CSCU9YE - Artificial Intelligence

Lecture 4: Problem Solving by Search (Paths)

Prof. Gabriela Ochoa, University of Stirling

Search in Computing Science

At least 4 meanings of the word ***search*** in CS

1. **Search for stored data**	2. **Search for web documents**
• Finding information stored in disc or memory. • Examples: Sequential search, Binary search	• Finding information on the world wide web • Results are presented as a list of results
3. Search for paths or routes	**4. Search for solutions**
• Finding a set of actions that will bring us from an initial stat to a goal stat • Relevant to AI • Algorithms: depth first search, breadth first search, branch and bound, A*, Monte Carlo tree search.	• Find a solution in a large space of candidate solutions • Relevant to AI, Optimisation, Operational Research • Algorithms: evolutionary algorithms, Tabu search, simulated annealing, ant colony optimisation, etc.

Content

- Examples of search problems
- Formal problem formulation
- Examples
- Solving using Tree-based search algorithms
    - Uninformed algorithms
    - Informed algorithms

Examples of search problems

- A robot vehicle would search for a route to a given destination.
- An automated air traffic controller would search for a safe landing sequence for a set of incoming planes
- In games of strategy, such as chess or checkers: search for a sequence of moves to beat your opponent
- Search problems are common in AI: Planning and Learning.

Solving problems by searching

- Problem-solving agents decide what to do by finding sequences of actions that lead to desirable states

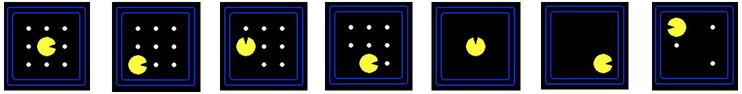
- What is a *problem* and what is a *solution*?
  - Problem: a goal and a set of means to achieve it
  - Solution: a sequence of actions to achieve that goal

- Given a precise definition of *problem*, it is possible to construct a search process for finding solutions

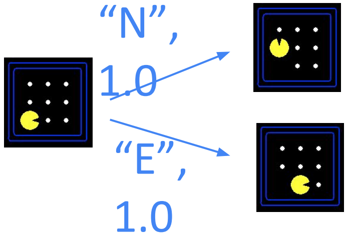
Search Problem: example Pac-Man

- A search problem consists of:

  A set of states
  (state space)



  A set of Actions
  (transitions, costs)

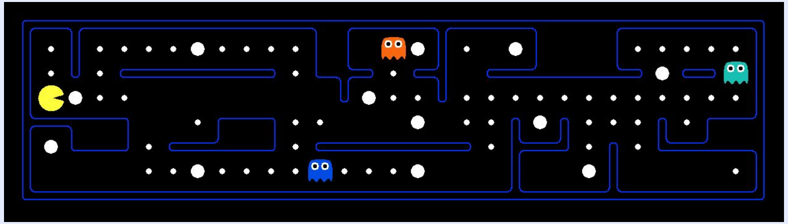


  A start state and a goal test

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

What's in a State Space?

The world state includes every last detail of the environment



A search state keeps only the details needed for planning (abstraction)

- **Problem**: Find paths
  - **States**: (x,y) location
  - **Actions**: NSEW
  - **Successor**: update location only
  - Goal test: (x,y)=END

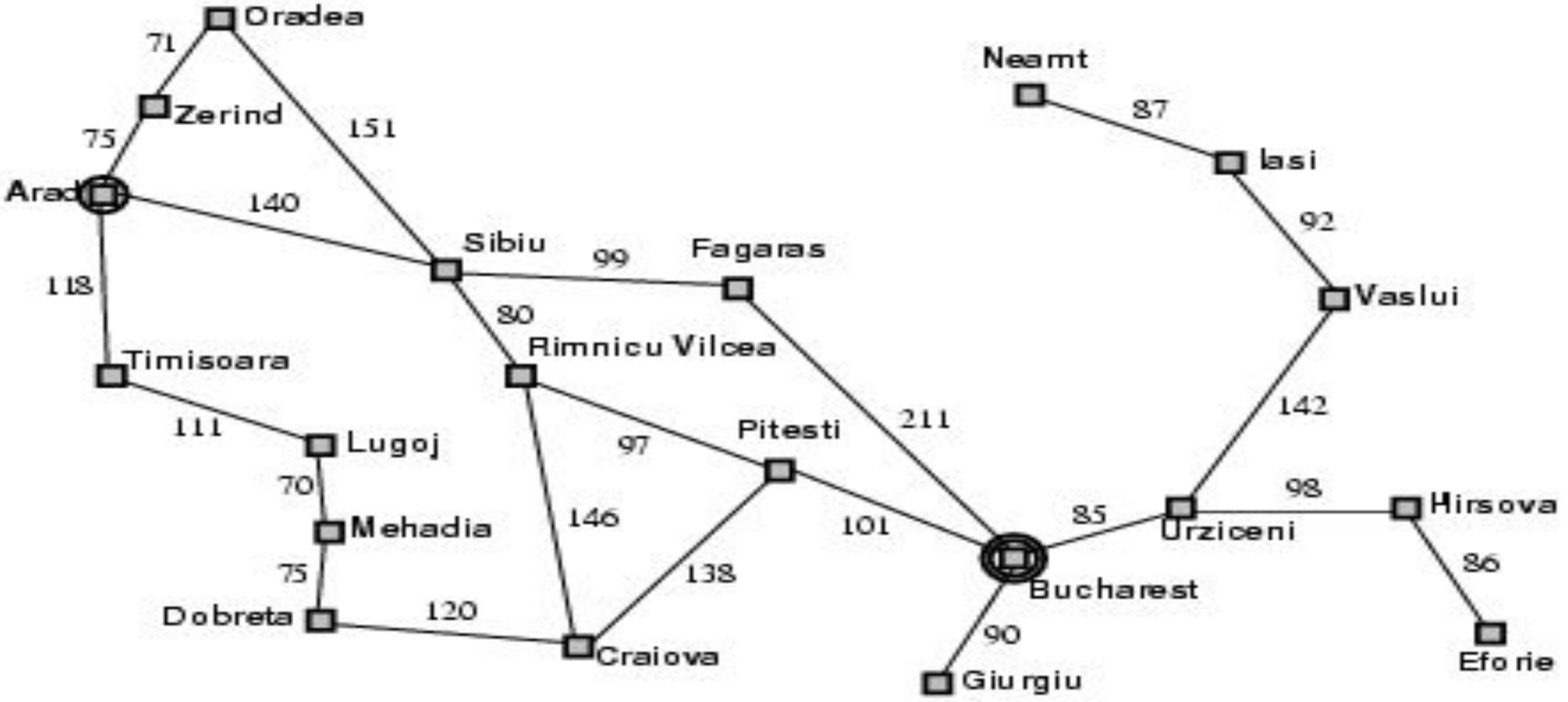
- **Problem**: Eat-All-Dots
  - **States**: {(x,y), dot Booleans}
  - **Actions**: NSEW
  - **Successor**: update location and possibly eat a dot
  - **Goal test**: dots all eaten

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

Google Map: [Romania]



Abstraction: Selecting a state space

- Real world is absurdly complex

  state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "Arad ->Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

Problem formulation

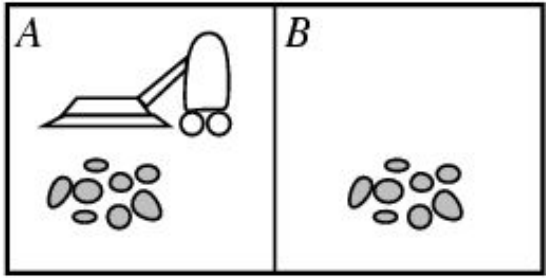
More formally, a problem is defined by 5 components:

1. Initial state where the agent starts: e.g., "at Arad"
2. Actions available to the agent

   e.g., *Arad -> Zerind* , *Arad -> Sibiu, ... etc.*

3. Transition model: description of what each action does. State that results from doing action *a* in state *s.*
4. Goal test, determines whether a given state is a goal state.
   a. explicit, e.g., *x* = "at Bucharest“
   b. implicit, e.g., *Checkmate(x)*
5. Path cost (additive) function that assigns a numeric cost to each path. Reflects agents performance measure
   a. e.g., sum of distances, number of actions executed, etc.
   b. *c(x,a,y)* is the step cost, assumed to be ≥ 0

More examples: The vacuum world



- States: determined by both the agent location and the dirt locations. Question : How many states?
- Initial states: Any state can be designated as the initial state.
- Actions: 3 actions: *Left*, *Right*, and *Suck*
- Transition model: effects of the actions, transition among states. Some actions have no effect. Example: *sucking* in a clean square
- 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 in the path.

The vacuum world

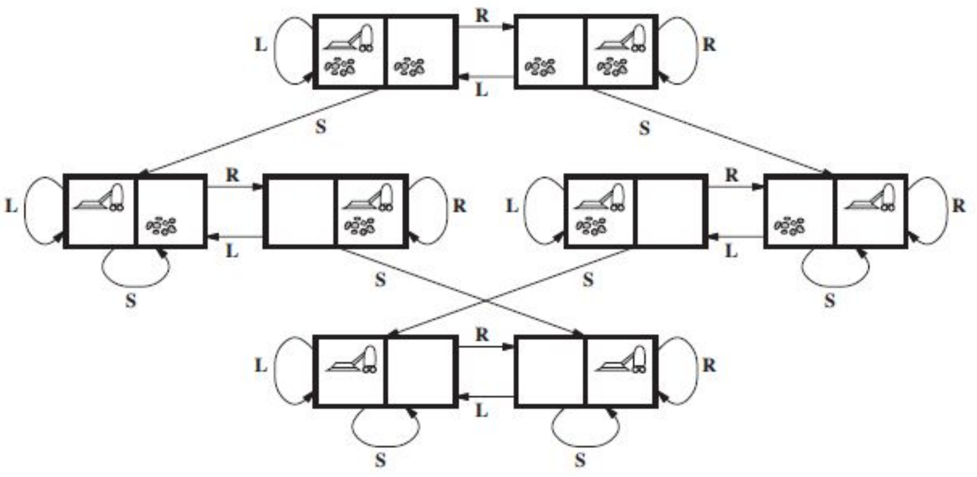


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Compared with the real world, this toy problem has discrete locations, discrete dirt, reliable cleaning, and it never gets any dirtier

Graph Theory and Pathfinding Basics for Games Development

A non-player character (NPC) is a video game character that is controlled by the game's artificial intelligence (AI) rather than by a gamer.

Tree-based search algorithms

- Having formulated some problems, we now need to solve them.
- A solution is an action sequence, so search algorithms work by considering various possible action sequences.

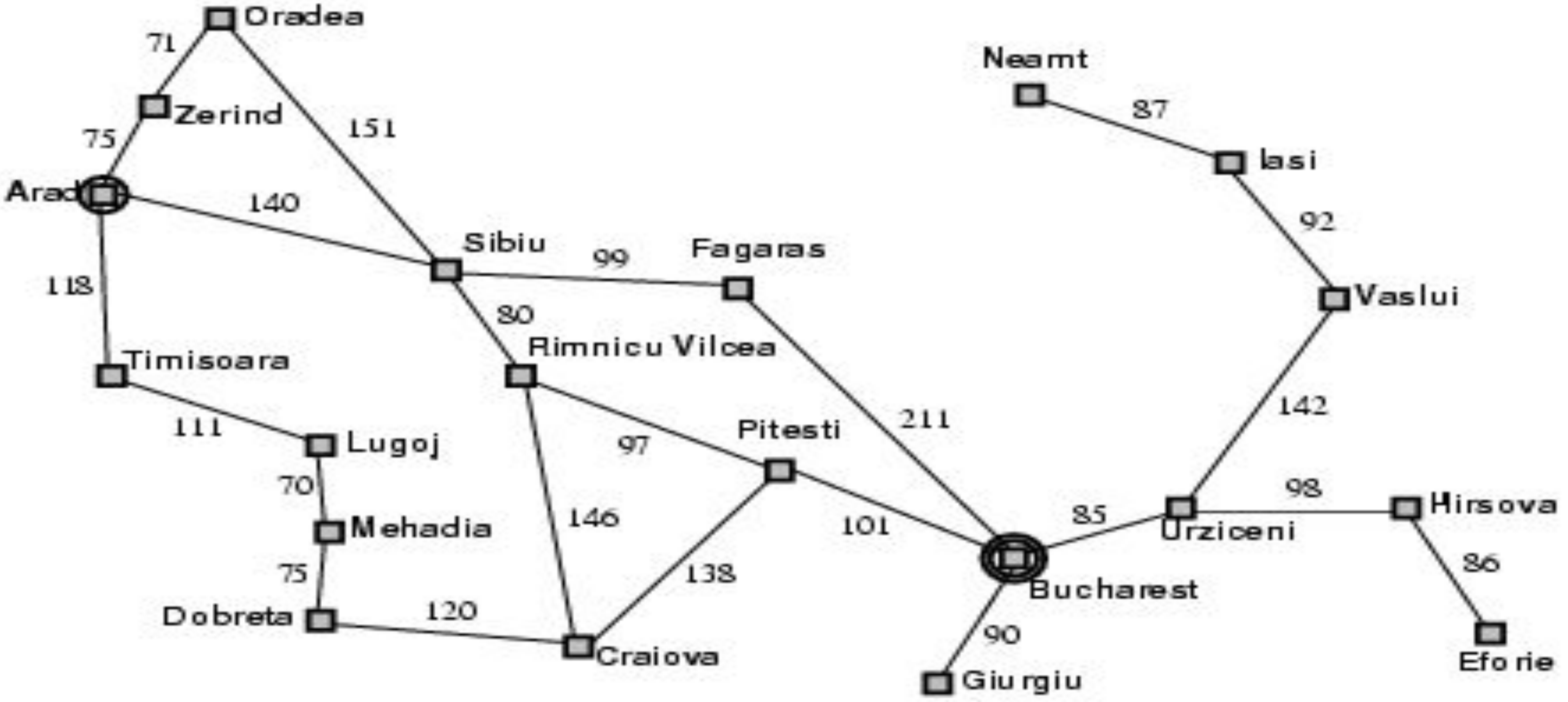
Tree search algorithms

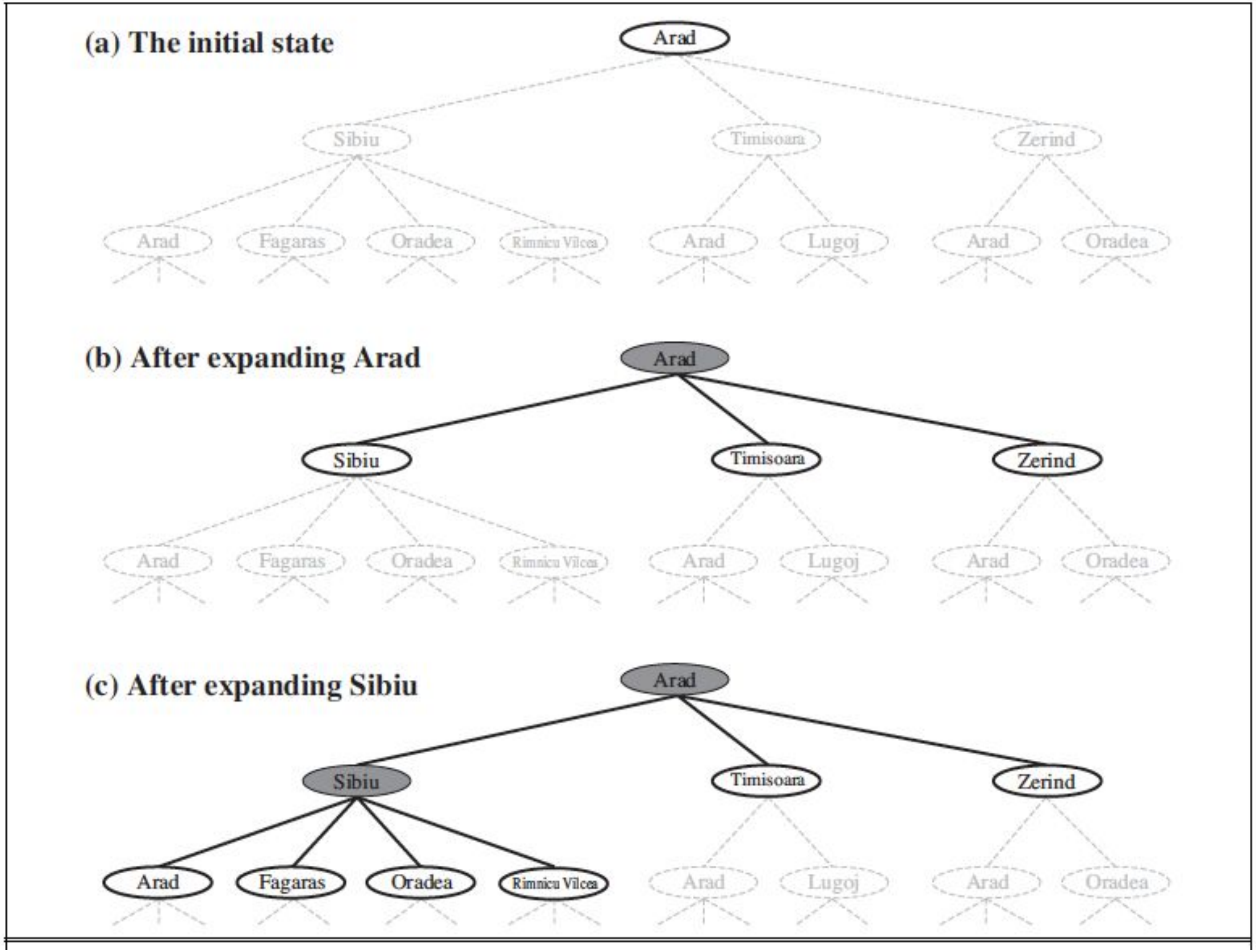
- The possible action sequences starting at the initial state form a tree with the initial state at the root;
- Basic idea: Simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

Example: Romania

Start: Arad
Goal: Bucharest





Outlined nodes: generated but not expanded

Shaded nodes: expanded

Faint dashed nodes: not yet been generated

Partial search trees for finding a route from Arad to Bucharest.

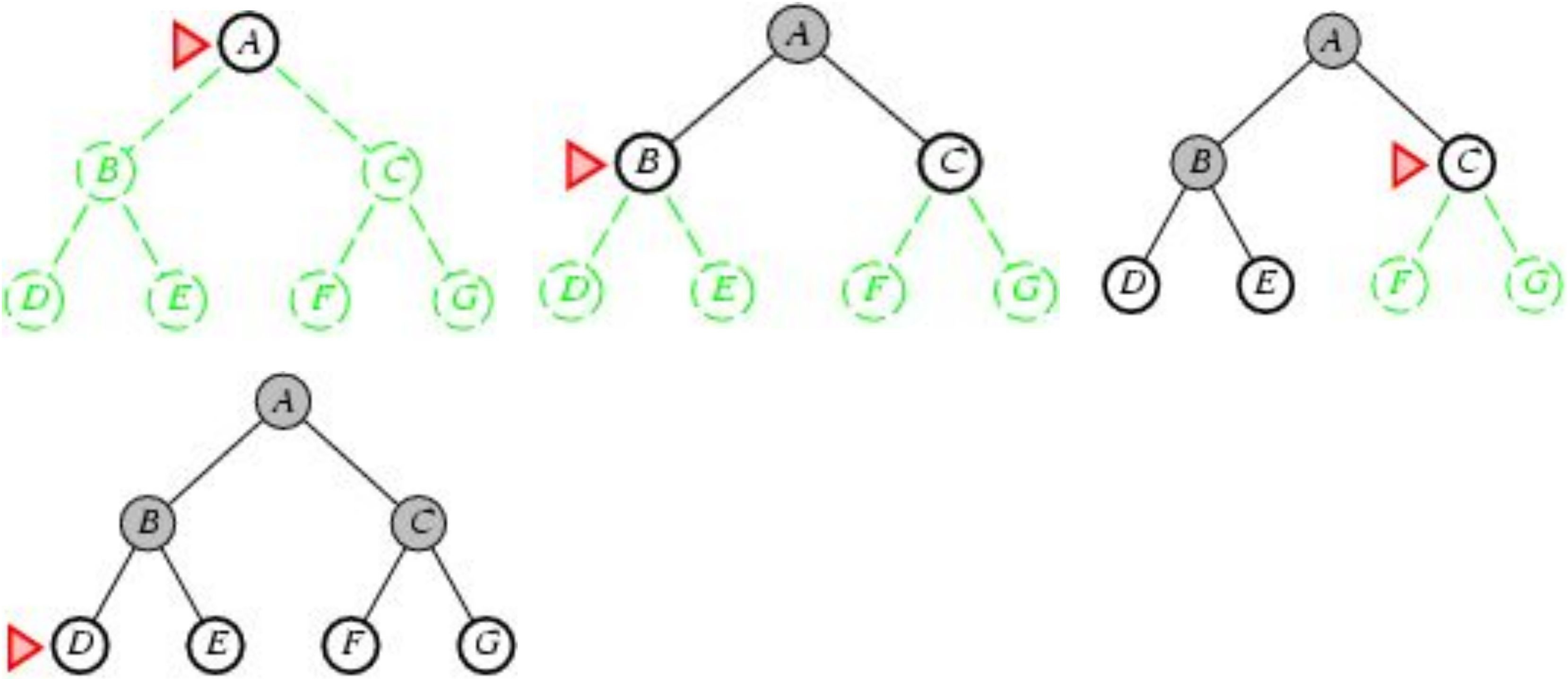
Uninformed search strategies

Uninformed, also called blind search strategies use only the information available in the problem definition

- Breadth-first search (BFS)
- Depth-first search (DFS)
- Improvements of DFS
  - Depth-limited search
  - Iterative deepening search

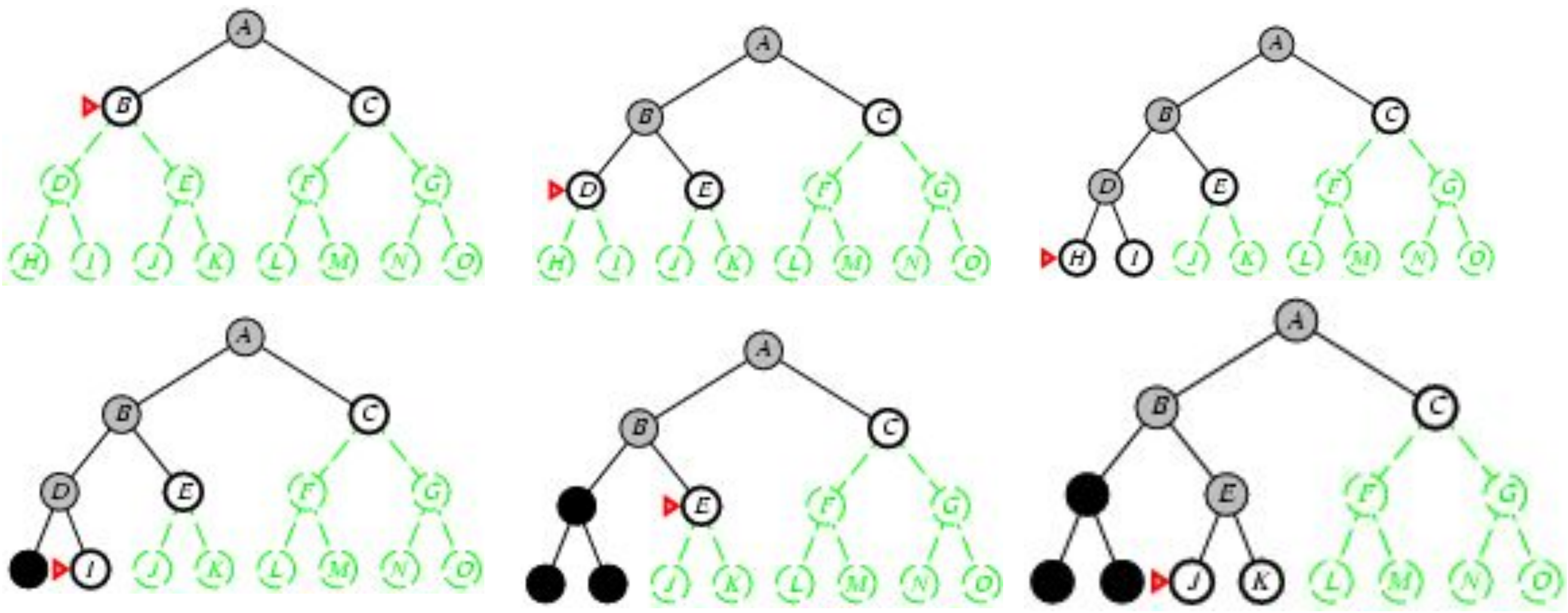
Breadth-first search

Strategy: Expand shallowest unexpanded node. Implementation: FIFO



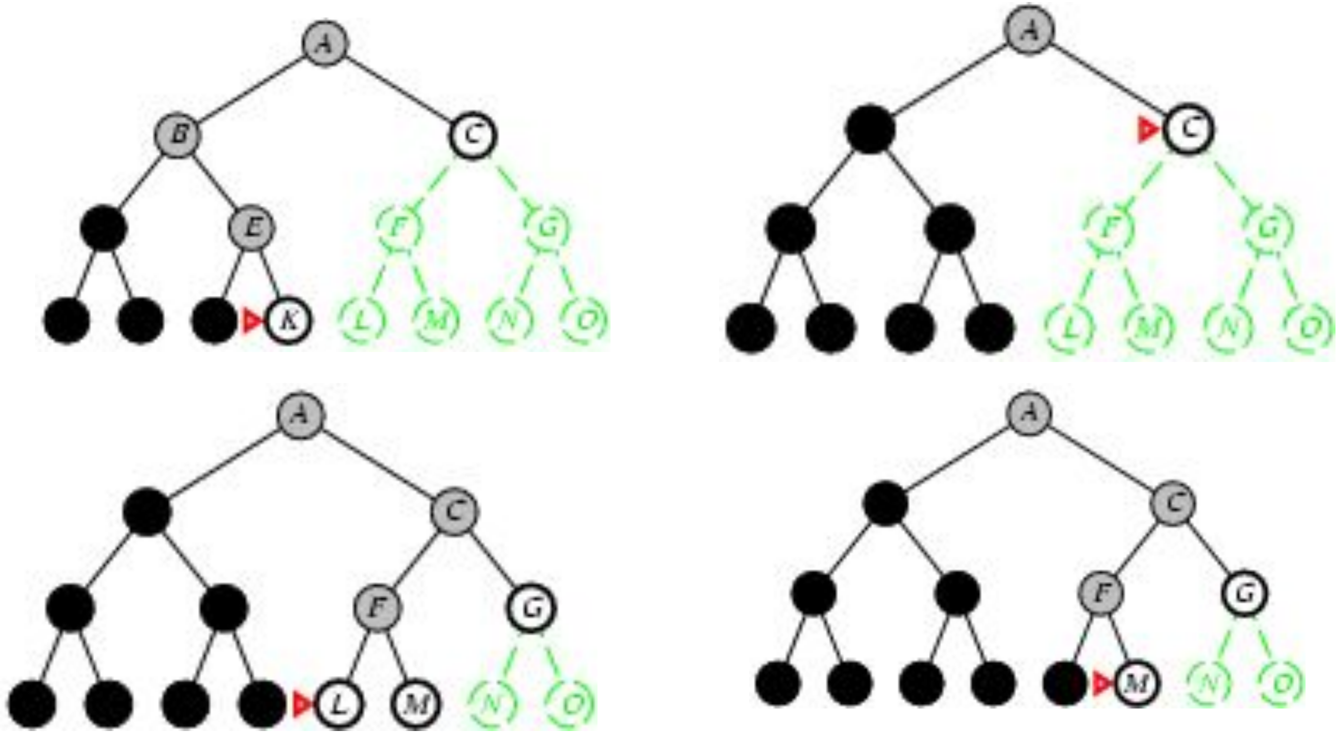
Depth-first search

Expand deepest unexpanded node. Implementation: LIFO queue (STACK),



Depth-first search

Expand deepest unexpanded node. Implementation: LIFO queue (STACK),



Comparing Search strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
    - completeness: does it always find a solution if one exists?
    - time complexity: number of nodes generated
    - space complexity: maximum number of nodes in memory
    - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
    - b: maximum branching factor of the search tree
    - d: depth of the least-cost solution
    - m: maximum depth of the state space (may be ∞)

Improving Depth-first search

Depth-limited search

- Avoid problems of DFS by imposing a maximum depth of a path
- Complete but not optimal

Iterative deepening search

- Sidesteps the issue of choosing the depth limit
- Tries all possible depth limits: 0, 1, 2, etc
- Combines the benefits of DFS and BFS
  - Optimal and complete like BFS
  - Modest memory requirements like DFS

Comparing Search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Time and space complexity are measured in terms of

b: maximum branching factor of the search tree

d: depth of the least-cost solution

m: maximum depth of the state space (may be ∞)

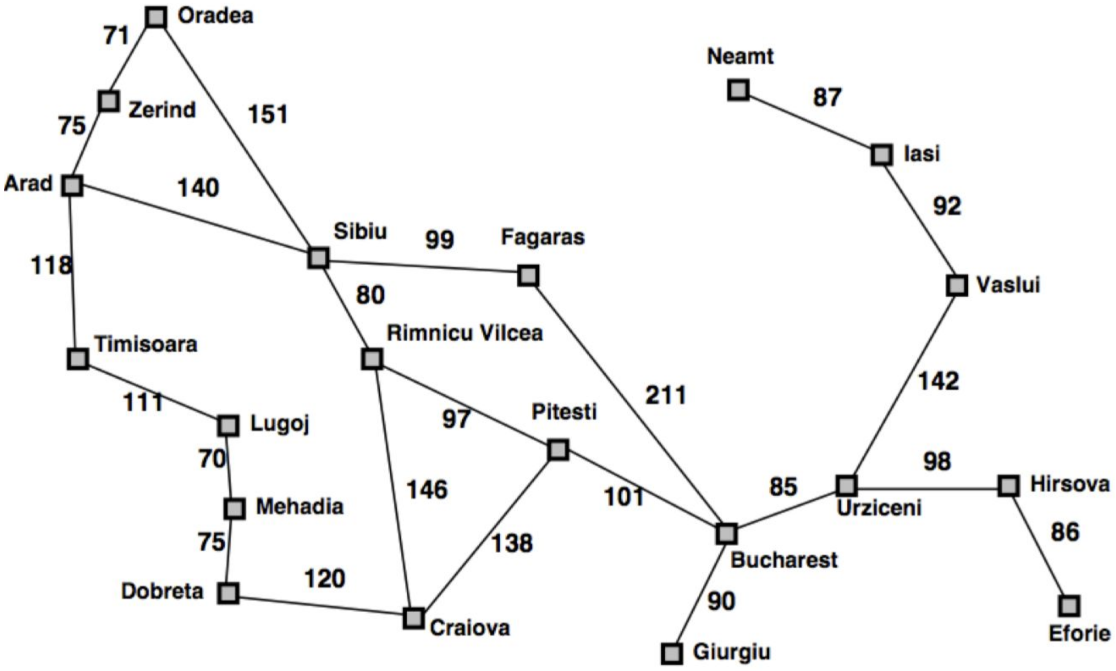
Informed search

- Uninformed search strategies can find solutions to problems by systematically generating new states, and testing them against the goal
- These strategies are inefficient in most cases
- Informed search strategies: use problem-specific knowledge
- Knowledge is given by an *evaluation function* that returns a number describing the desirability (or lack thereof) of expanding a nodes

Best-first search

- Idea: use an evaluation function for each node – estimate of "desirability" ⇒ Expand most desirable unexpanded node
- Implementation: a queue sorted in decreasing order of desirability
- Special cases:
  - greedy search: Chooses (expands) the step that takes us closest to the goal at each branch of the tree. Repeats until goal found or end of tree.
  - A∗ search: expand node on the least-cost solution path

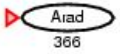
Romania with step costs in km

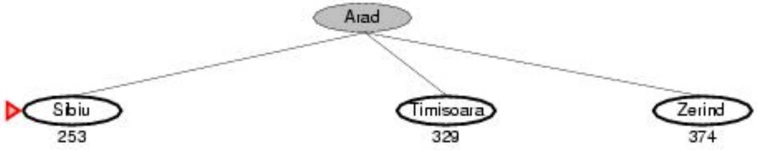


Straight–line distance
to Bucharest

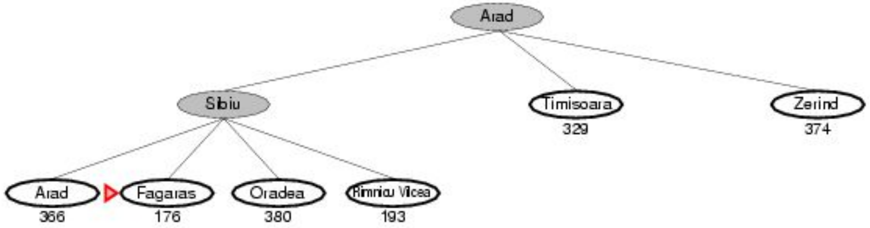
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

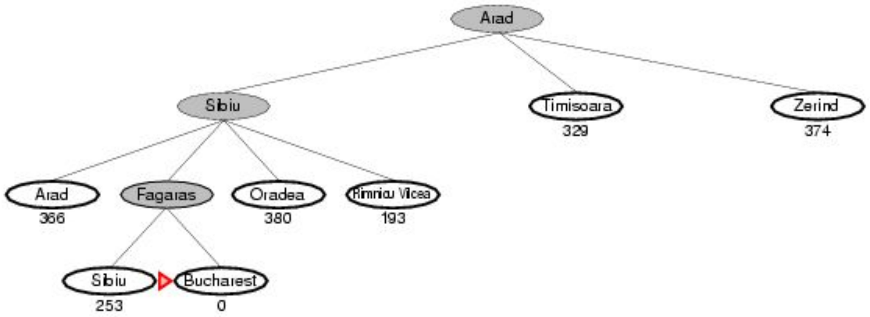
Greedy search example





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

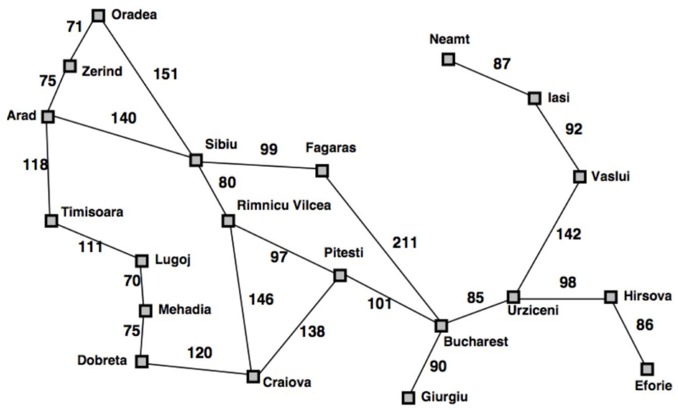




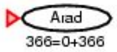
A* search: Minimising the total estimated solution cost

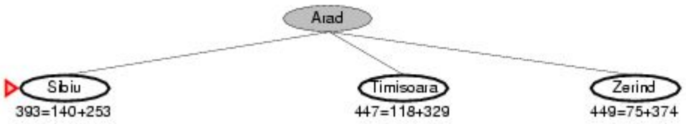
- The most widely known form of best-first search
- Node evaluation, combines:
  - $g(n)$: path cost from the start node to node n
  - $h(n)$: the cost to get from the node to the goal
  - $f(n) = g(n) + h(n)$
- $f(n)$ is the estimated cost of the cheapest solution through n
- It makes sense to try first the node with lowest $f(n)$
- This strategy is more than just reasonable: provided that the heuristic function h(n) satisfies certain conditions, A∗ search is both complete and optimal.

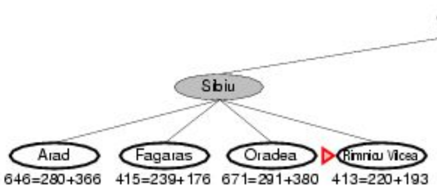
A* search example

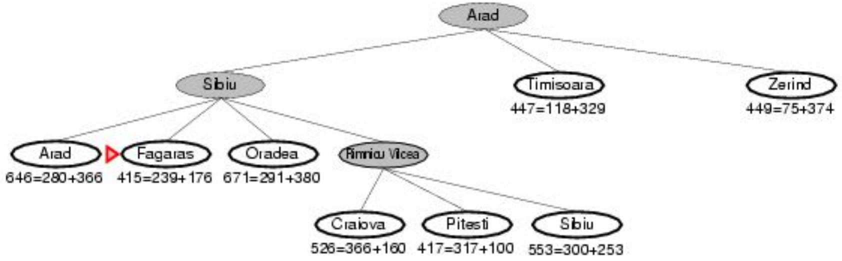


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

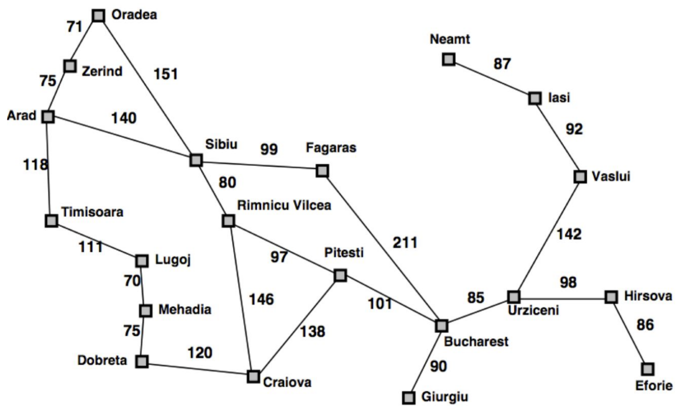






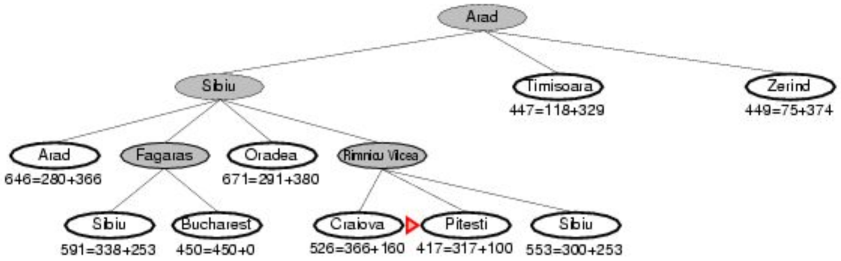


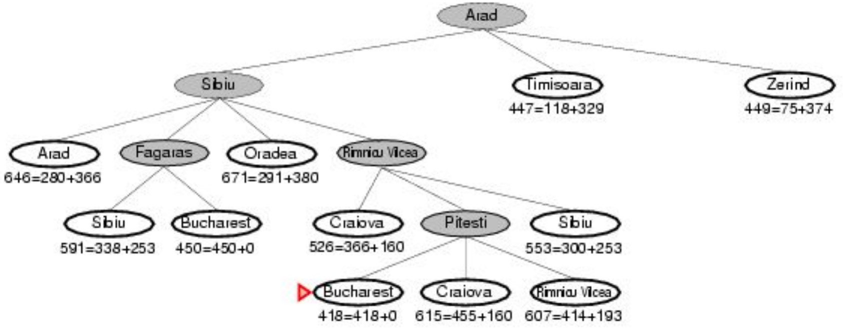
A* search example



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





Summary: formulating problems as search problems

A search problem consists of:

- A set of states (state space)
- A set of Actions (transitions, 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

Summary: tree-based search methods

Uninformed Search

Breadth-first search

Depth-first search

Depth-first search improvements

- Depth-limited search
- Iterative deepening search

Informed Search

Best-first search: ‘minimum’ cost nodes are expanded first

- Greedy search: Expands node closer to goal
- A*: Expands the node on the least-cost solution pat. Complete and optimal. Uses a lot of memory!