NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence

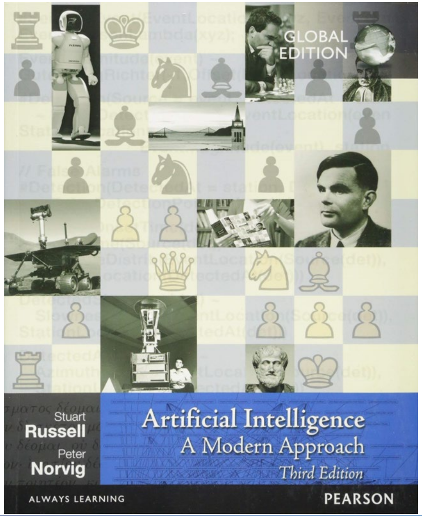
Intelligent Agents and Search

Assoc Prof Bo AN

www.ntu.edu.sg/home/boan
Email: boan@ntu.edu.sg
Office: N4-02b-55

Lesson Outline

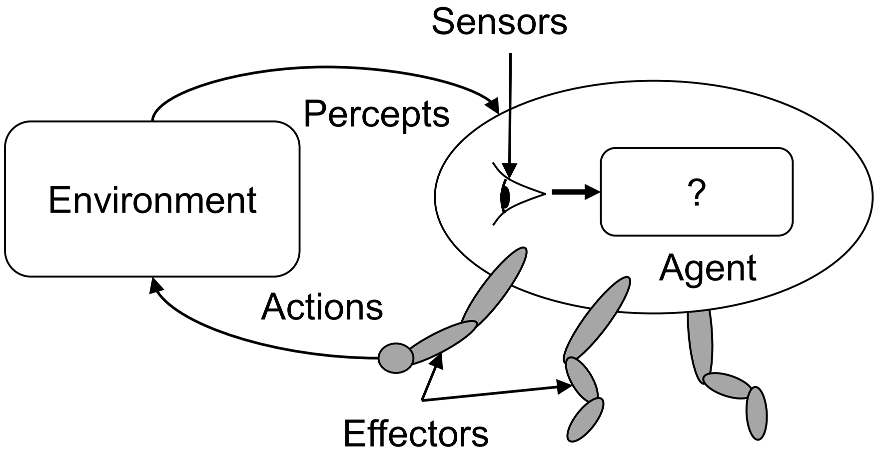
- How can one describe the task/problem for the agent?
- What are the properties of the task environment for the agent?
- Problem formulation
- Uninformed search strategies
- Informed search strategies: greedy search, A * search



Agent

An agent is an entity that

- Perceives through sensors (e.g. eyes, ears, cameras, infrared range sensors)
- Acts through effectors (e.g. hands, legs, motors)



Rational Agents

- A rational agent is one that does the right thing
- Rational action: action that maximises the expected value of an objective performance measure given the percept sequence to date
- Rationality depends on
  - performance measure
  - everything that the agent has perceived so far
  - built-in knowledge about the environment
  - actions that can be performed

Example: Google X2: Driverless Taxi

- **Percepts:** video, speed, acceleration, engine status, GPS, radar, …
- **Actions:** steer, accelerate, brake, horn, display, …
- **Goals:** safety, reach destination, maximise profits, obey laws, passenger comfort,…
- **Environment:** Singapore urban streets, highways, traffic, pedestrians, weather, customers, …

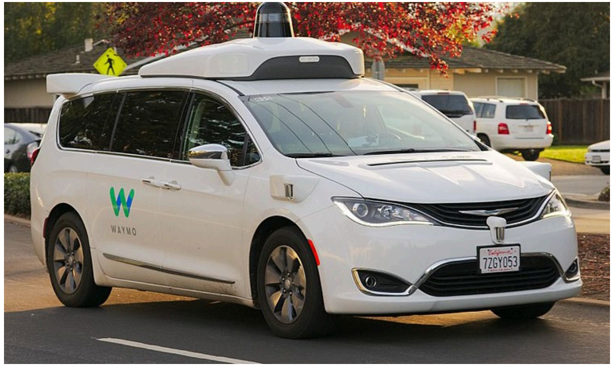


Image source: https://en.wikipedia.org/wiki/Waymo#/media/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg

More Examples

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient’s answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sorts into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student’s score on test	Set of students

Types of Environment

Accessible (vs inaccessible)	Agent's sensory apparatus gives it access to the complete state of the environment
Deterministic (vs nondeterministic)	The next state of the environment is completely determined by the current state and the actions selected by the agent
Episodic (vs Sequential)	Each episode is not affected by the previous taken actions
Static (vs dynamic)	Environment does not change while an agent is deliberating
Discrete (vs continuous)	A limited number of distinct percepts and actions

Example: Driverless Taxi

Accessible?	No. Some traffic information on road is missing
Deterministic?	No. Some cars in front may turn right suddenly
Episodic?	No. The current action is based on previous driving actions
Static?	No. When the taxi moves, Other cars are moving as well
Discrete?	No. Speed, Distance, Fuel consumption are in real domains

Example: Chess

Accessible?	Yes. All positions in chessboard can be observed
Deterministic?	Yes. The outcome of each movement can be determined
Episodic?	No. The action depends on previous movements
Static?	Yes. When there is no clock, when are you considering the next step, the opponent can't move; Semi. When there is a clock, and time is up, you will give up the movement
Discrete?	Yes. All positions and movements are in discrete domains

More Examples

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Design of Problem-Solving Agent

Idea

- Systematically considers the expected outcomes of different possible sequences of actions that lead to states of known value
- Choose the best one
  - shortest journey from A to B?
  - most cost effective journey from A to B?

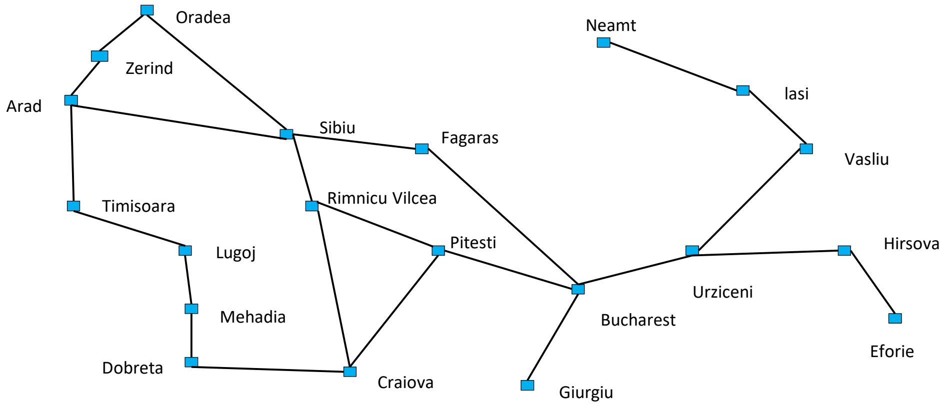
Design of Problem-Solving Agent

Steps

1. Goal formulation
2. Problem formulation
3. Search process
    - No knowledge → uninformed search
    - Knowledge → informed search
4. Action execution (follow the recommended route)

Example: Romania

- Goal: be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

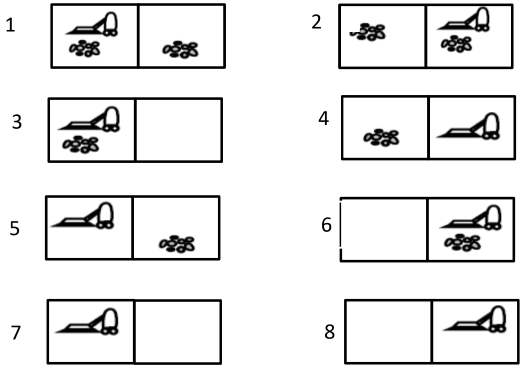


Example: Vacuum Cleaner Agent

- Robotic vacuum cleaners move autonomously
- Some can come back to a docking station to charge their batteries
- A few are able to empty their dust containers into the dock as well

Example: A Simple Vacuum World

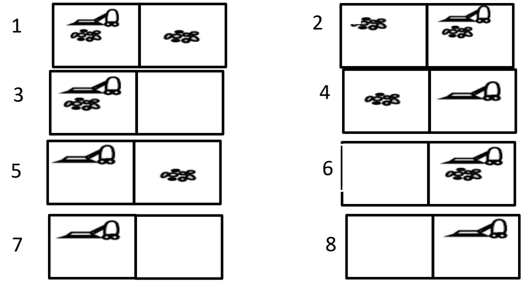
- Two locations, each location may or may not contain dirt, and the agent may be in one location or the other



- 8 possible world states
- Possible actions: left, right, and suck
- Goal: clean up all dirt → Two goal states, i.e. {7, 8}

Single-State Problem

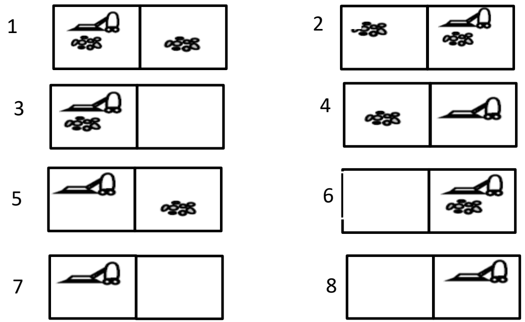
– Accessible world state (sensory information is available)
– Known outcome of action (deterministic)



– e.g.: start in #5
  – Solution: **right, suck**

Multiple-State Problem

- Inaccessible world state (with limited sensory information):
  - agent only knows which sets of states it is in
- Known outcome of action (deterministic)



- e.g.: start in {1, 2, 3, 4, 5, 6, 7, 8}
  - Action **right** goes to {2, 4, 6, 8}
  - Solution: **right, suck, left, suck**

Well-Defined Formulation

Definition of a problem	The information used by an agent to decide what to do

Specification	• Initial state
	• Action set, i.e. available actions (successor functions)
	• State space, i.e. states reachable from the initial state
	    • Solution path: sequence of actions from one state to another
	• Goal test predicate
	    • Single state, enumerated list of states, abstract properties
	• Cost function
	    • Path cost g(n), sum of all (action) step costs along the path

Solution	A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test

Measuring Problem-Solving Performance

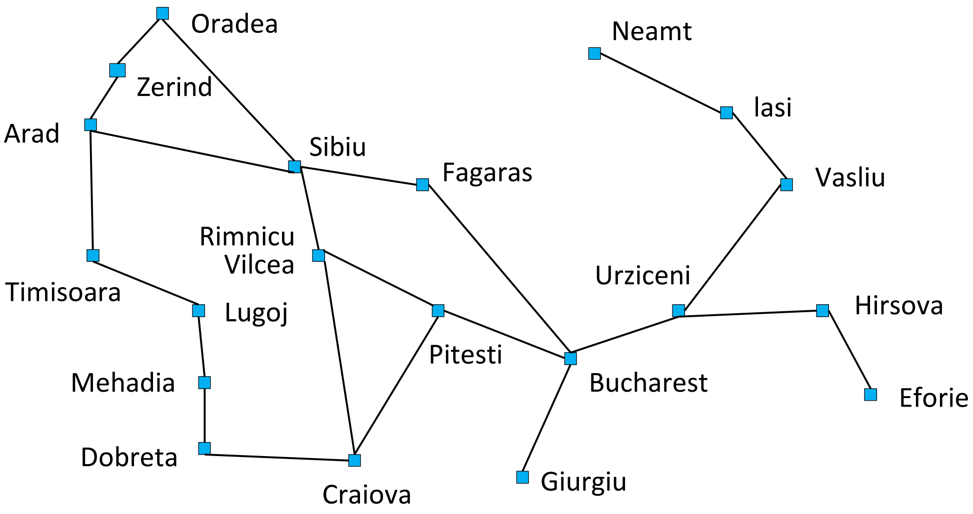
Search Cost

- What does it cost to find the solution?
    - e.g. How long (time)? How many resources used (memory)?

Total cost of problem-solving

- Search cost (“offline") + Execution cost (“online")
- Trade-offs often required
    - Search a very long time for the optimal solution, or
    - Search a shorter time for a “good enough" solution

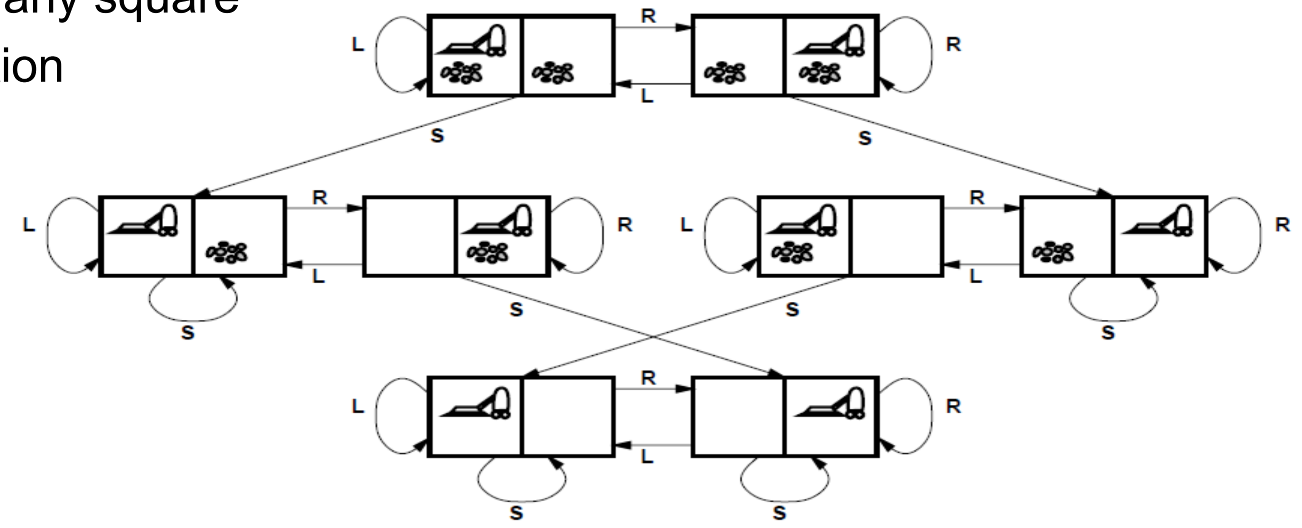
Single-State Problem Example



- Initial state: e.g., “at Arad“
- Set of possible actions and the corresponding next states
  - e.g., Arad → Zerind
- Goal test:
  - explicit (e.g., x = “at Bucharest")
- Path cost function
  - e.g., sum of distances, number of operators executed solution: a sequence of operators leading from the initial state to a goal state

Example: Vacuum World (Single-state Version)

- Initial state: one of the eight states shown previously
- Actions: left, right, suck
- Goal test: no dirt in any square
- Path cost: 1 per action



Multiple-State Problem Formulation

- ❑ Initial state set
- ❑ Set of possible actions and the corresponding sets of next states
- ❑ Goal test
- ❑ Path cost function

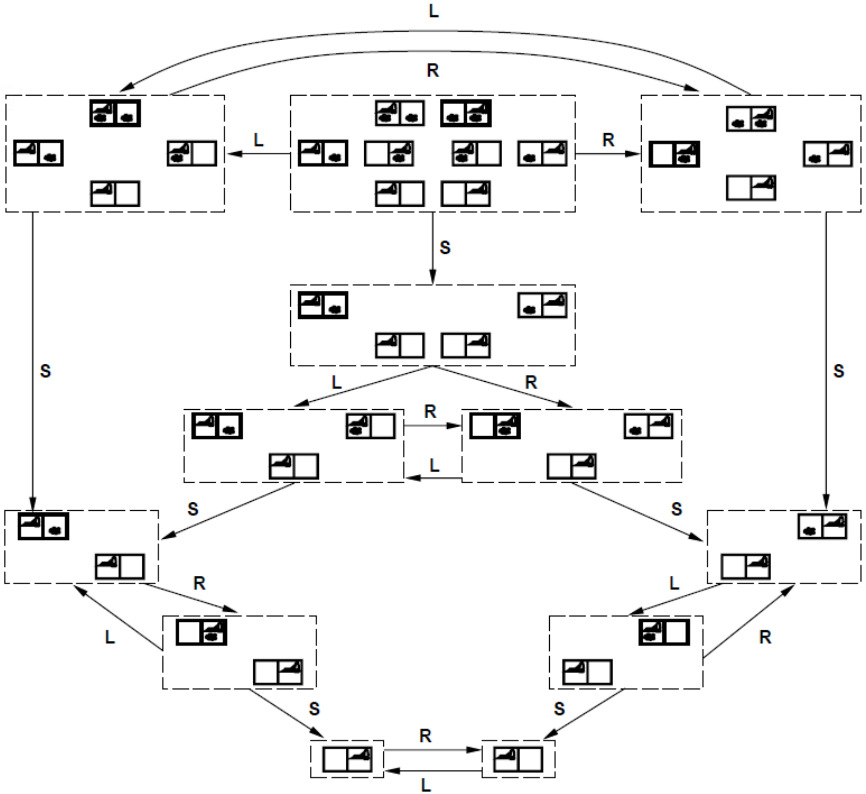
- Solution:

- ❑ a path (connecting sets of states) that leads to a set of states all of which are goal states

Example: Vacuum World (Multiple-state Version)

- States: subset of the eight states
- Operators: left, right, suck
- Goal test: all states in state set have no dirt
- Path cost: 1 per operator

Example: Vacuum World (Multiple-state Version)



Example: 8-puzzle

- States: integer locations of tiles
  - number of states = 9!
- Actions: move blank left, right, up, down
- Goal test: = goal state (given)
- Path cost: 1 per move

Start state

5	4	
6	1	8
7	3	2

Goal state

1	2	3
8		4
7	6	5

Real-World Problems

Route finding problems:

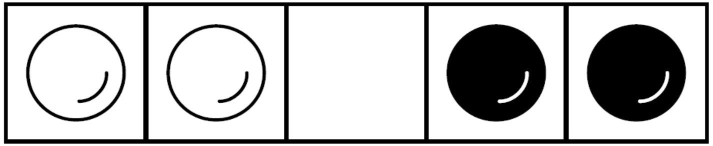
- Routing in computer networks
- Robot navigation
- Automated travel advisory
- Airline travel planning

Touring problems:

- Traveling Salesperson problem
- “Shortest tour": visit every city exactly once

Question to Think About

The stone puzzle is characterised as follows. Two white stones and two black stones are initially positioned on a board, as illustrated below. The board is composed of a single row of five squares, each of which can be either empty or hold one stone. The white stones can only move to the right and the black stones only to the left, either into an immediately adjacent empty square or by jumping over an adjacent stone of opposite colour into an empty square.

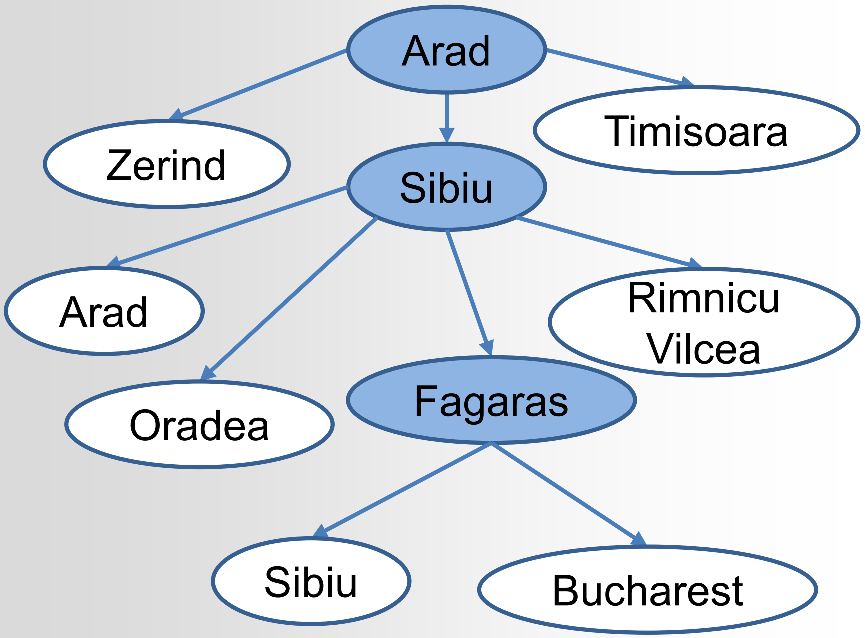


The problem is to exchange the respective positions of the white and black stones, moving them one at a time according the rules above. Answer the following to solve this puzzle using a problem-solving approach:

(a) Give a *well-defined formulation* of the problem in terms of states, operators, goal test predicate, and path cost.

Search Algorithms

- Exploration of state space by generating successors of already-explored states
  - Frontier: candidate nodes for expansion
  - Explored set



Search Strategies

- A 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	How long does it take to find a solution: the number of nodes generated
Space Complexity	Maximum number of nodes in memory
Optimality	Does it always find the best (least-cost) solution?

Uninformed vs Informed

Uninformed search strategies

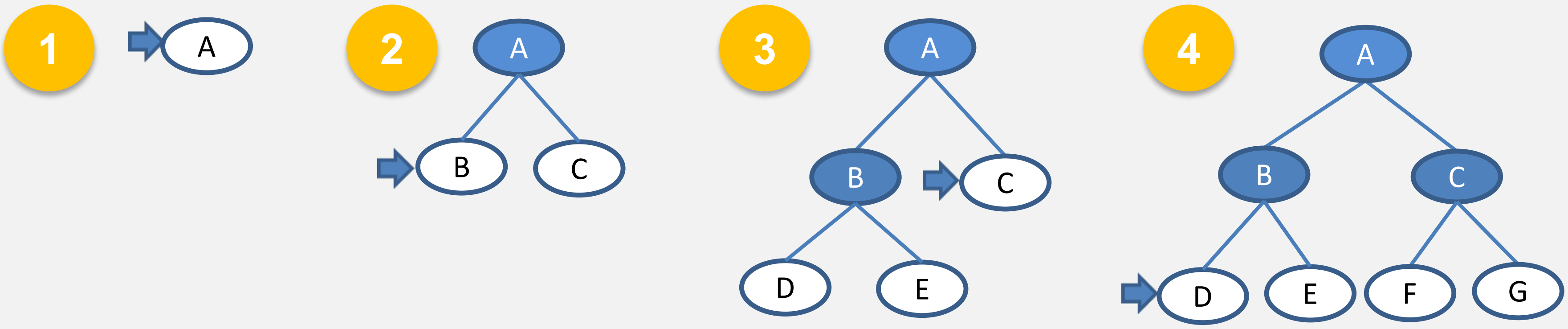
- Use only the information available in the problem definition
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth-limited search
  5. Iterative deepening search

Informed search strategies

- Use problem-specific knowledge to guide the search
- Usually more efficient

Breadth-First Search

Expand shallowest unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue



Denote

- b: maximum branching factor of the search tree
- d: depth of the least-cost solution
- Complete: Yes
- Optimal: Yes when all step costs equally

Complexity of BFS

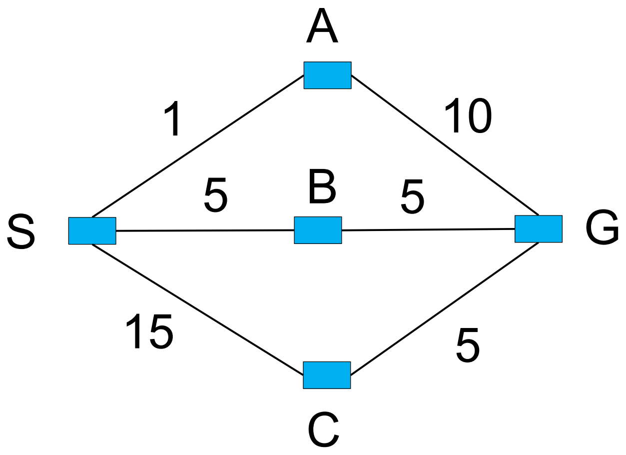
- Hypothetical state-space, where every node can be expanded into *b* new nodes, solution of path-length *d*
- Time: $1 + b + b^2$+b^3+ ... + b^d = $O(b^d)$
- Space: (keeps every node in memory) $O(b^d)$are equal

Depth	Nodes	Time		Memory	
0	1	1	millisecond	100	bytes
2	111	0.1	seconds	1	kilobytes
4	11111	11	seconds	11	kilobytes
6	10^6	18	minutes	111	megabyte
8	10^8	31	hours	11	gigabytes
10	10^{10}	128	days	1	terabyte
12	10^{12}	35	years	111	terabytes
14	10^{14}	3500	years	11111	terabytes

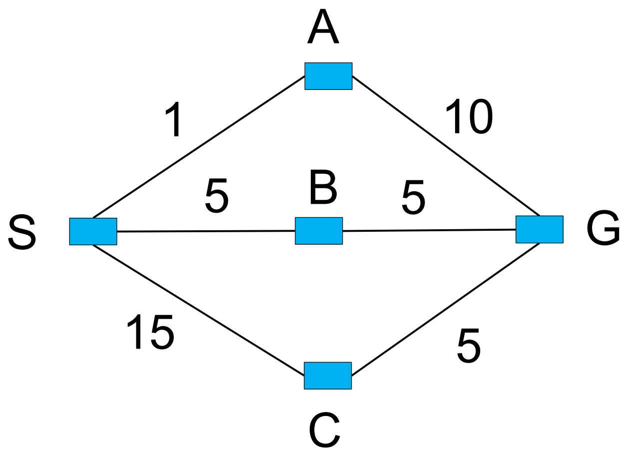
Uniform-Cost Search

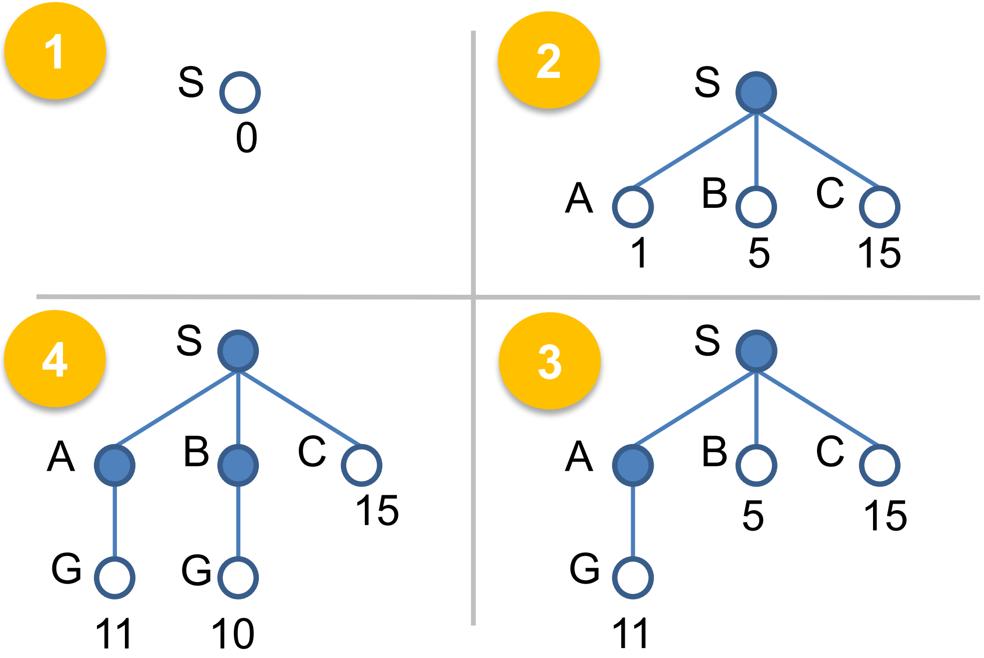
To consider edge costs, expand unexpanded node with the least path cost *g*

- Modification of breath-first search
- Instead of First-In-First-Out (FIFO) queue, using a priority queue with path cost *g(n)* to order the elements
- BFS = UCS with *g(n)* = Depth(n)



Uniform-Cost Search





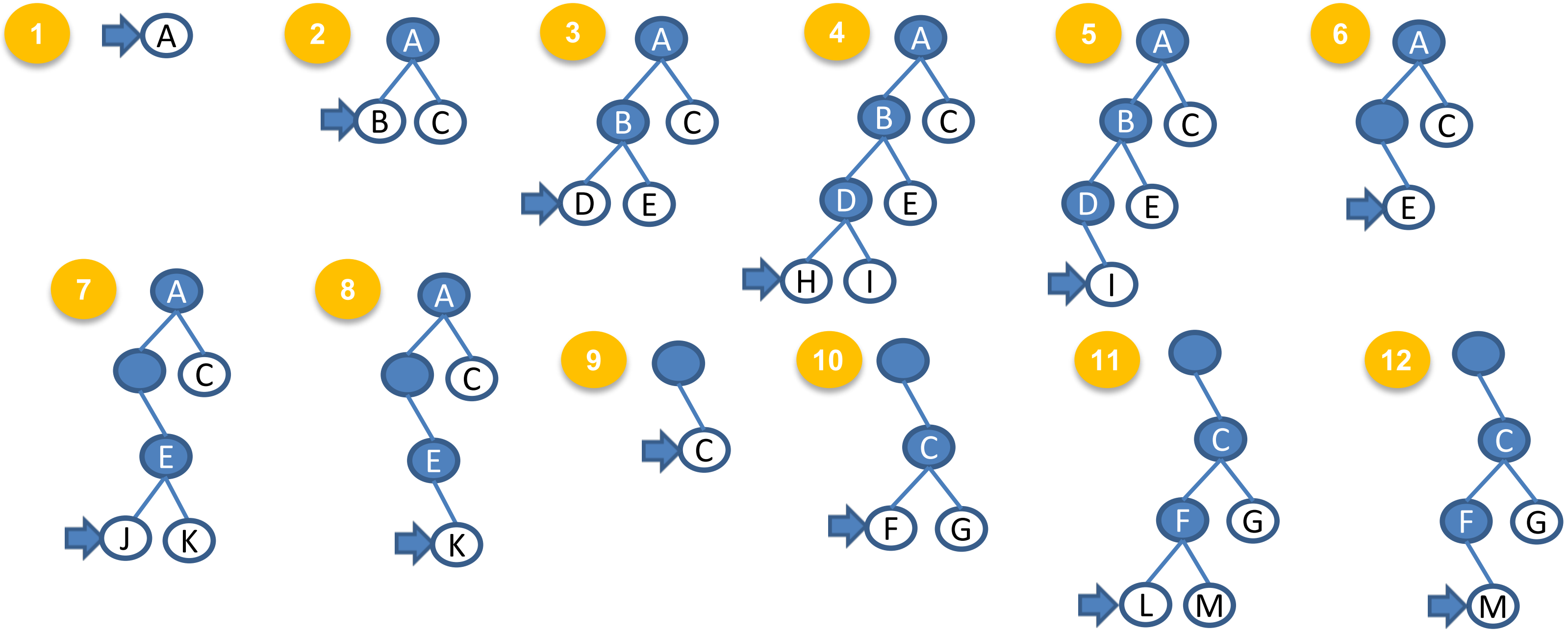
Here we do not expand notes that have been expanded.

Uniform-Cost Search

Complete	Yes
Time	# of nodes with path cost g <= cost of optimal solution (eqv. # of nodes pop out from the priority queue)
Space	# of nodes with path cost g <= cost of optimal solution
Optimal	Yes

Depth-First Search

Expand deepest unexpanded node which can be implemented by a Last-In-First-Out (LIFO) stack, Backtrack only when no more expansion



Depth-First Search

Denote

- m: maximum depth of the state space

Complete	• infinite-depth spaces: No • finite-depth spaces with loops: No • with repeated-state checking: Yes • finite-depth spaces without loops: Yes
Time	$O(b^m)$ If solutions are dense, may be much faster than breadth-first
Space	$O(bm)$
Optimal	No

Depth-Limited Search

To avoid infinite searching, Depth-first search with a cutoff on the max depth / of a path

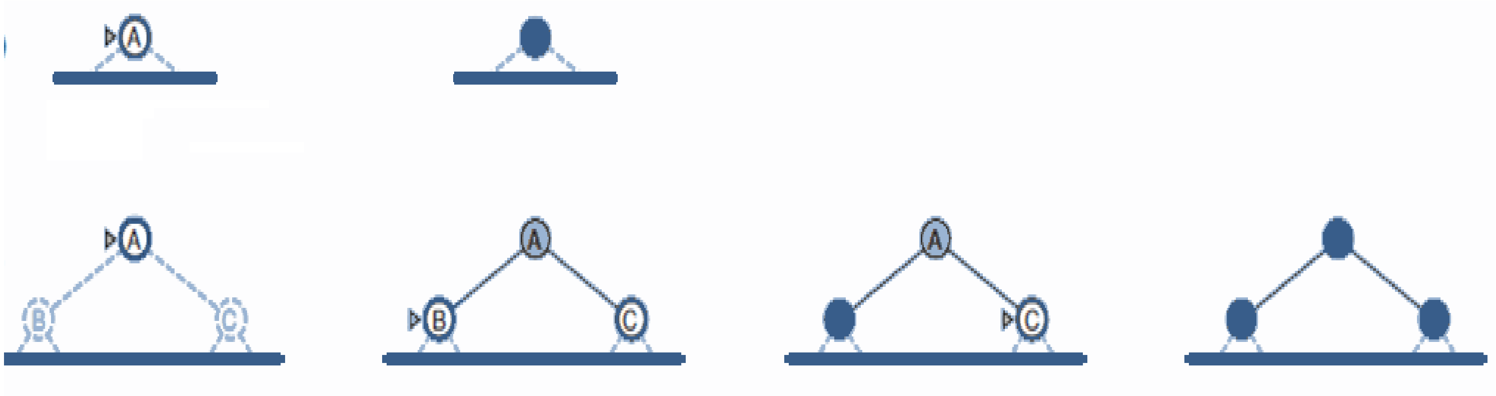
Complete	Yes, if $I \geq d$
Time	$O(b^I)$
Space	$O(bI)$
Optimal	No

Iterative Deepening Search

Iteratively estimate the max depth l of DLS one-by-one

Limit = 0

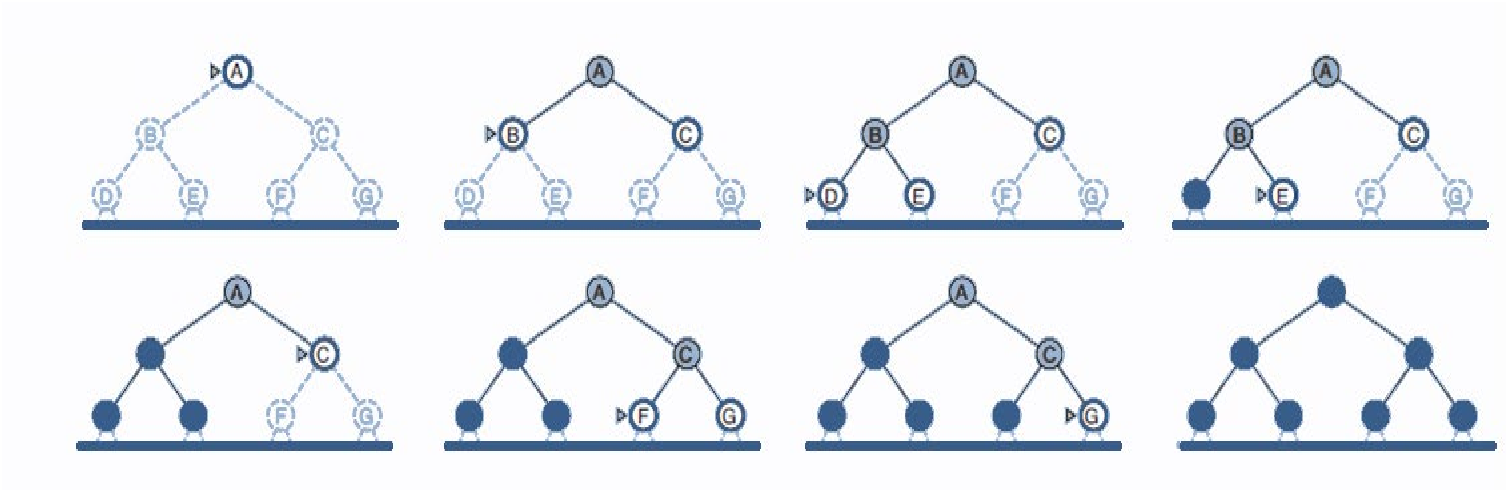
Limit = 1



Iterative Deepening Search

Iteratively estimate the max depth l of DLS one-by-one

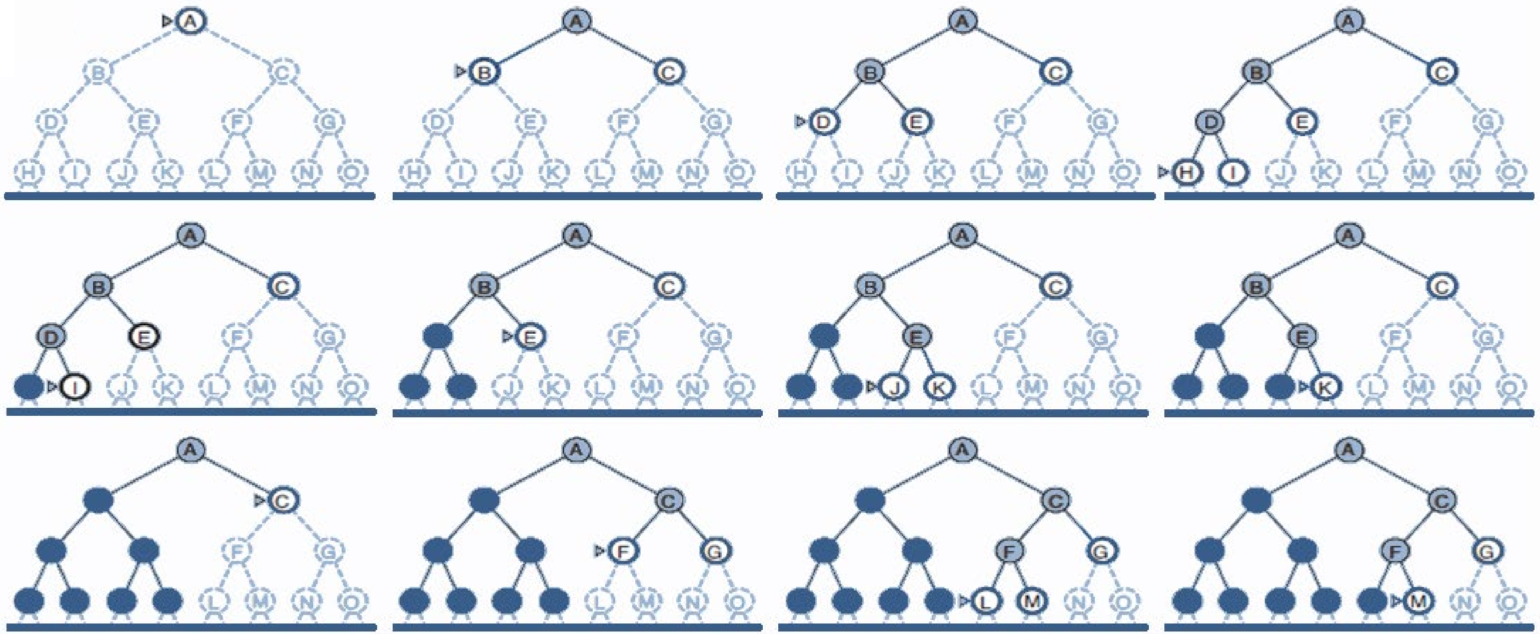
Limit = 2



Iterative Deepening Search

Iteratively estimate the max depth *l* of DLS one-by-one

Limit = 3



Iterative Deepening Search...

```
Function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution sequence
   inputs: problem, a problem

   for depth   0 to ∞ do
    if DEPTH-LIMITED-SEARCH(problem, depth) succeeds then return its result
   end
   return failure
```

Complete	Yes
Time	$O(b^d)$
Space	$O(bd)$
Optimal	Yes

Summary (we make assumptions for optimality)

Criterion	Breadth-first	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

General Search

Uninformed search strategies

- Systematic generation of new states (→Goal Test)

- Inefficient (exponential space and time complexity)

Informed search strategies

- Use problem-specific knowledge
  - To decide the order of node expansion

- Best First Search: expand the most desirable unexpanded node
  - Use an evaluation function to estimate the "desirability" of each node

Evaluation function

- Path-cost function $g(n)$
  - Cost from initial state to current state (search-node) n
  - No information on the cost toward the goal
- Need to estimate cost to the closest goal
- "Heuristic" function $h(n)$
  - Estimated cost of the cheapest path from n to a goal state $h(n)$
    - Exact cost cannot be determined
  - depends only on the state at that node
  - $h(n)$ is not larger than the real cost (admissible)

Greedy Search

Expands the node that appears to be closest to goal

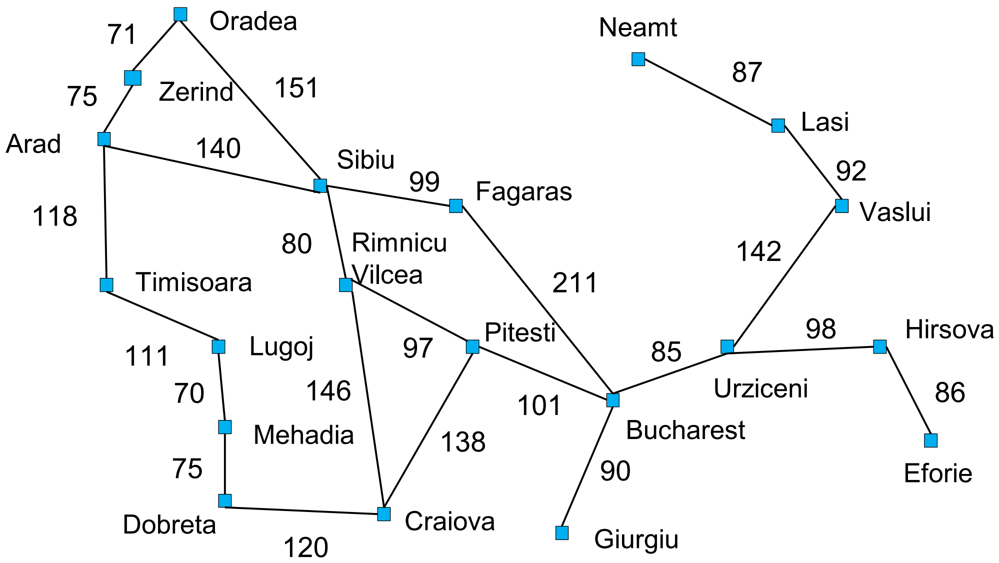
- Evaluation function $h(n)$:estimate of cost from n to $goal$
- Function Greedy-Search(problem) returns solution
  - Return Best-First-Search(problem, h)   // $h(goal) = 0$

Question: How to estimation the cost from n to goal?

Answer: Recall that we want to use problem-specific knowledge

Example: Route-finding from Arad to Bucharest

$h(n)$ = straight-line distance from n to Bucharest



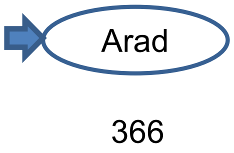
- Useful but potentially fallible (heuristic)
- Heuristic functions are problem-specific

Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example

a) The initial state

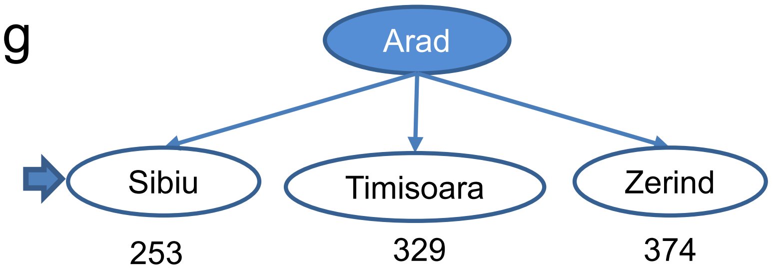


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example

b) After expanding Arad

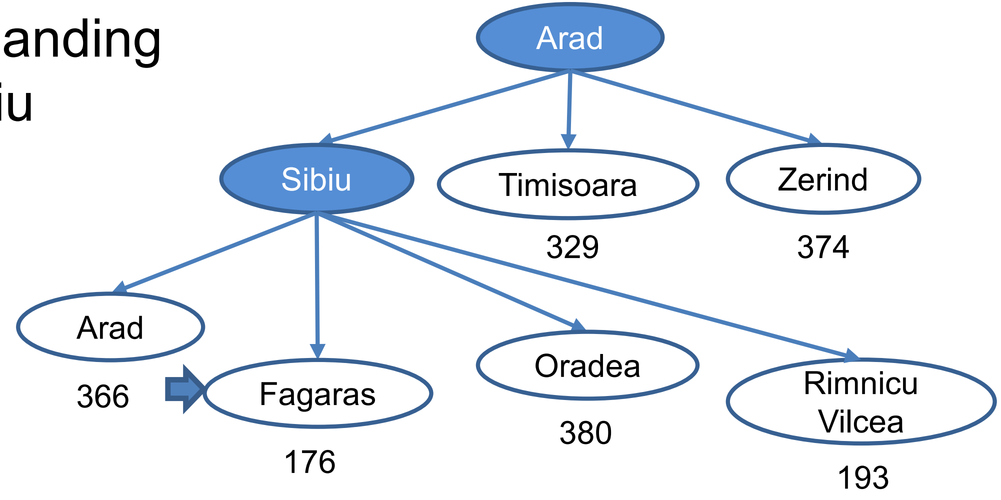


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example

c) After expanding Sibiu

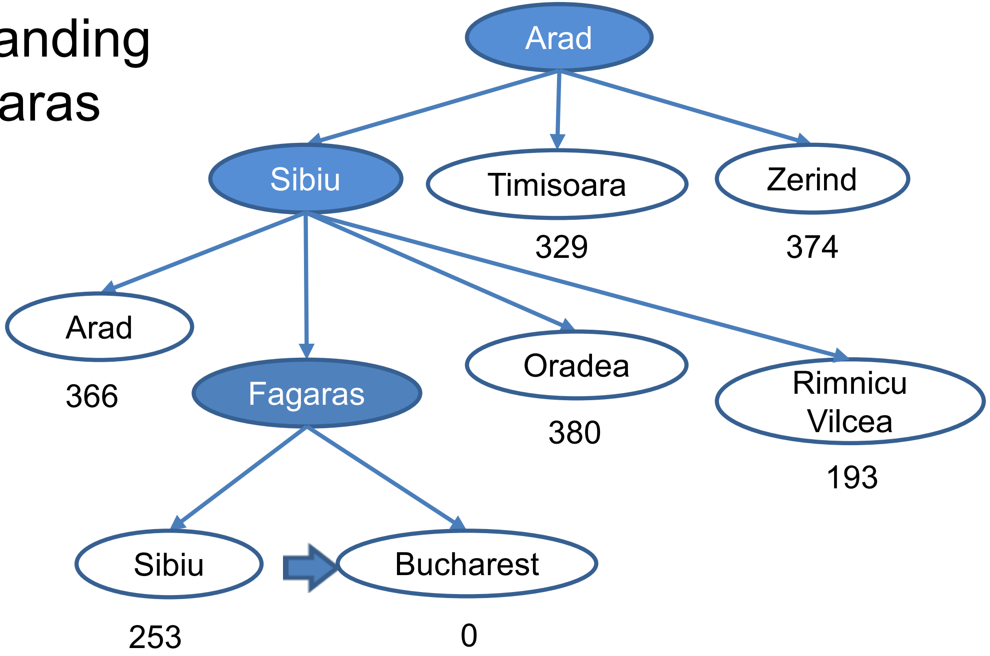


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example

d) After expanding Fagaras



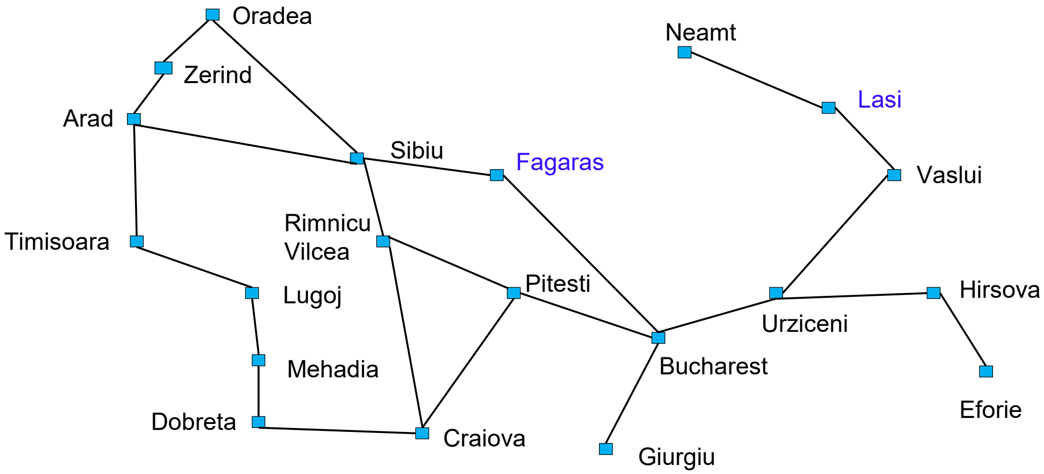
Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Complete?

Question: Is this approach complete?

Example: Find a path from Lasi to Fagaras



Answer: No

Greedy Search...

- m: maximum depth of the search space

Complete	No
Time	$O(b^m)$
Space	$O(b^m)$(keeps all nodes in memory)
Optimal	No

A * Search

- Uniform-cost search
  - *g(n):* cost to reach n (Past Experience)
  - optimal and complete, but can be very inefficient
- Greedy search
  - *h(n):* cost from n to goal (Future Prediction)
  - neither optimal nor complete, but cuts search space considerably

A * Search

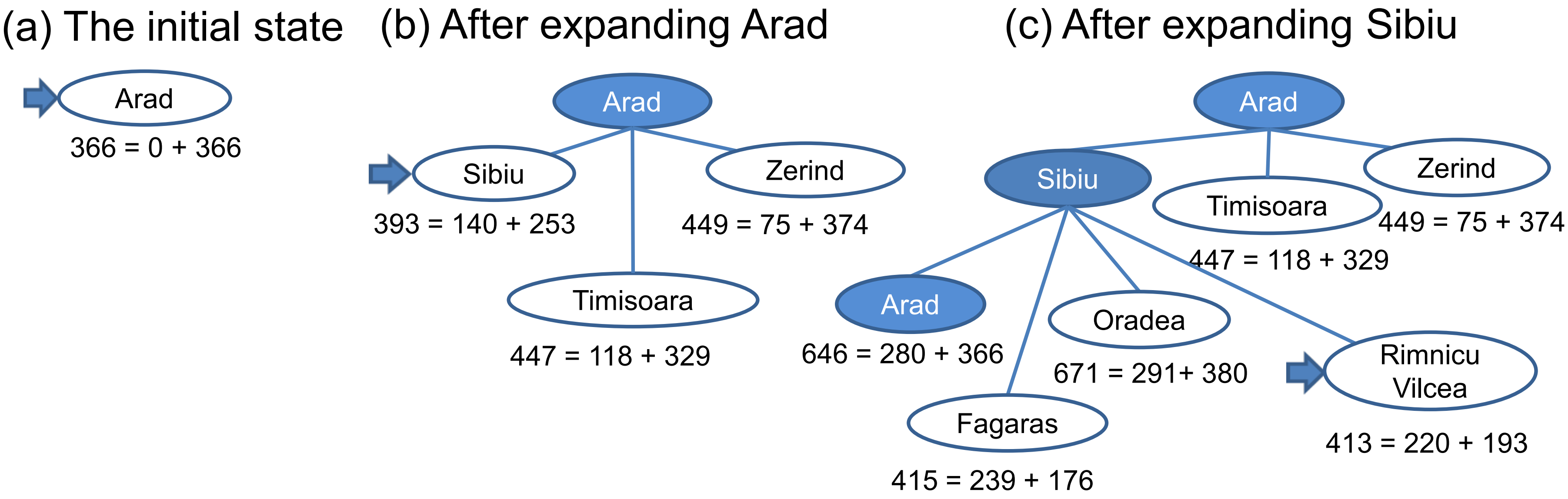
Idea: Combine Greedy search with Uniform-Cost search

Evaluation function: $f(n) = g(n) + h(n)$

- *f (n):* estimated total cost of path through *n* to goal (Whole Life)
- If g = 0 → greedy search; If h = 0 → uniform-cost search
- Function A* Search(problem) returns solution
  - Return Best-First-Search(problem, $g + h$)

Example: Route-finding from Arad to Bucharest

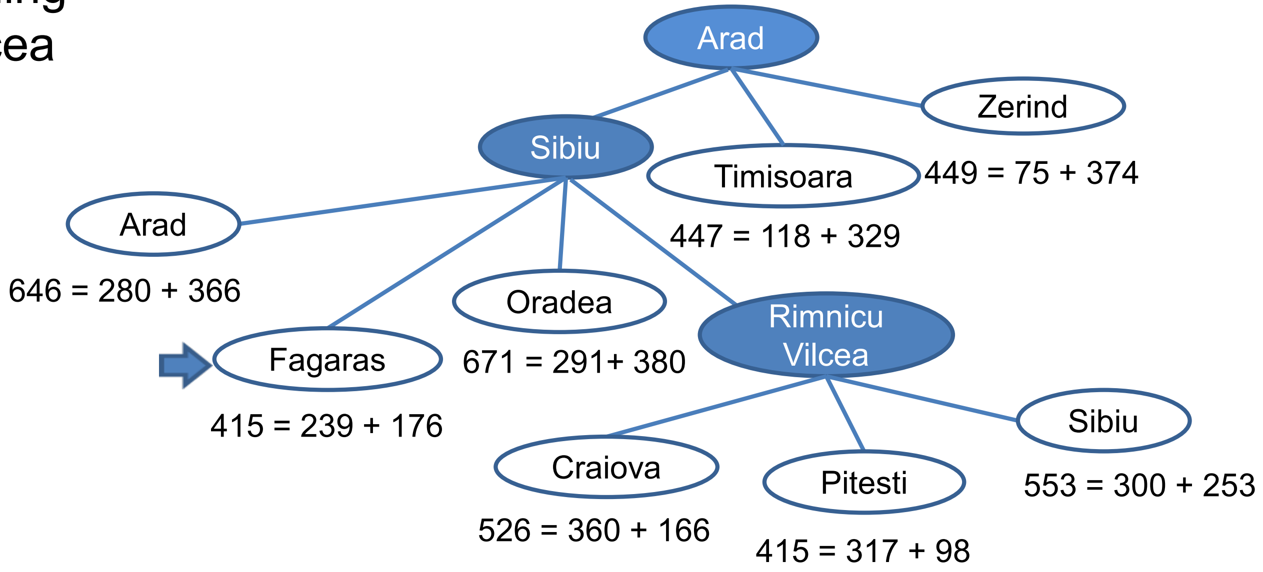
Best-first-search with evaluation function $g + h$



Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

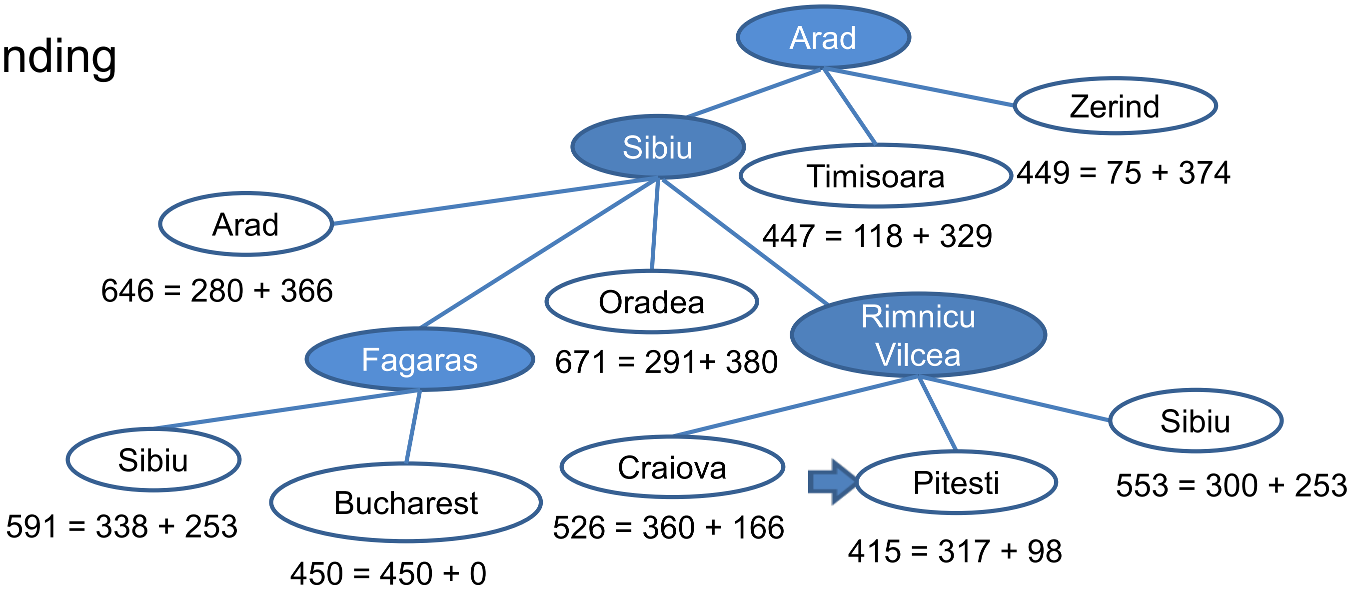
(d) After expanding
Rimnicu Vilcea



Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

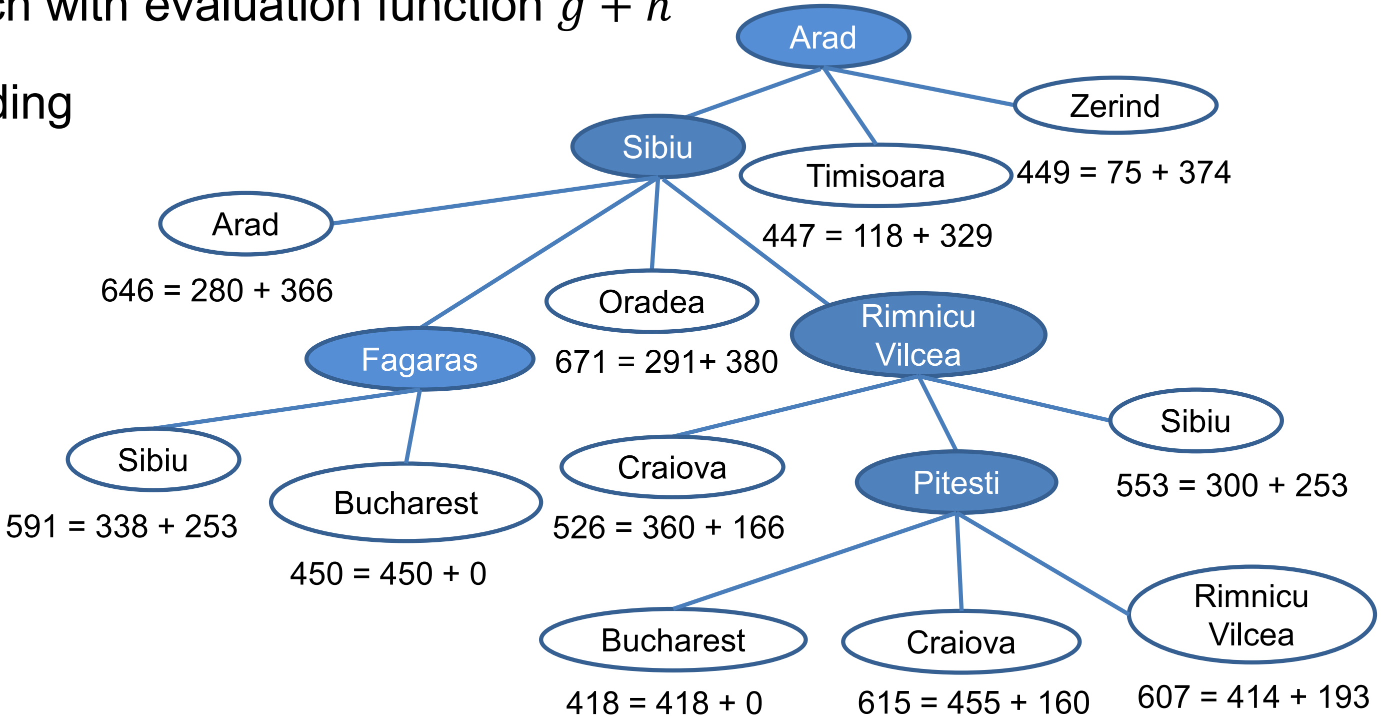
(e) After expanding
Fagaras



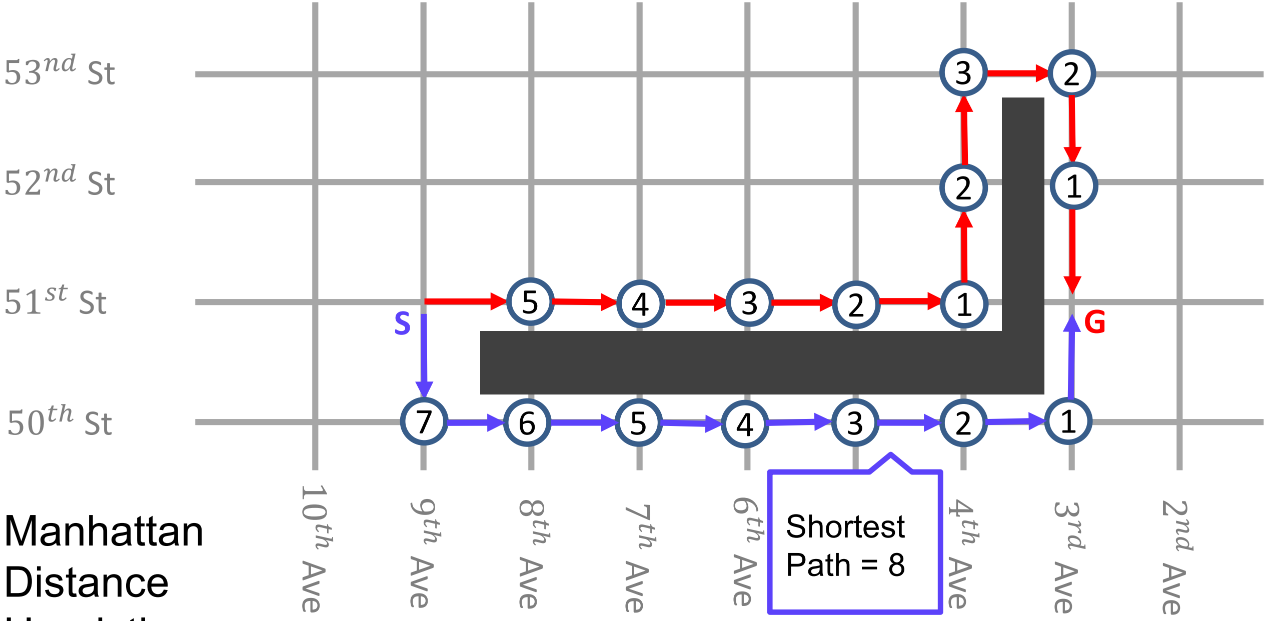
Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

(f) After expanding Pitesti

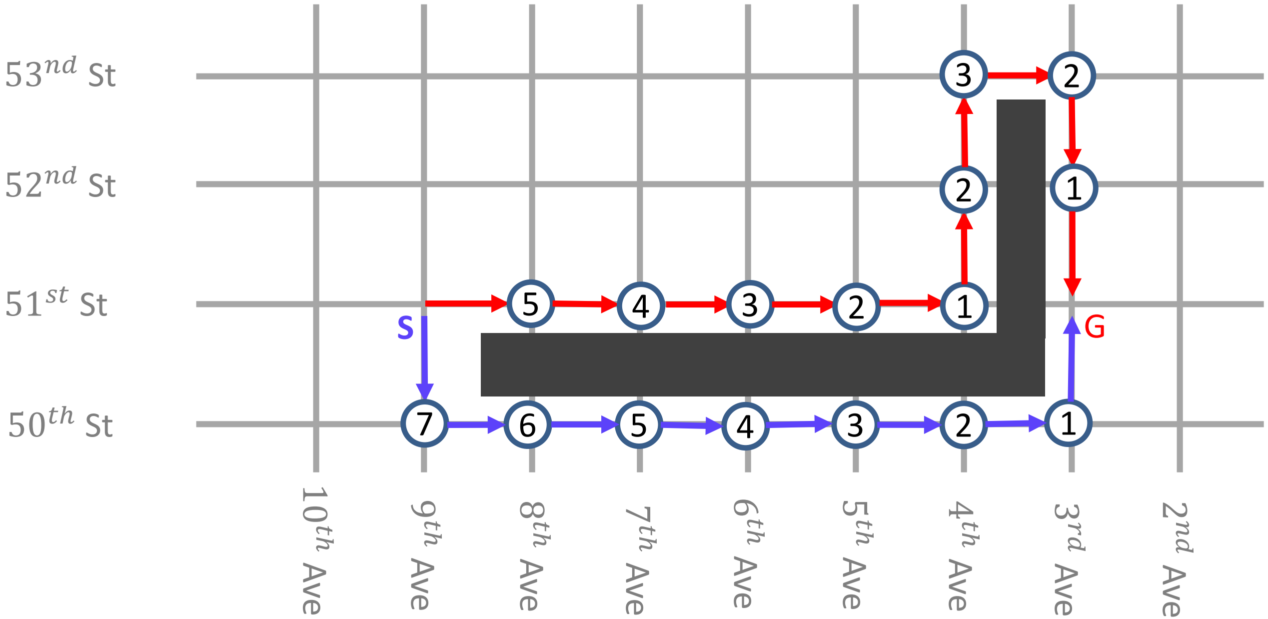


Example: Route-finding in Manhattan

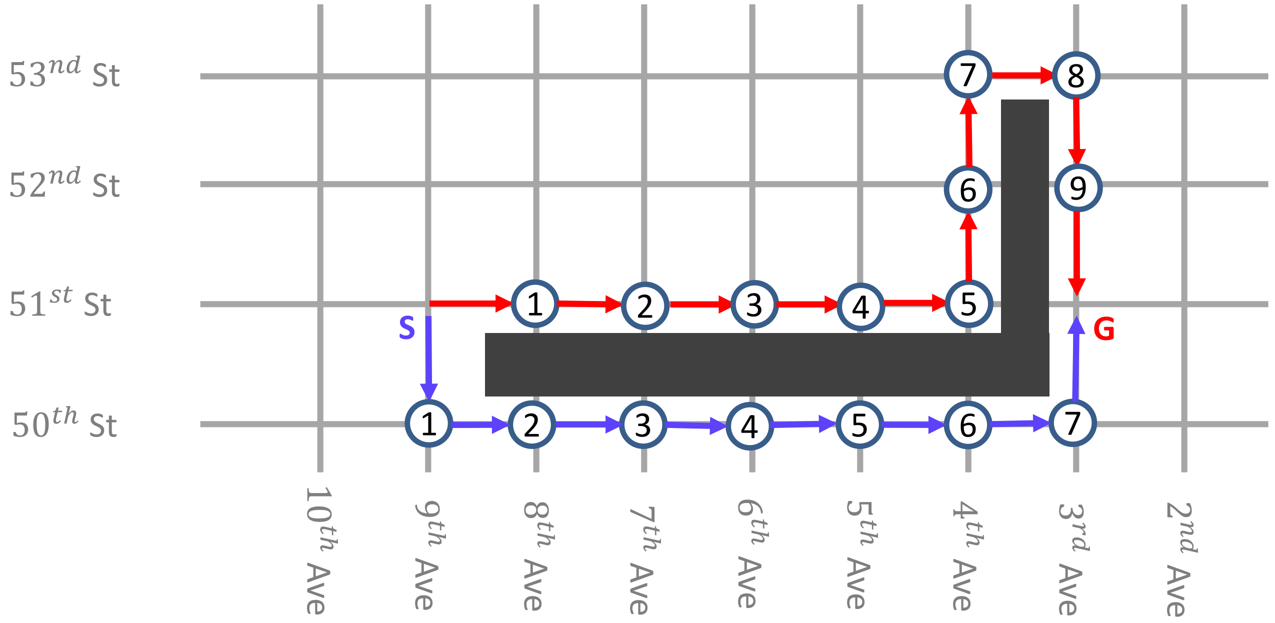


Manhattan Distance Heuristic

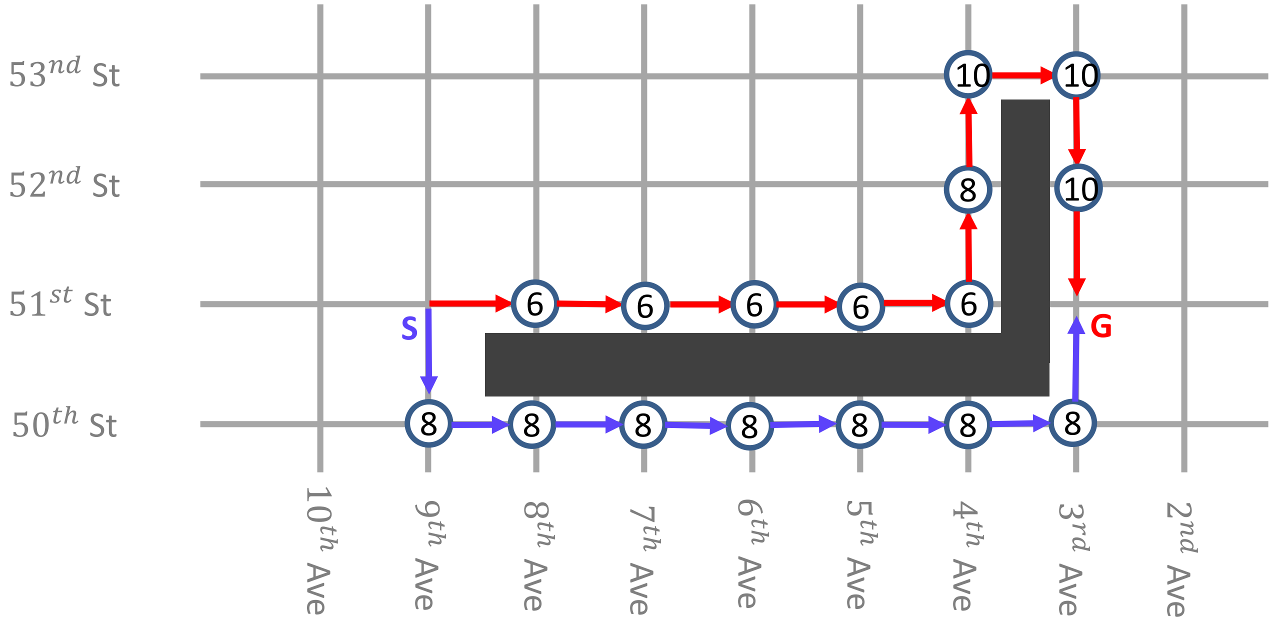
Example: Route-finding in Manhattan (Greedy)



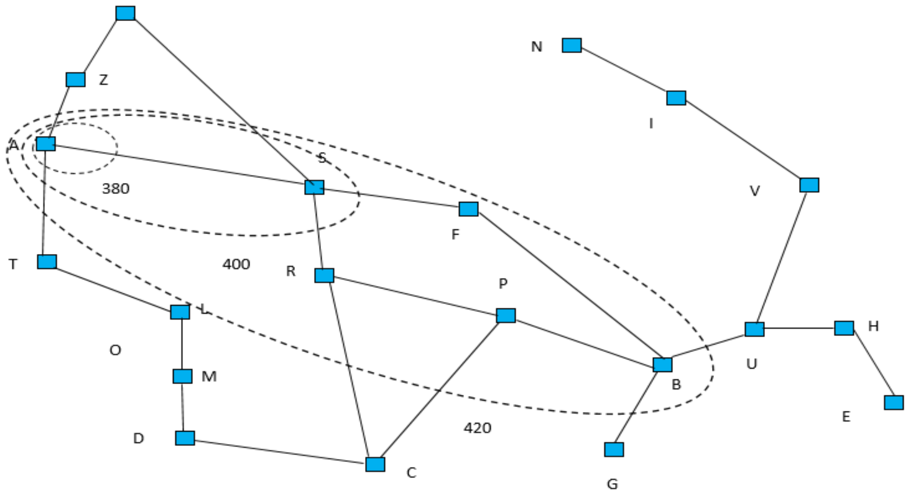
Example: Route-finding in Manhattan (UCS)



Example: Route-finding in Manhattan (A*)



Complexity of A*



Time	Exponential in length of solution
Space	(all generated nodes are kept in memory) Exponential in length of solution

With a good heuristic, significant savings are still possible compared to uninformed search methods