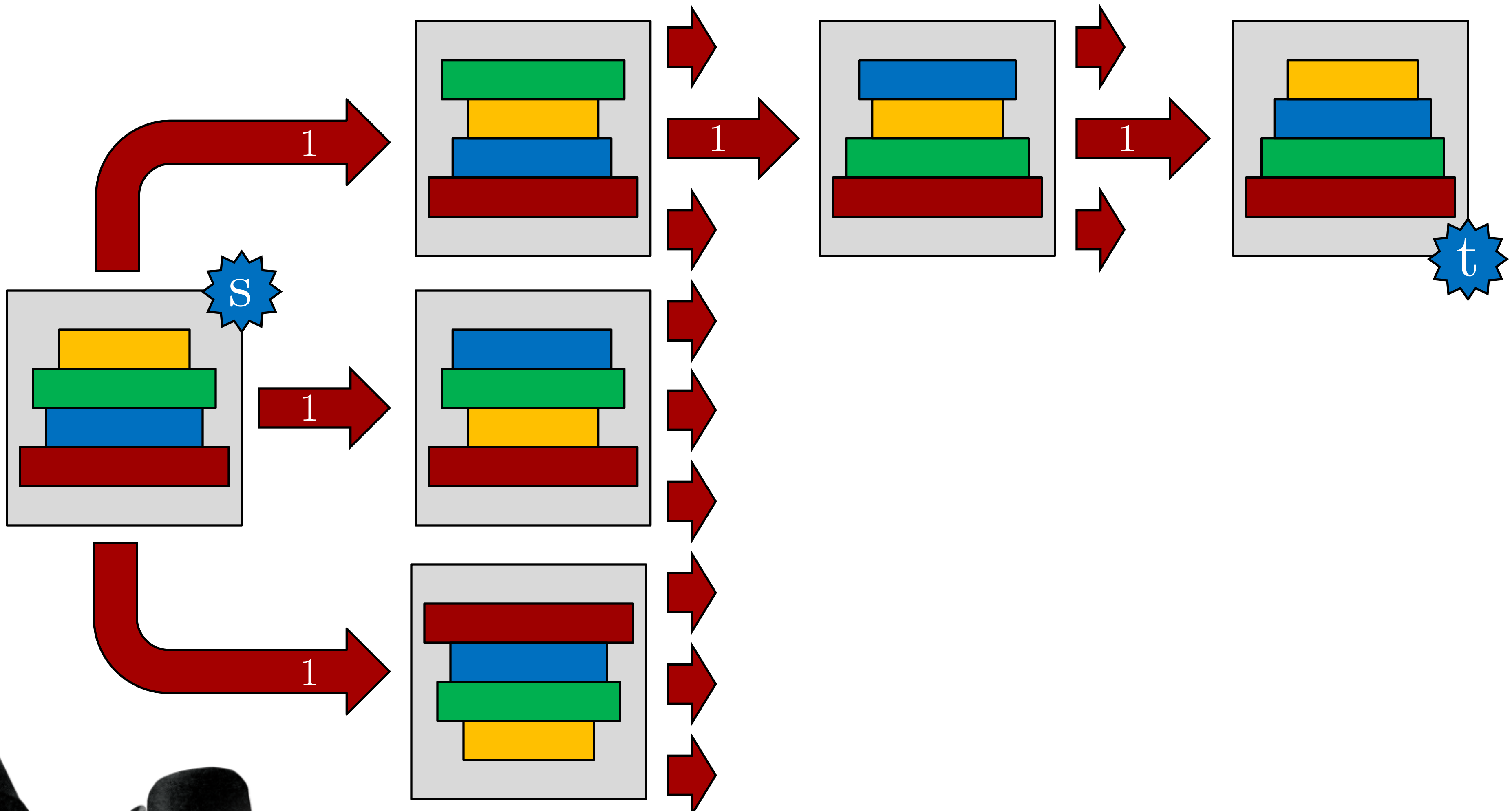
William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

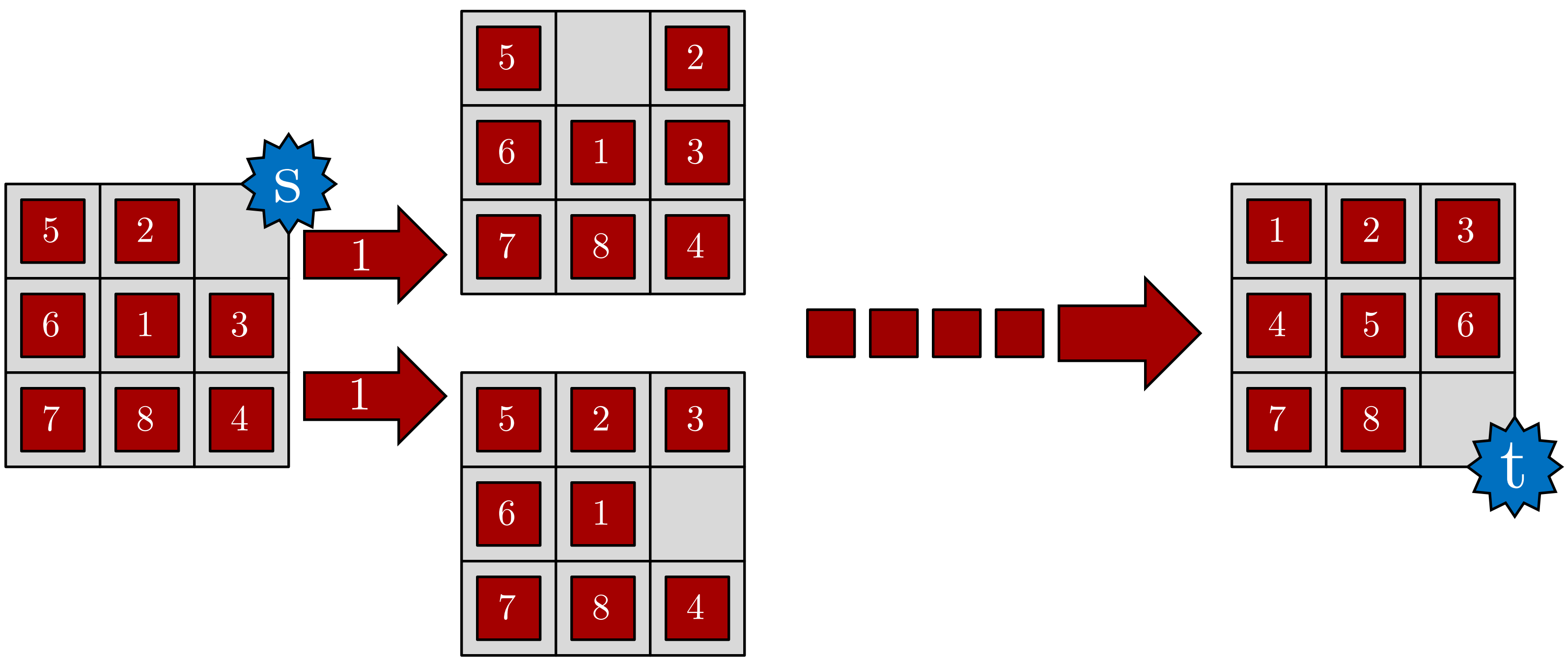
Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.

Example: Pancakes

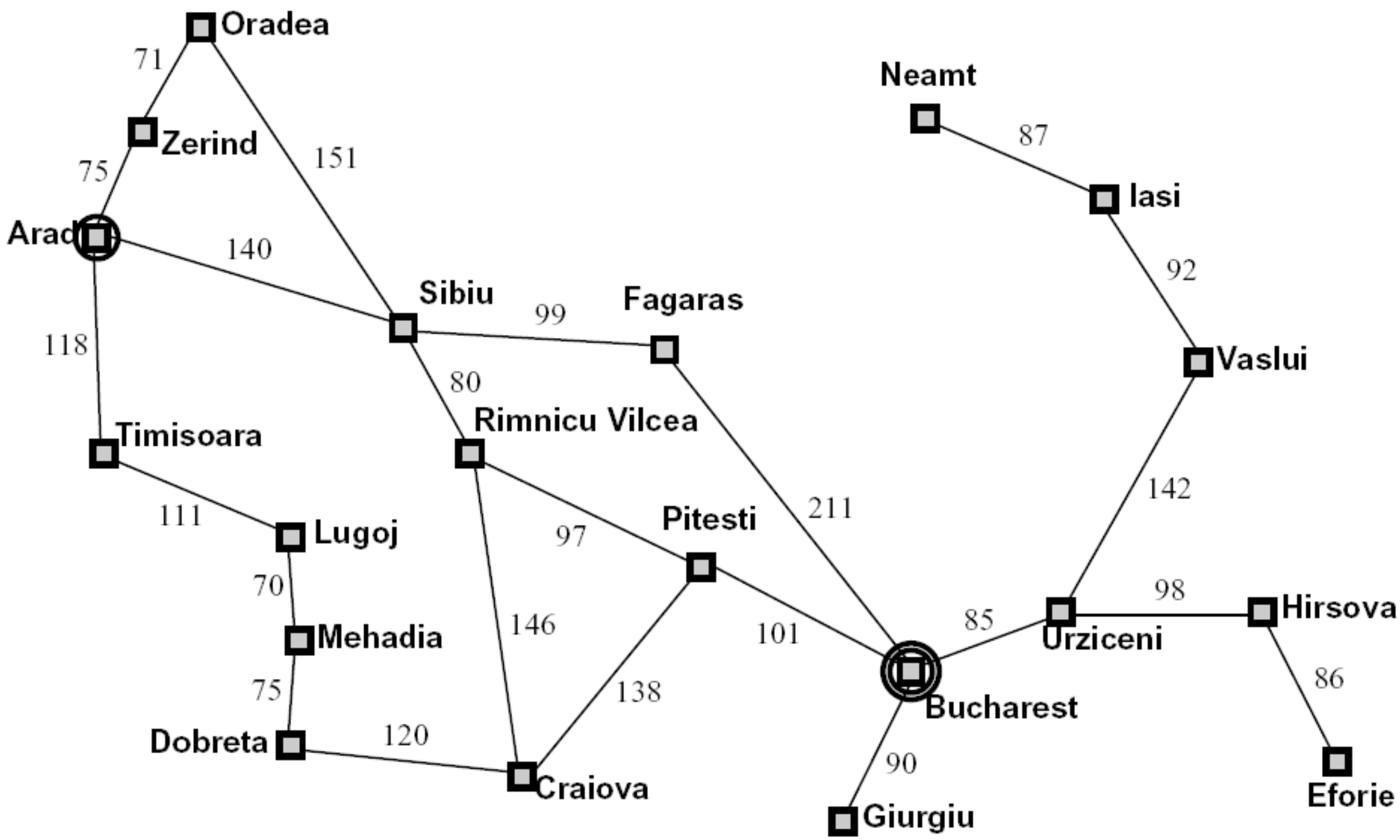


Example: 8-Puzzle

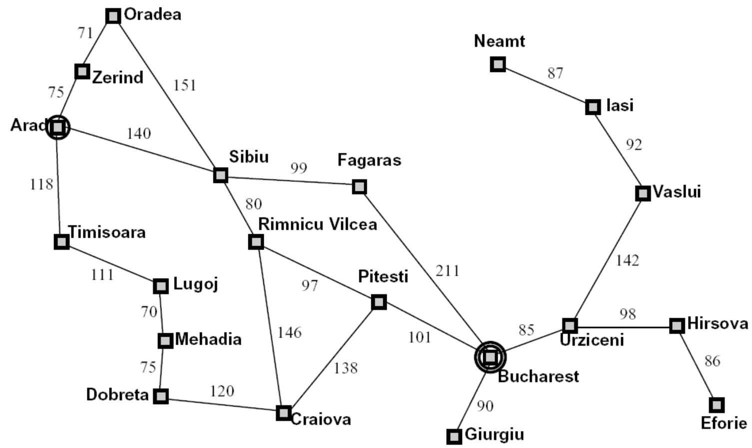


How many states?

EXAMPLE: PATHFINDING

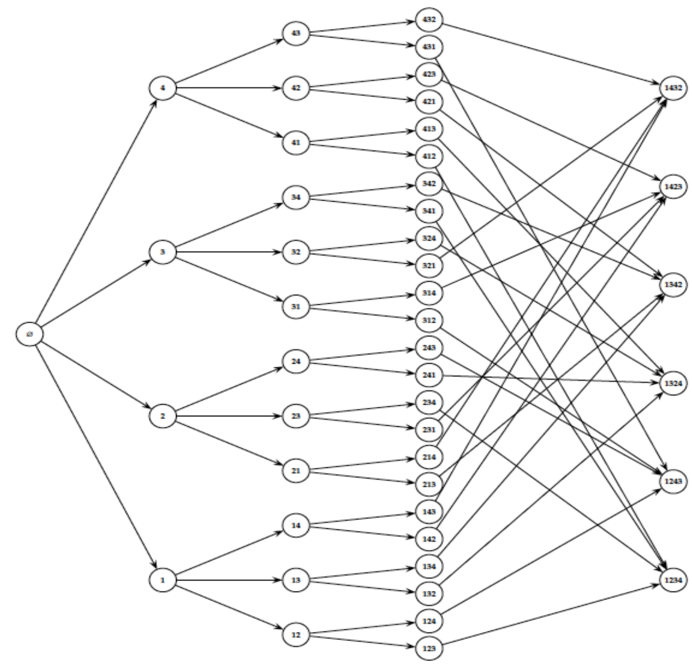


Example: Touring problems



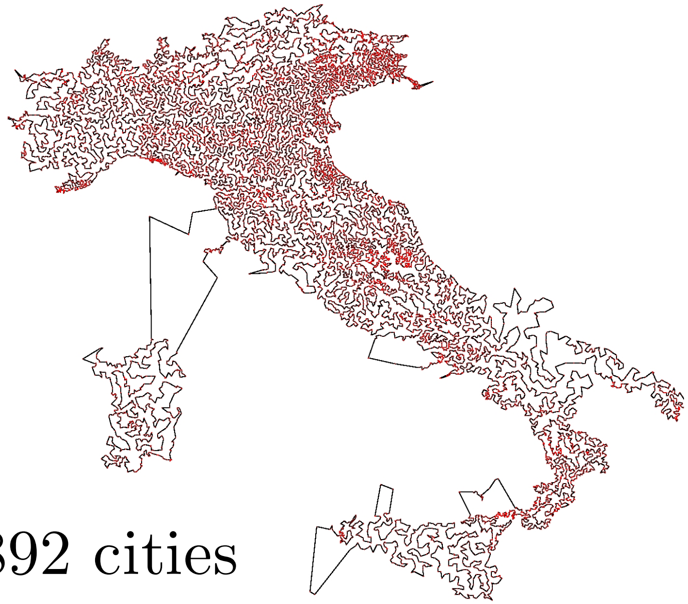
Visit every city at least once, starting and ending in Bucharest

How the state space looks like?

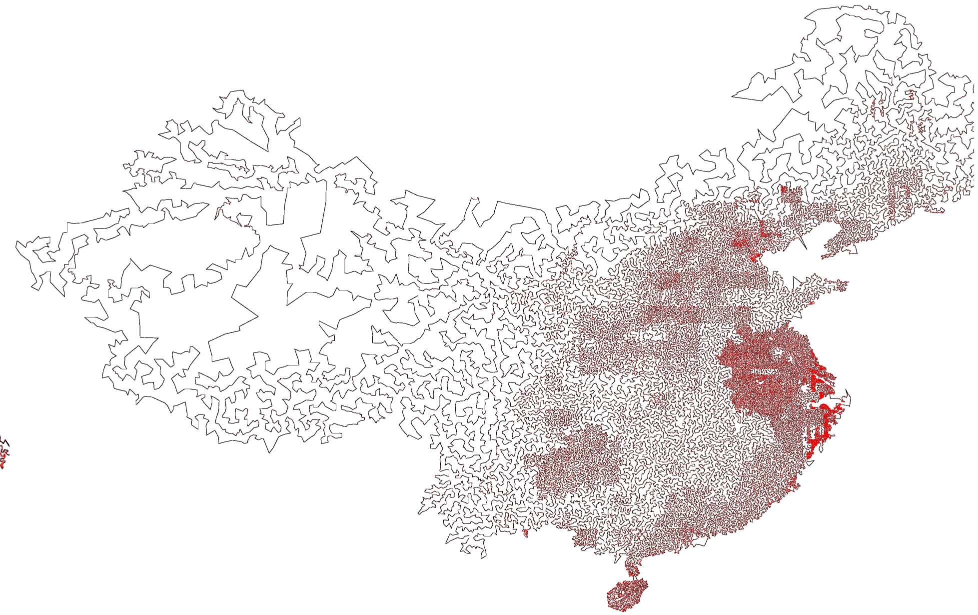


Traveling salesperson problem (TSP): Visit each city exactly once and find the tour of minimum "cost" (costs may be not symmetric...)

Example: Touring problems



16,892 cities

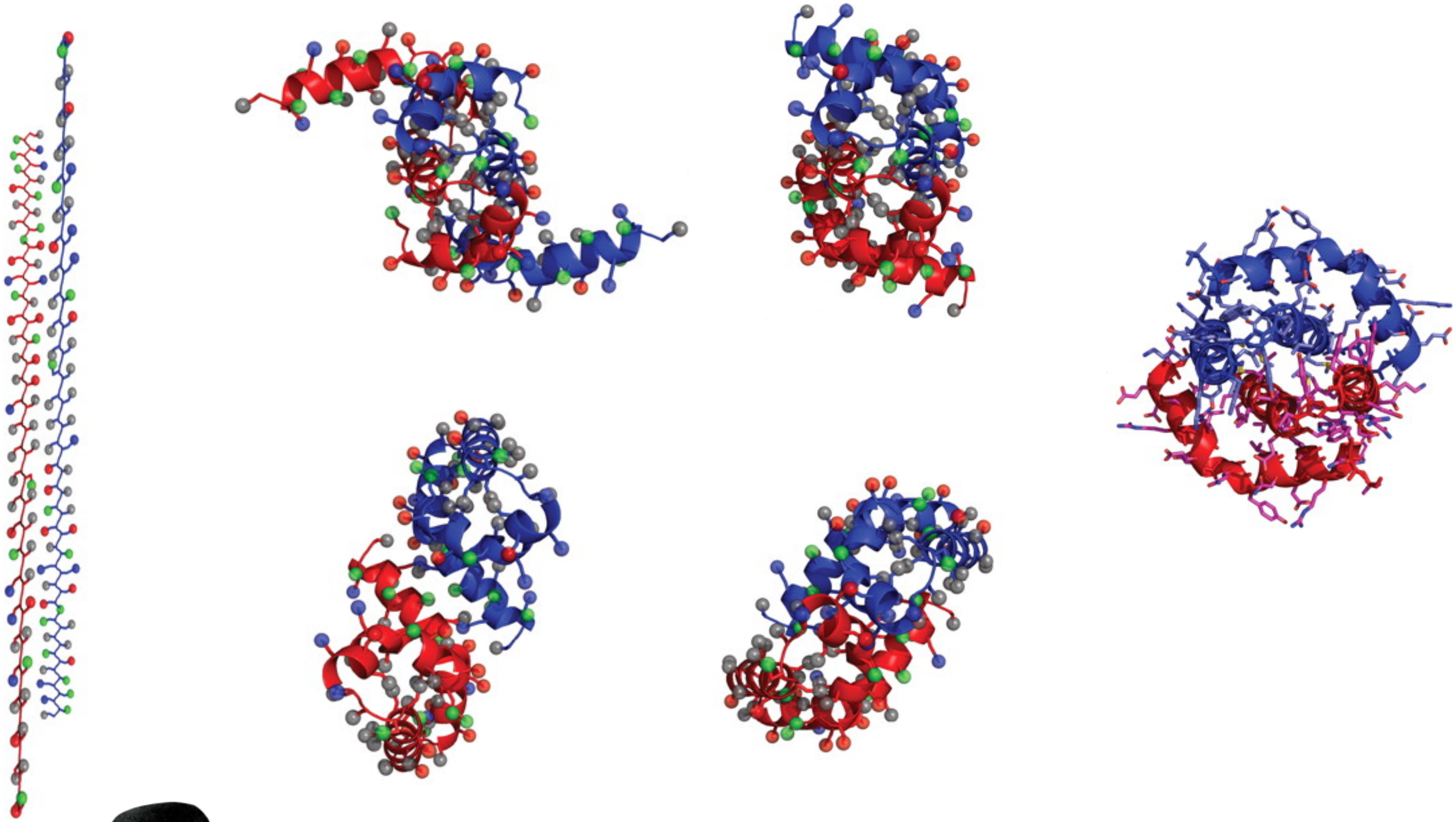


71,009 cities

http://www.math.uwaterloo.ca/tsp/world/countries.html

Trivial to find one feasible solution, (NP-) hard to find the best one

Example: Protein Folding



TREE SEARCH

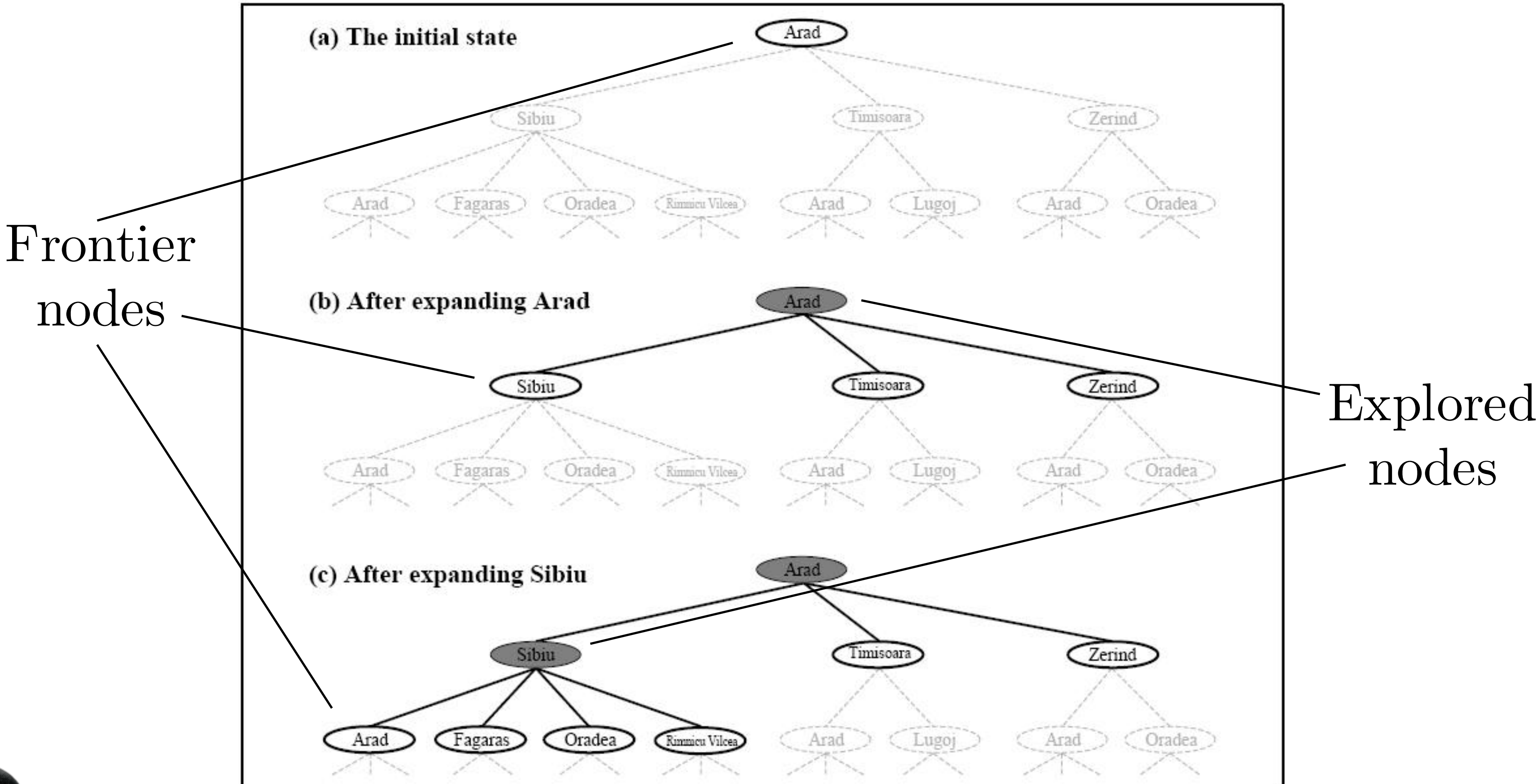
function TREE-SEARCH(problem, strategy)

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

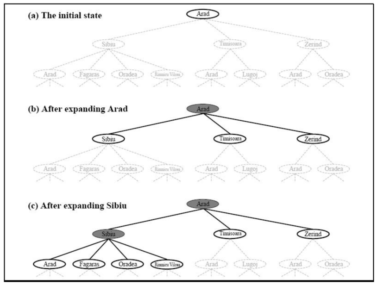
loop

- 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

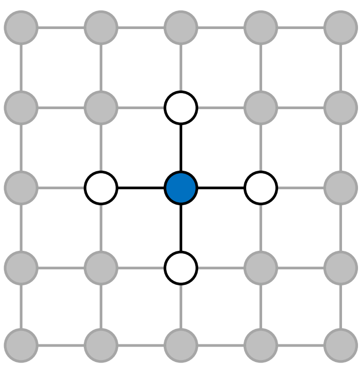
Tree Search

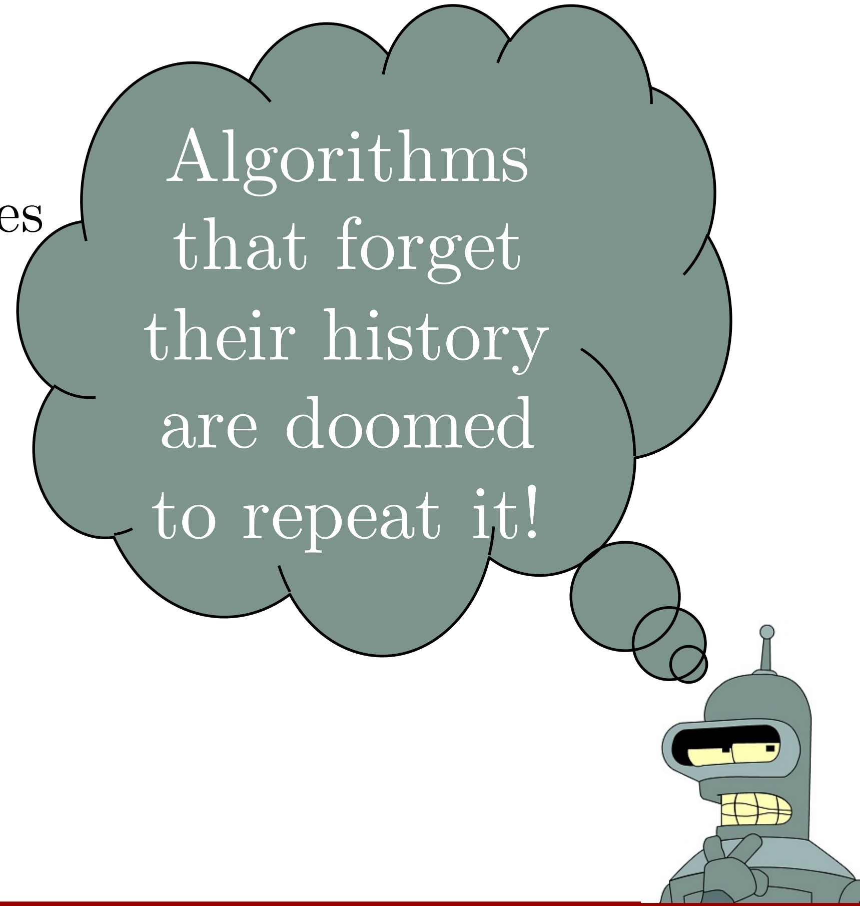

(a) The initial state
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
Frontier nodes
Explored nodes

Tree Search



- Tree search can expand the same states again and again
- In a rectangular grid:
  - Search tree of depth d has 4^d leaves
  - There are only $4d$ states within d steps of any given state

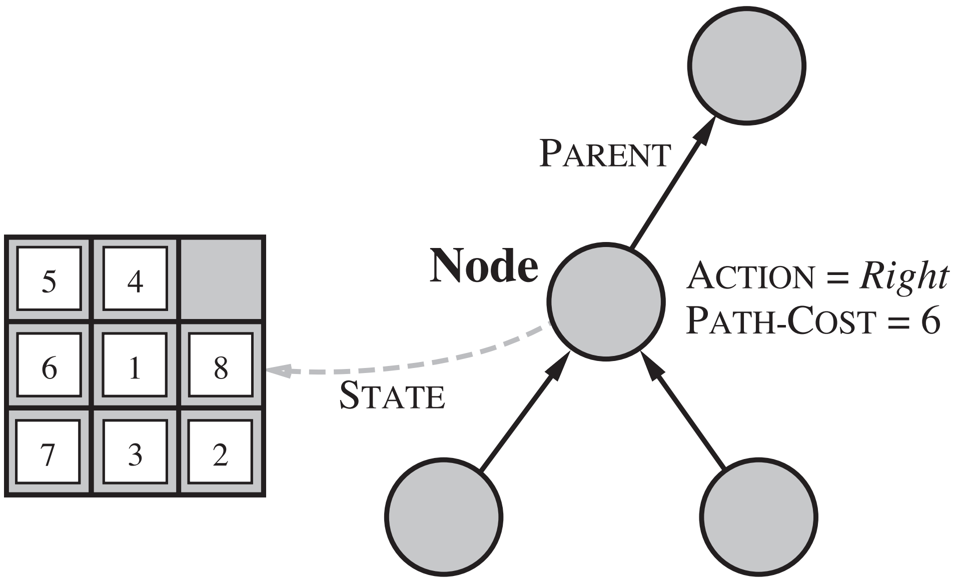




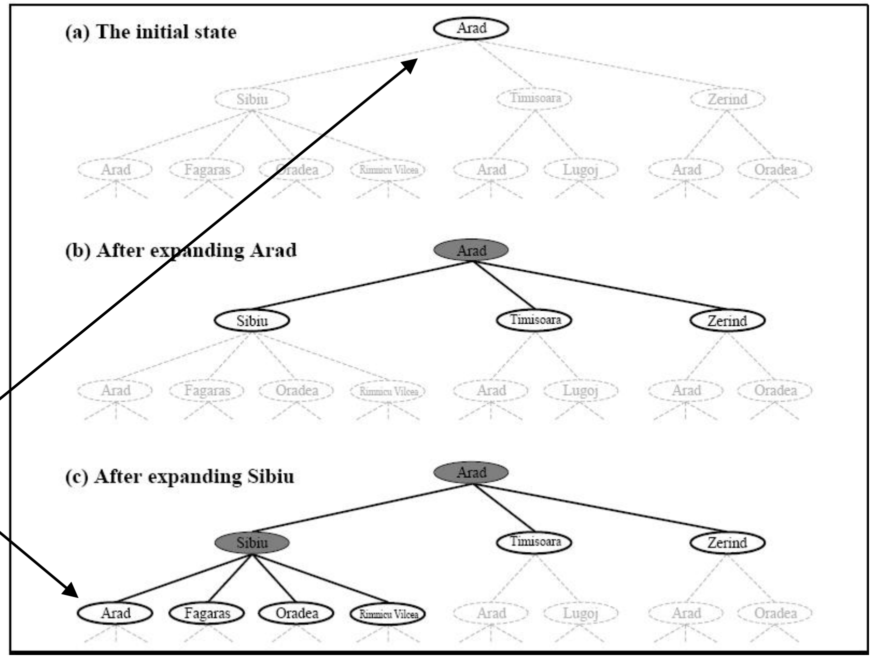
States vs. Nodes

- **State *s*:** an admissible configuration of the world
- **Node *n*:** a bookkeeping data structure used to represent the search tree, that contains:
  - n.STATE: the state s to which n corresponds to
  - n.PARENT: the node in the search tree that generated node n
  - n.ACTION: the action that was applied to the parent to generate n
  - n.PATH-COST: the cost $g(n)$ of the path from the initial node to n as indicated by the parent pointers (cost-to-come)
- Nodes are on specific paths, as defined by PARENT pointers, states are not
- Two different nodes can contain the same state s, if s was generated via two different search paths

States vs. Nodes



Same state,
different nodes



GRAPH SEARCH

function GRAPH-SEARCH(problem, strategy)

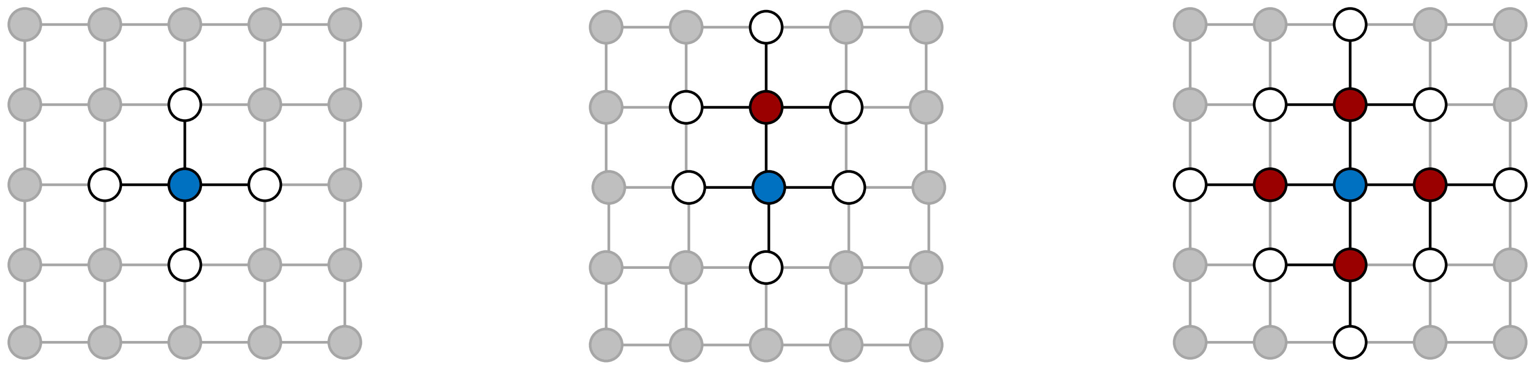
set of frontier nodes contains the start state of problem

loop

- if there are no frontier nodes then return failure
- choose a frontier node for expansion using strategy, and add it 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 frontier or explored set

Graph Search Illustrated

Each node is associated to a different state,
the search tree grows on the state graph

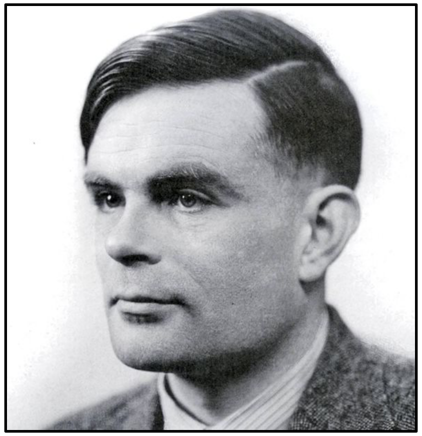


Separation property: Every path from *initial state* to an *unexplored state* has to pass through the frontier. The frontier separates explored vs. unexplored regions

Uninformed vs. Informed

Uninformed

Can only generate successors and distinguish goals from non-goals



Informed

Strategies that know whether one non-goal is more promising than another

MEASURING PERFORMANCE

Completeness

Guaranteed to find a solution when there is one?

Optimality

Finds the cheapest solution?

Time

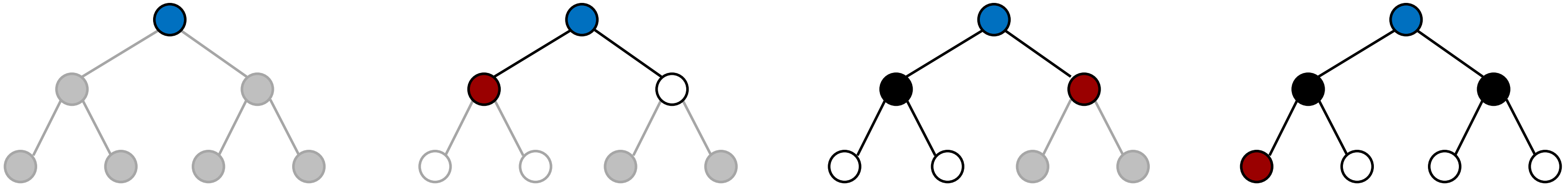
How long does it take to find a solution?

Space

How much memory is needed to perform the search?

Breadth-First Search

- Strategy: Expand shallowest unexpanded node
- Can be implemented by using a FIFO queue for the frontier
- Goal test applied when node is *generated*



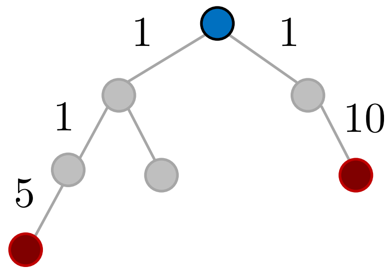
Breadth-First Search

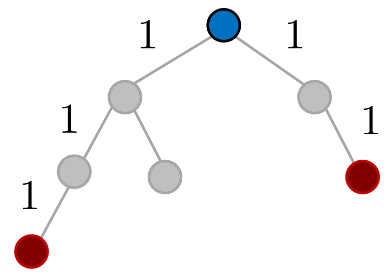
Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Not really	$\Theta(b^d)$	$\Theta(b^d)$

- Complete: Yes, also in infinite graphs, but b must be finite
- Optimality: If the path cost is a nondecreasing function of the depth (e.g., all actions have the same cost)
- Time complexity: Imagine each node has b successors, and solution is at depth d, then generate $\sum_{i=1}^{d} b^i = \Theta(b^d)$ nodes. If goal test is applied when node expanded ➔ $\Theta(b^{d+1})$
- Space complexity: For graph search $\Theta(b^d)$ nodes are in frontier and $\Theta(b^{d-1})$ in the explored (➔ tree search would not save much)

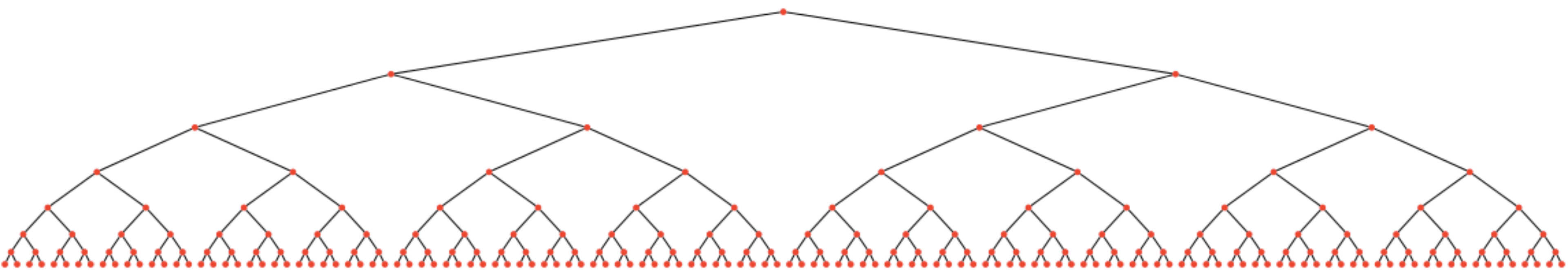
Breadth-First Search

Not optimal





Optimal



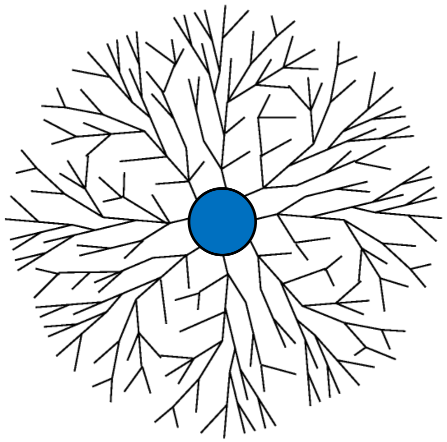
Exponential BFS tree growth (b=2)

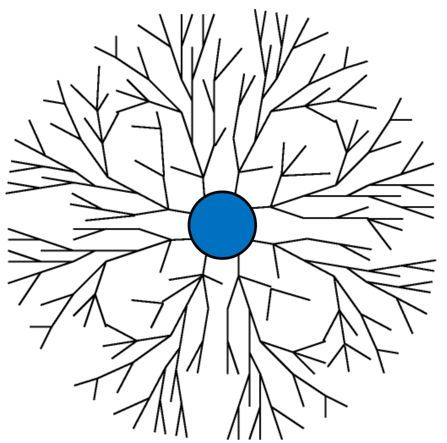
For b=10, d=12 ➔ $\Theta(10^{12})$ nodes

If 1 node requires 1 kB ➔ 10^{12} kB!

BIDIRECTIONAL SEARCH

- Idea: Possibly improve the running time of BFS by running two simultaneous searches, forward from the initial state and backward from the goal
- Poll 1: What is the worst-case running time of BIDIRECTIONAL SEARCH?
  1. $\Theta(b \cdot d)$
  2. $\Theta((b/2)^d)$
  3. $\Theta(b^{d/2})$
  4. $\Theta(b^d)$

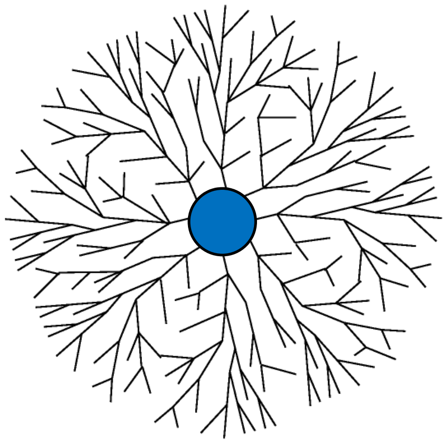


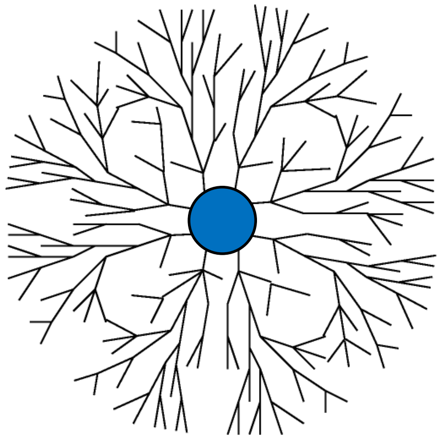


Bidirectional Search

$\Theta(b^{d/2}) + \Theta(b^{d/2}) \ll \Theta(b^d)$

For $b=10$, $d=6$, each BFS generates up to depth $d=3$ ➜ 2,220 nodes vs. 1,111,110, big memory save!



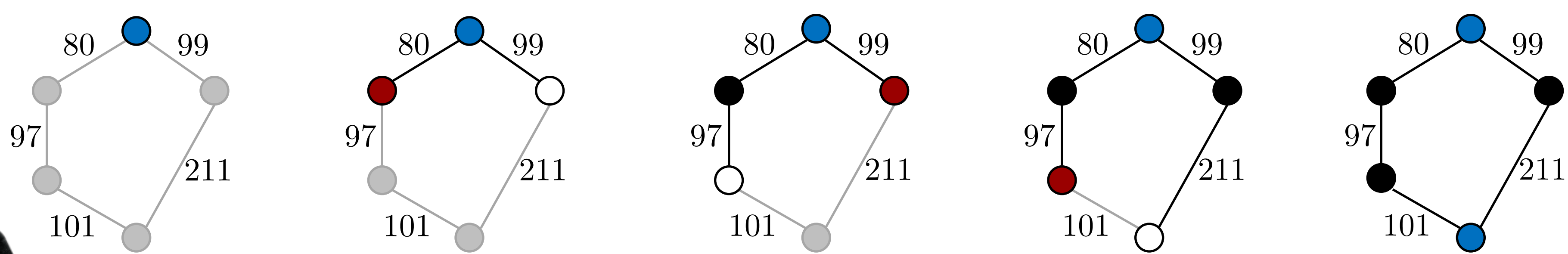


Issues:

- Asymmetric costs
- Unidirectional moves
- Repeated check for frontier intersection (additional constant time with hashing)
- Existence of multiple goals
- Abstract goal definition

Uniform-Cost Search

- Strategy: Expand unexpanded node with *lowest path cost* $g(n)$
- Can be implemented by using a priority queue ordered by $g(n)$ for the frontier
- Other changes from BFS:
  - Goal test applied when node is selected for expansion
  - If a successor is already in the frontier set, its path cost needs to be updated if lower than the previously computed one



Uniform-Cost Search

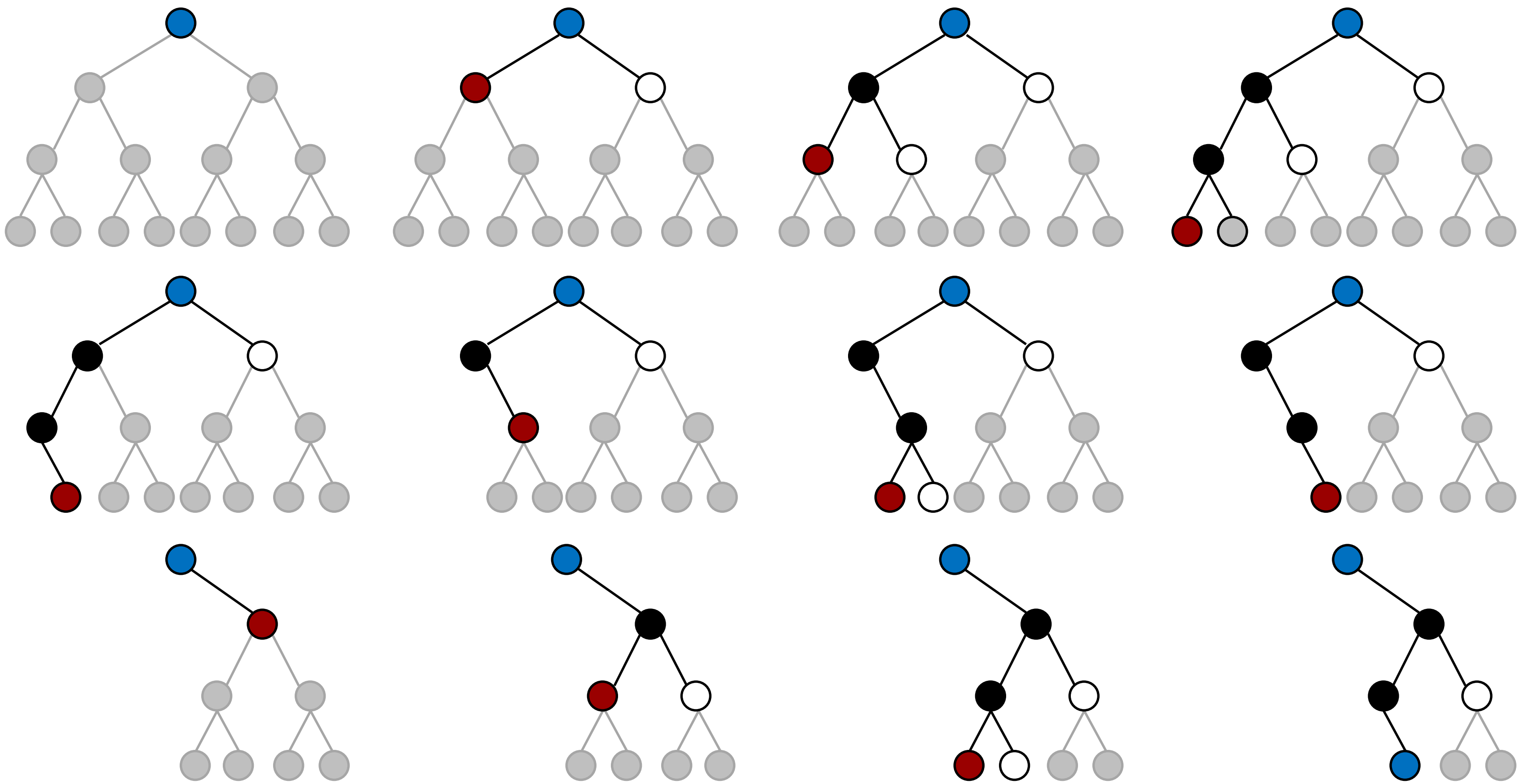
Algorithm	Complete?	Optimal?	Time	Space
UCS	Sorta	Yes	$\Theta(b^{1+\lfloor C^*/\epsilon \rfloor})$	$\Theta(b^{1+\lfloor C^*/\epsilon \rfloor})$

- Optimality: When a node is selected for expansion the optimal path to the node has been found
- Completeness: If the cost of every step exceeds $\epsilon > 0$ (and b is finite)
- Time complexity: If C^* is the cost of the optimal solution and ϵ is a lower bound on the step cost, the worst-case depth of the search tree is $1 + \lfloor C^*/\epsilon \rfloor$
- The complexity is $\Theta(b^{d+1})$ when step costs are uniform

Depth-First Search

- Strategy: Expand deepest unexpanded node
- Can be implemented by using a stack for the frontier (LIFO)
- Recursive implementation is also common

Depth-First Search



Depth-First Search

Algorithm	Complete?	Optimal?	Time	Space
DFS	No	No	$\Theta(b^m)$	$\Theta(b \cdot m)$

- Completeness: Clearly not in general
- Poll 2: In a finite state space, which version of DFS is complete?
  1. Tree Search
  2. Graph Search
  3. Both
  4. Neither

DEPTH-FIRST SEARCH

Algorithm	Complete?	Optimal?	Time	Space
DFS	No	No	$\Theta(b^m)$	$\Theta(b \cdot m)$

- Time complexity: $\Theta(b^m)$, where m is the maximum depth of any solution!
- Space complexity: *DFS tree search* needs to store only a single path from the root to a leaf, along with unexpanded sibling nodes for each node on the path
- Consequently, depth-first tree search is the workhorse of many areas of AI (including CSPs and SAT solving)

ITERATIVE DEEPENING SEARCH

Algorithm	Complete?	Optimal?	Time	Space
IDS	Yes	No	$\Theta(b^d)$	$\Theta(b \cdot d)$

- Run DFS with depth limit $\ell = 1,2,\ldots$
- Combines the best properties of BFS and DFS
- Completeness: Yes, for the same reason BFS is complete
- Time complexity: Seems wasteful but most of the nodes are at the bottom level; total

$$d \cdot b + (d-1)b^2 + \cdots + 1 \cdot b^d = \Theta(b^d)$$

Summary of Algorithms

Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Not really	$\Theta(b^d)$	$\Theta(b^d)$
UCS	Sorta	Yes	$\Theta(b^{1+\lfloor C^*/\epsilon \rfloor})$	$\Theta(b^{1+\lfloor C^*/\epsilon \rfloor})$
DFS	No	No	$\Theta(b^m)$	$\Theta(b \cdot m)$
IDS	Yes	No	$\Theta(b^d)$	$\Theta(b \cdot d)$

SUMMARY

- Terminology:
  - Search problems
  - Local search
- Algorithms:
  - Generic search algorithms:
    tree search vs. graph search
  - Strategies: BFS, Bidirectional,
    UCS, DFS, Iterative Deepening
  - Local search algorithms …. Next time