Zurich University
of Applied Sciences

Artificial Intelligence
V03: Problem solving through search

Searching as a problem solving strategy
Uninformed search
Heuristic (informed) search

Based on material by
- Stuart Russell, UC Berkeley
- Inês de Castro Dutra, Cooperating Intelligent Systems, U. Porto

University of Porto

Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

Educational objectives

- **Know classical search algorithms** and **selection criteria** based on time and space **complexity**
- **Understand** how **intelligent behavior** evolves **out of efficient algorithms**
- **Know how** to inform search methods by **heuristics**
- **Be able to model** a real world problem to be solved by searching

"In which we see how an agent can find a sequence of actions that achieves its goals when no single action will do."

➔ Reading: AIMA, ch. 3

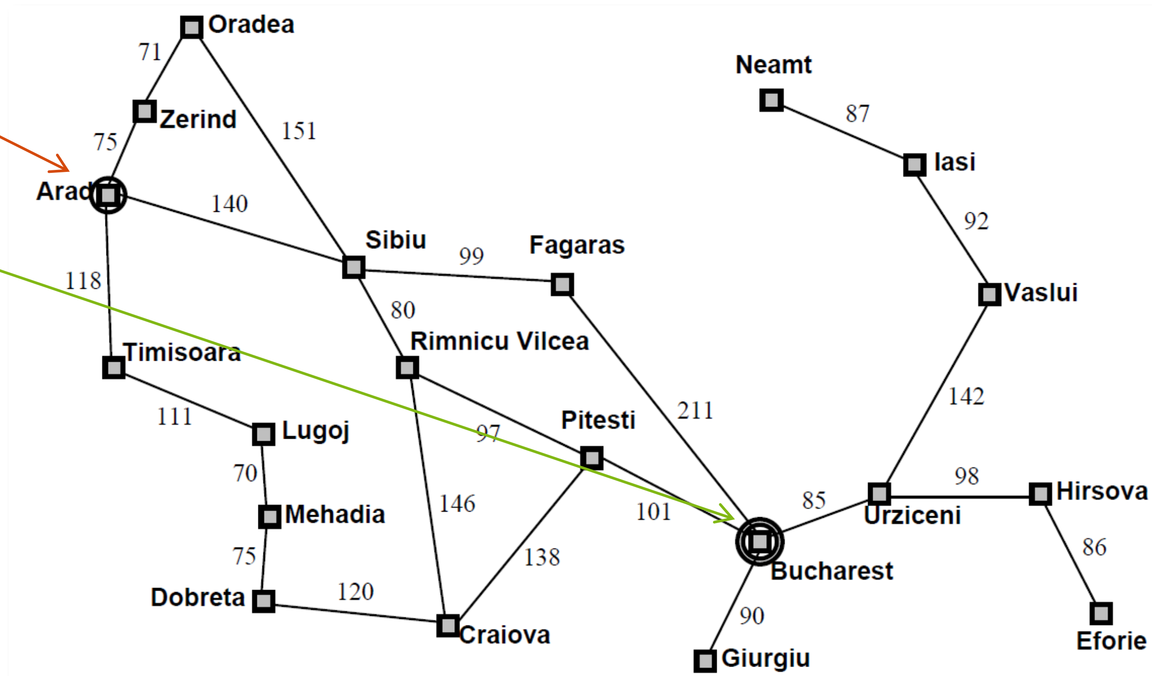
1. SEARCHING AS A PROBLEM SOLVING STRATEGY

Example: On holiday in Romania
Task: Catch flight that leaves tomorrow from Bucharest

Initial state
- Currently in Arad

Formulate goal
- be in Bucharest

Formulate problem
- states: various cities
- actions: drive between cities

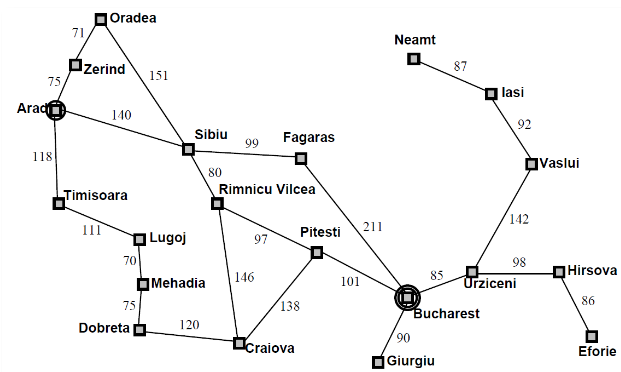


Find solution
- sequence of cities
  e.g., Arad→Sibiu→Fagaras→Bucharest

Problem formulation
For deterministic & fully observable environments

Problem is defined by four items

- **initial state**
  e.g., `In(Arad)`
- **successor function** `S(x)`
  set of action-state pairs, e.g.
  `S(Arad) = {<Arad→Zerind;Zerind>, …}`
- **goal test**
  explicit or implicit, e.g.
  `x = In(Bucharest)` or `NoDirt(x)`
- **path cost** (additive)
  e.g., sum of distances, number of actions, etc.
  `c(x,a,y)>=0` is the step cost



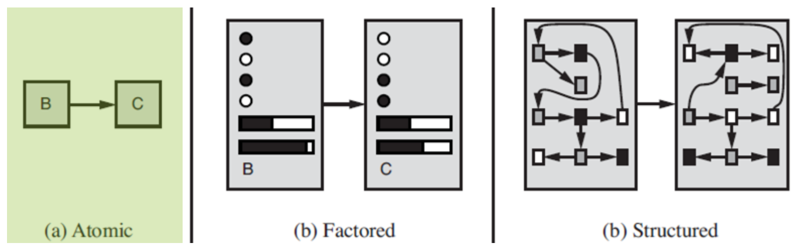
Selecting a proper state space

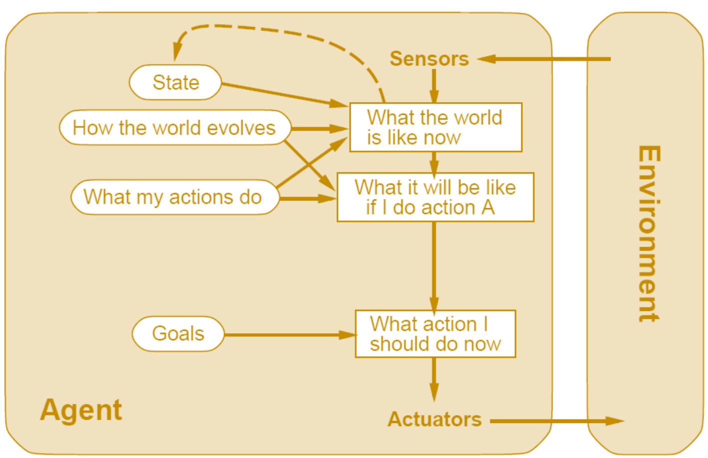
- Real world is very complex
  ➔ state & action space must be abstracted
- Abstract state: **set of** real states
- Abstract action: complex **combination of** real actions
  e.g., `Arad→Zerind` represents a complex set of possible routes, detours, rest stops, etc.
- Abstract solution: **set of real paths** that are solutions in the real world
- For guaranteed realizability, **any** real state `In(Arad)` must get to some real state `In(Zerind)`

➔ See also appendix on modeling

➔ Each abstract action **should be easier** than the original problem

Suitable agent structure






```
function Simple-Problem-Solving-Agent(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation
    state ← Update-State(state, percept)
    if seq is empty then
        goal ← Formulate-Goal(state)
        problem ← Formulate-Problem(state, goal)
        seq ← Search(problem)
        action ← Recommendation(seq, state)
    seq ← Remainder(seq, state)
    return action
```

Note: this is **offline** problem solving; solution executed “eyes closed“.

➔ If the **task is** represented as a **graph of atomic states**, and the **solution is** a **sequence** of state changes → a **model based agent** may solve it by **searching**

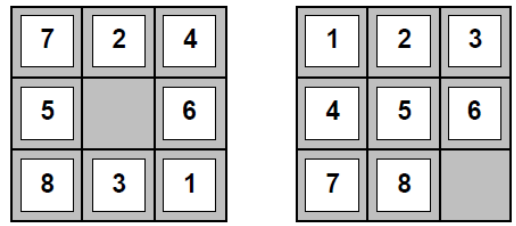
Examples of problems solvable by searching

“Toy” problem: helps to identify strengths and weaknesses of different methods

8-puzzle

Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States?
- Actions?
- Goal test?
- Path cost?

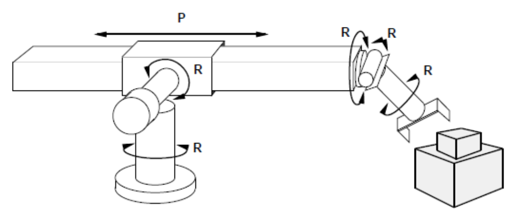


Start State Goal State

Real-world problem

Robotic assembly

- States?
- Actions?
- Goal test?
- Path cost?

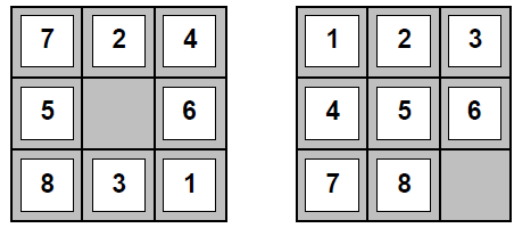


Examples of problems solvable by searching

“Toy” problem: helps to identify strengths and weaknesses of different methods

8-puzzle Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States? integer locations of tiles (ignoring intermediate positions)
- Actions?
- Goal test?
- Path cost?

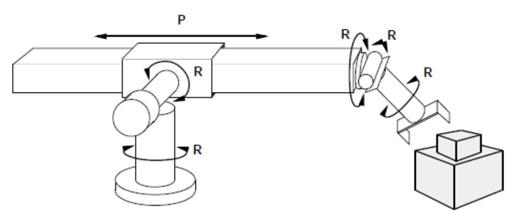


Start State Goal State

Real-world problem

Robotic assembly

- States?
- Actions?
- Goal test?
- Path cost?



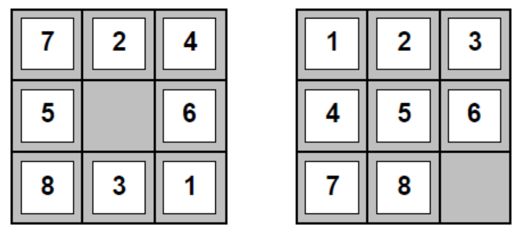
Examples of problems solvable by searching

"Toy" problem: helps to identify strengths and weaknesses of different methods

8-puzzle

Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States? integer locations of tiles (ignoring intermediate positions)
- Actions? move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test?
- Path cost?

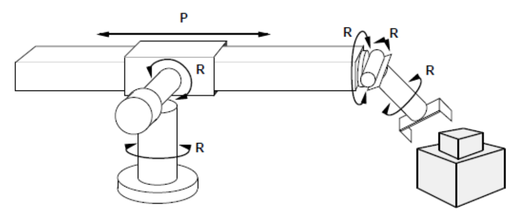


Start State   Goal State

Real-world problem

Robotic assembly

- States?
- Actions?
- Goal test?
- Path cost?

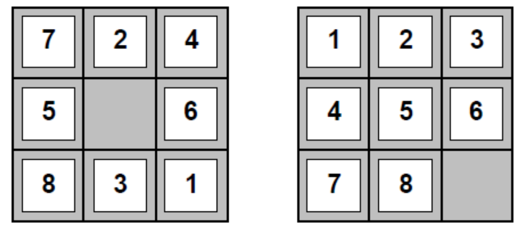


Examples of problems solvable by searching

“Toy” problem: helps to identify strengths and weaknesses of different methods

8-puzzle — Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States? — integer locations of tiles (ignoring intermediate positions)
- Actions? — move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test? — equals given goal state
- Path cost?

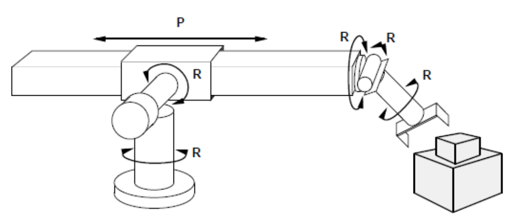


Start State Goal State

Real-world problem

Robotic assembly

- States?
- Actions?
- Goal test?
- Path cost?

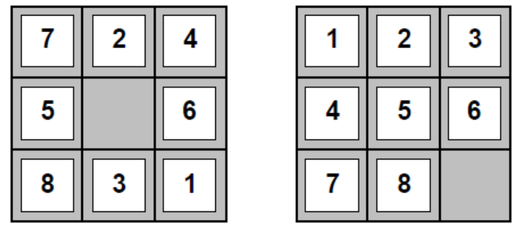


Examples of problems solvable by searching

“Toy” problem: helps to identify strengths and weaknesses of different methods

8-puzzle Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States? integer locations of tiles (ignoring intermediate positions)
- Actions? move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test? equals given goal state
- Path cost? 1 per move

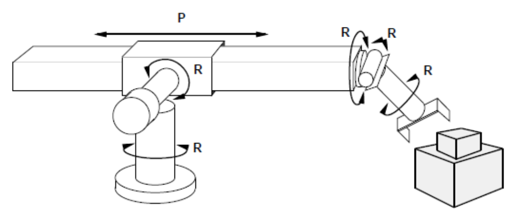


Start State Goal State

Real-world problem

Robotic assembly

- States?
- Actions?
- Goal test?
- Path cost?

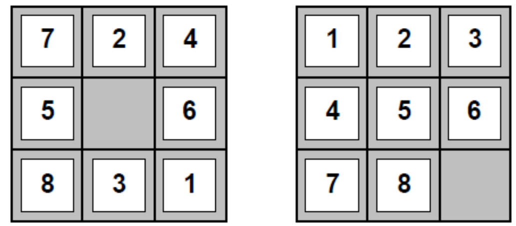


Examples of problems solvable by searching

"Toy" problem: helps to identify strengths and weaknesses of different methods

8-puzzle — Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States? — integer locations of tiles (ignoring intermediate positions)
- Actions? — move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test? — equals given goal state
- Path cost? — 1 per move

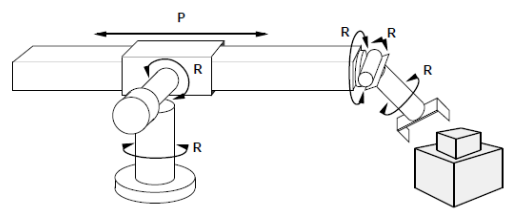


Start State — Goal State

Real-world problem

Robotic assembly

- States? — real-valued coordinates of robot joint angles; parts to be assembled
- Actions?
- Goal test?
- Path cost?



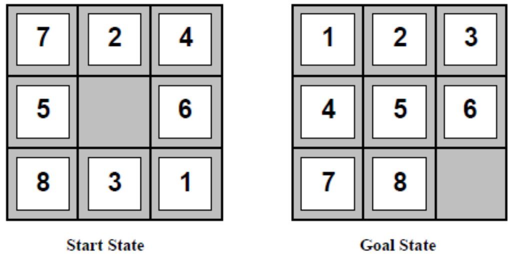
?

Examples of problems solvable by searching

"Toy" problem: helps to identify strengths and weaknesses of different methods

8-puzzle Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

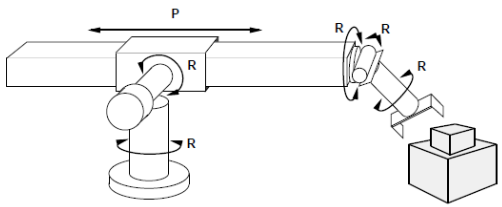
- States? integer locations of tiles (ignoring intermediate positions)
- Actions? move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test? equals given goal state
- Path cost? 1 per move



Real-world problem

Robotic assembly

- States? real-valued coordinates of robot joint angles; parts to be assembled
- Actions? continuous motions of robot joints
- Goal test?
- Path cost?

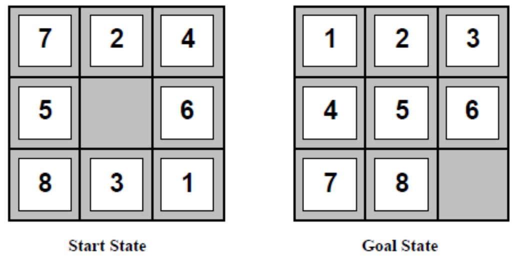


Examples of problems solvable by searching

“Toy” problem: helps to identify strengths and weaknesses of different methods

8-puzzle — Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

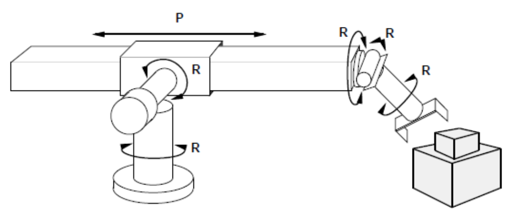
- States? integer locations of tiles (ignoring intermediate positions)
- Actions? move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test? equals given goal state
- Path cost? 1 per move



Real-world problem

Robotic assembly

- States? real-valued coordinates of robot joint angles; parts to be assembled
- Actions? continuous motions of robot joints
- Goal test? complete assembly
- Path cost?

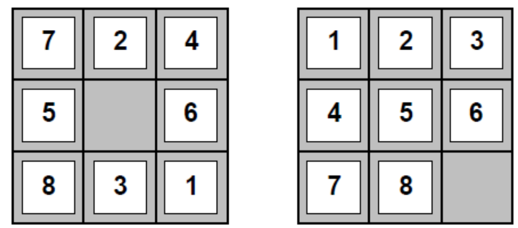


Examples of problems solvable by searching

“Toy” problem: helps to identify strengths and weaknesses of different methods

8-puzzle — Note: Optimal solution of n-Puzzle family is NP-hard (→ see appendix)

- States? — integer locations of tiles (ignoring intermediate positions)
- Actions? — move blank to left, right, up, down (ignoring unjamming etc.)
- Goal test? — equals given goal state
- Path cost? — 1 per move

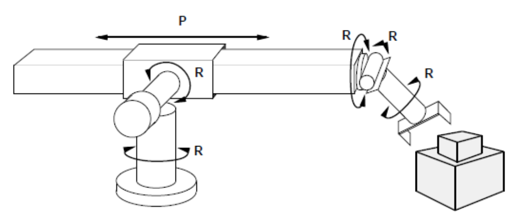


Start State — Goal State

Real-world problem

Robotic assembly

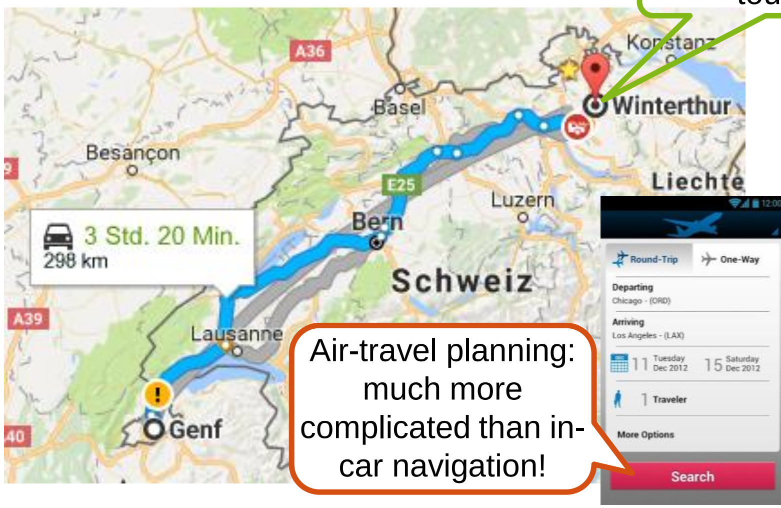
- States? — real-valued coordinates of robot joint angles; parts to be assembled
- Actions? — continuous motions of robot joints
- Goal test? — complete assembly
- Path cost? — execution time



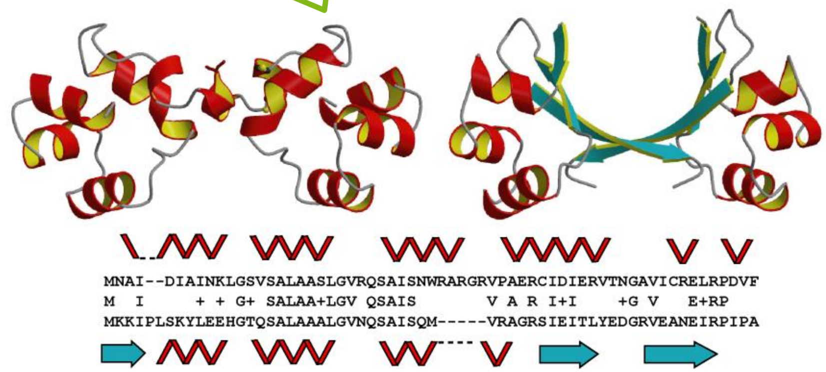
Other real-world problems

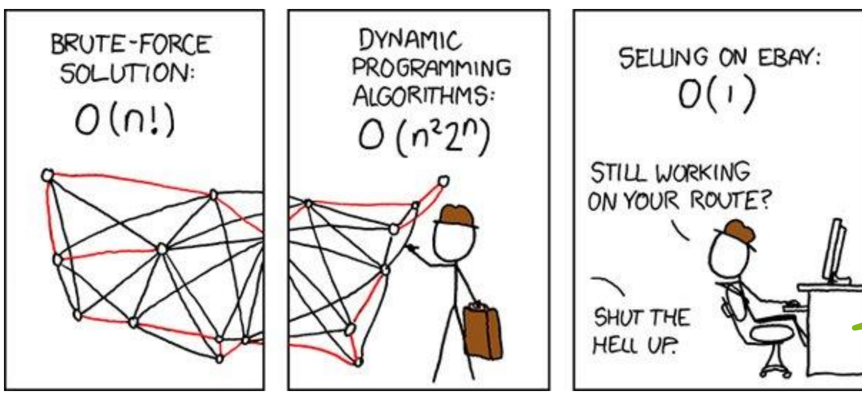
Protein design: find a sequence of amino acids that folds into a structure with certain properties

Route-finding (incl. touring)



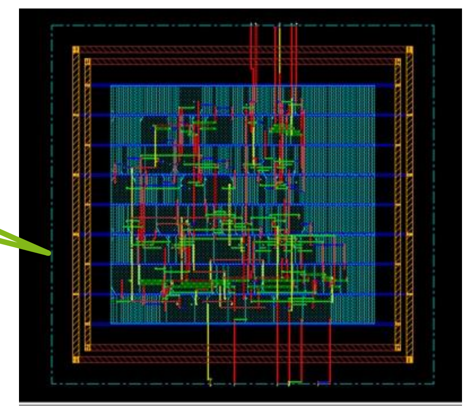
Air-travel planning: much more complicated than in-car navigation!





VLSI layout: place components and optimize wiring

All TSP-related problems of finding a shortest path



Diversity of search approaches
…solving increasingly complex problem types

→ this lecture

Uninformed (blind) search
- **All it can do**: **generate successors of tree-nodes**, distinguish goal- from non-goal states
- Suitable environments: **fully observable, deterministic, discrete** (episodic, static, single agent)

Extensions of today's methods exist to **non-deterministic** and **partially observable** as well as **(semi-)dynamic** environments (**online** search) (→ see AIMA, ch. 4.3-4.5)

Heuristic (informed) search
- **Knows whether** one non-goal **state is "more promising"** than another
- Suitable environments: as above, but **larger**

More informed search methods

Online search
- Environments are **dynamic** (i.e., not fully known from the beginning ➔ percepts become important)

→ next lecture

Local search
- Cares only to **find a goal** state **rather then** the optimal **path**
- Suitable environments: also **continuous** state/action spaces (hill climbing, simulated annealing)

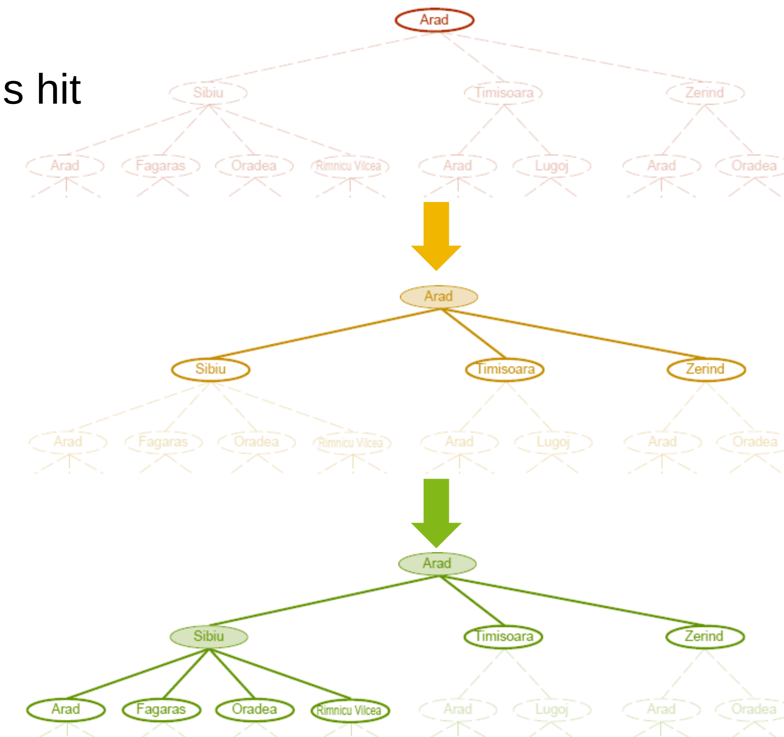
Adversarial search
- Search in the face of an opponent (i.e., **dynamic multi-agent** environments; also **stochastic** and **partially observable** forms)

2. UNINFORMED SEARCH

Uninformed search

Approach

- Tree search: iteratively **expand nodes** until a goal node is hit
- Different strategies: **order** of node expansion



Evaluation criteria for strategies

- completeness: does it **always find** a **solution** if one exists?
- optimality: does it always find a **least-cost solution**?
- time complexity: **number of nodes** generated/expanded
- space complexity: maximum number of **nodes in memory**

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 ∞)

Zurich University of Applied Sciences

Example

Growth of time and memory requirements

- Algorithm: breadth-first search (→ ADS: exponential time & space complexity $O(b^d)$)
  Assumptions: $b = 10$, 1 mio nodes/sec, 1 kB/node
  Question: what d is easily manageable?

➔ See appendix for some **recap** on **complexity theory**

Example

Growth of time and memory requirements

- Algorithm: breadth-first search (→ ADS: exponential time & space complexity $O(b^d)$)
  Assumptions: $b = 10$, 1 mio nodes/sec, 1 kB/node
  Question: what d is easily manageable?

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

➔ Practical advice: **Exponential-complexity** search problems **cannot be solved by uninformed methods** for any but the smallest instances

➔ See appendix for some **recap** on **complexity theory**

Uninformed search strategies

→ Details: ADS or AIMA ch. 3.4

Expand the shallowest unexpanded node | Expand node with lowest path cost $g(n)$ | Expand deepest node | DFS only up to level l | Try DLS with $l = 1, l = 2$, ... until goal is reached

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

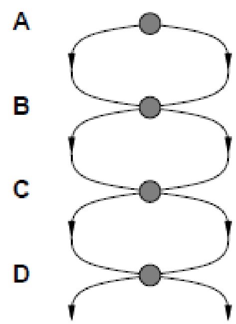
Practical advice

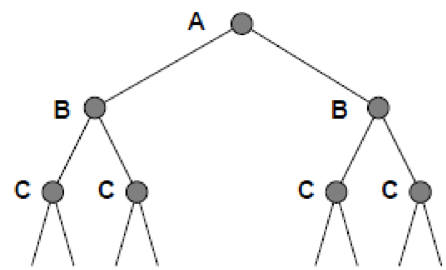
- **Depth-first tree search** is a **major work horse** for many AI tasks (due to linear space complexity)
- **Iterative deepening** is **not wasteful** (a tree with nearly the same b at each level has most nodes in the bottom level → generating higher-level states multiple times doesn't matter)
- **Iterative deepening** is **preferred uninformed** search **method** (for large search space and d is unknown)
- **Bi-directional search** can **help** a lot, but $O(b^{d/2})$ space complexity is major drawback

Repeated states

Problem

- **Failure to detect repeat**ed states can turn a linear problem into an **exponential** one!





Solution

- **Graph search**: remember nodes already expanded, and **don't revisit** them
  ➔ keep a list of explored nodes

Practical advice

- **All** previous **strategies can be implemented** as both tree- or graph search
- If **additional space complexity** is affordable determines whether graph search is possible

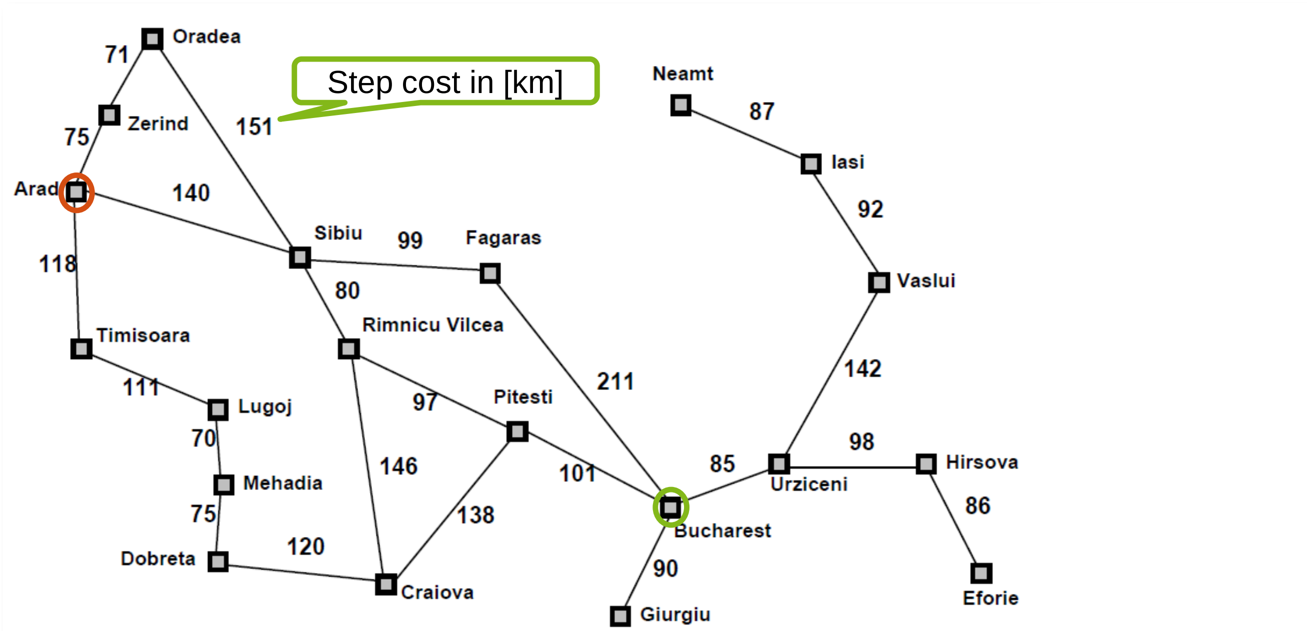
3. HEURISTIC (INFORMED) SEARCH

Tree-/graph search using additional knowledge
…beyond the definition of the problem

Best-first search

- **Select** the node to be expanded next **based on** some **evaluation function** $f(node)$
- Typically, $\boldsymbol{f}$ **is** implemented by a **heuristic** $h(node)$ (measure of "desirability")
- $h(node)$ facilitates pruning of the search tree: options are eliminated without examination

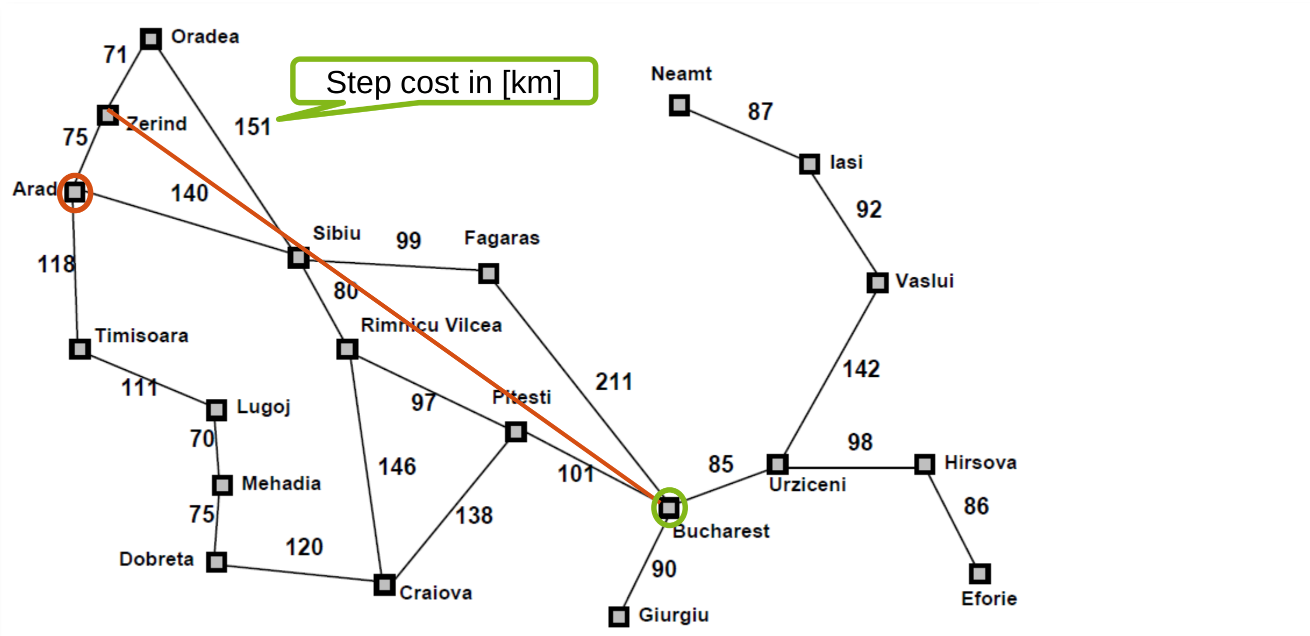
What could be a **good heuristic** for the distance to Bucharest (being in Arad)**?**



Tree-/graph search using additional knowledge
…beyond the definition of the problem

Best-first search
- **Select** the node to be expanded next **based on** some **evaluation function** $f(node)$
- Typically, $\boldsymbol{f}$ **is** implemented by a **heuristic** $h(node)$ (measure of “desirability”)
- $h(node)$ facilitates pruning of the search tree: options are eliminated without examination

What could be a **good heuristic** for the distance to Bucharest (being in Arad)**?**

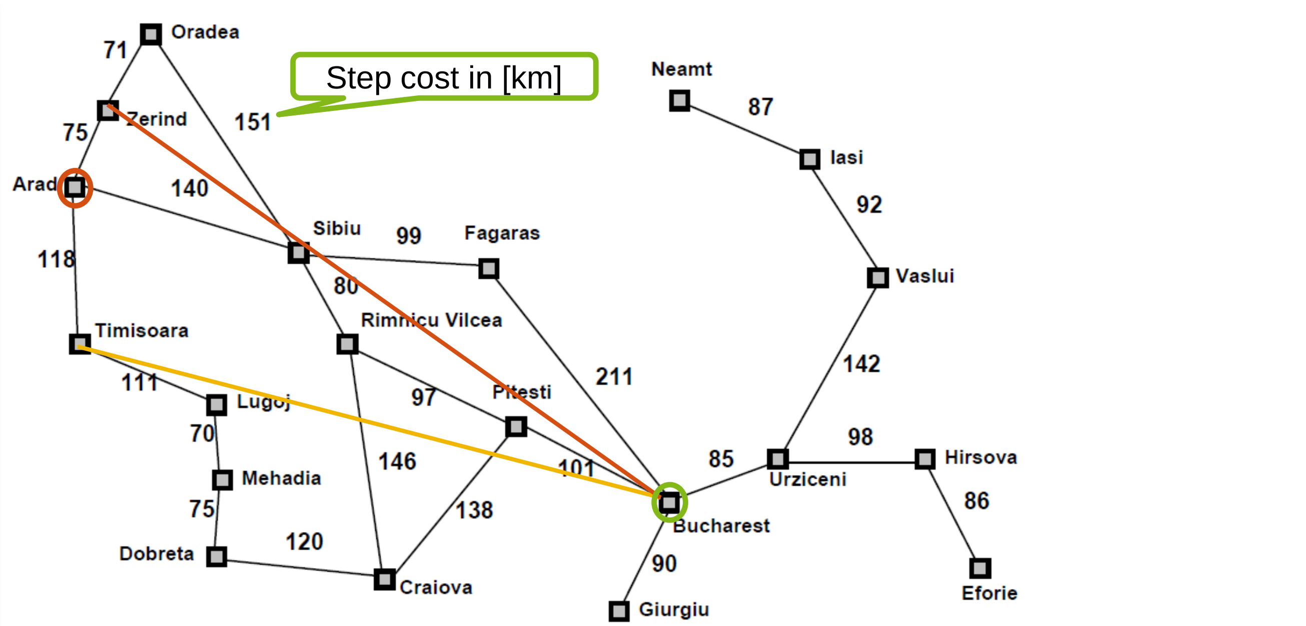


Zurich University
of Applied Sciences

Tree-/graph search using additional knowledge
…beyond the definition of the problem

Best-first search
- **Select** the node to be expanded next **based on** some **evaluation function** $f(node)$
- Typically, $\boldsymbol{f}$ **is** implemented by a **heuristic** $h(node)$ (measure of "desirability")
- $h(node)$ facilitates pruning of the search tree: options are eliminated without examination

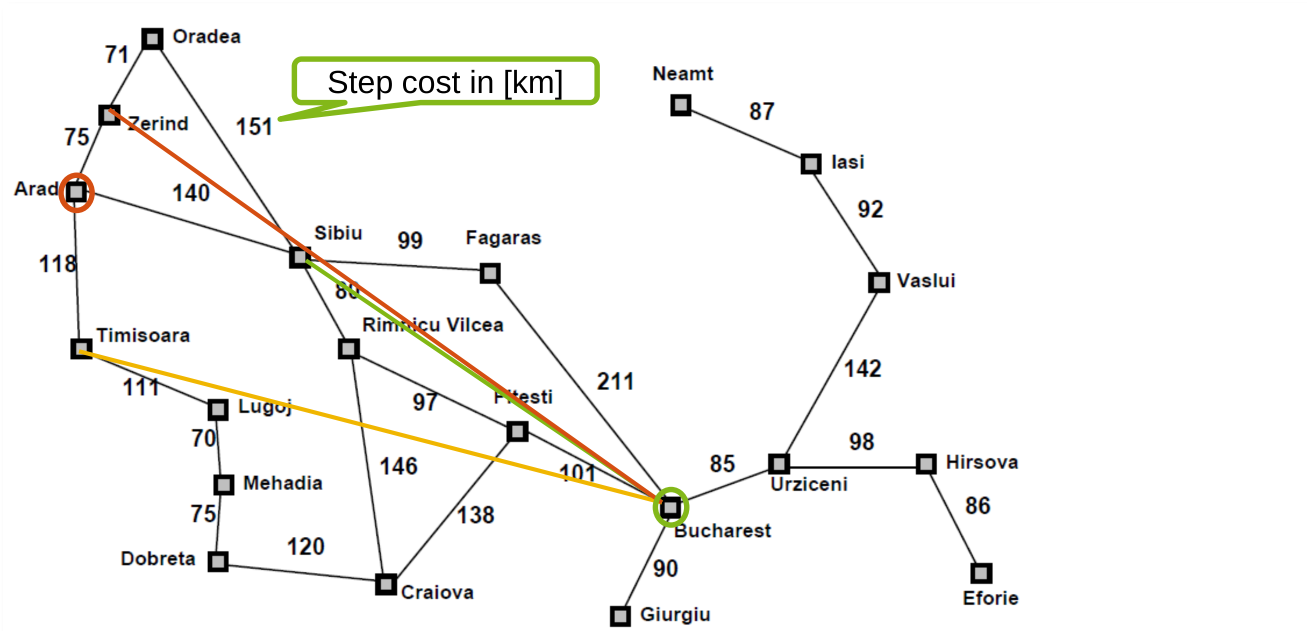
What could be a **good heuristic** for the distance to Bucharest (being in Arad)**?**



Tree-/graph search using additional knowledge
…beyond the definition of the problem

Best-first search
- **Select** the node to be expanded next **based on** some **evaluation function** $f(node)$
- Typically, $\boldsymbol{f}$ **is** implemented by a **heuristic** $h(node)$ (measure of "desirability")
- $h(node)$ facilitates pruning of the search tree: options are eliminated without examination

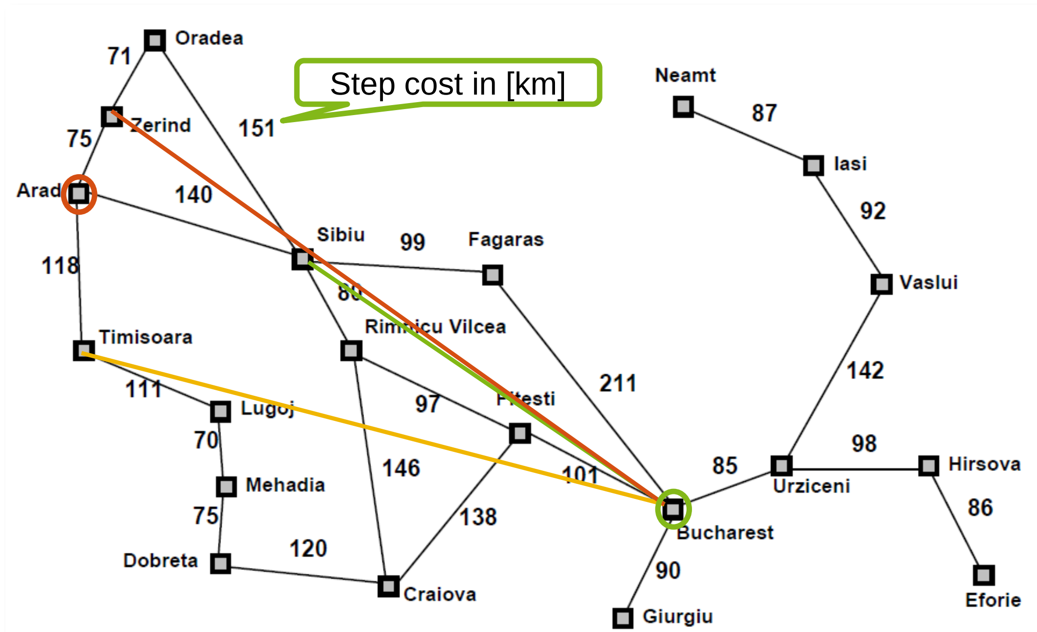
What could be a **good heuristic** for the distance to Bucharest (being in Arad)**?**



Tree-/graph search using additional knowledge
…beyond the definition of the problem

Best-first search
- **Select** the node to be expanded next **based on** some **evaluation function** $f(node)$
- Typically, $\boldsymbol{f}$ **is** implemented by a **heuristic** $h(node)$ (measure of "desirability")
- $h(node)$ facilitates pruning of the search tree: options are eliminated without examination

What could be a **good heuristic** for the distance to Bucharest (being in Arad)**?**



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

Typical implementations

Greedy search

- Expand node with lowest *subsequent* cost estimate according to some h, i.e. $f(n) = h(n)$
- n may only *appear* to be closest to the goal

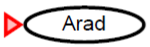
A*

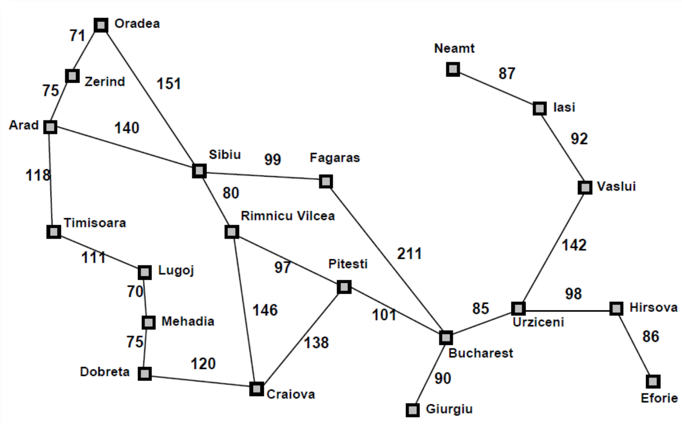
- Obvious improvement: **consider full path cost**, i.e. $f(n) = g(n) + h(n)$
  ($g(n)$ cost so far to reach n, $h(n)$ estimated cost to goal from n, $f(n)$ estimated total path cost)
- $h(n)$ needs to be admissible: $\leq true\ cost$ and ≥ 0 (e.g., $h_{straight\ line\ distance}$)
- A* search is optimal, complete
- A* has time complexity $O(2^{(error\ of\ h)\cdot d})$ and **keeps all nodes in memory**

SMA* - simplified **memory-bounded A***

- A* usually runs out of space first → SMA* overcomes this by
- …filling the memory up, then **starting to forget** the worst expanded nodes
- …ancestors of forgotten **subtrees remember** the value of the **best path** within them
- …thus, subtrees are only **regenerated if no better** solution exists

A* Example

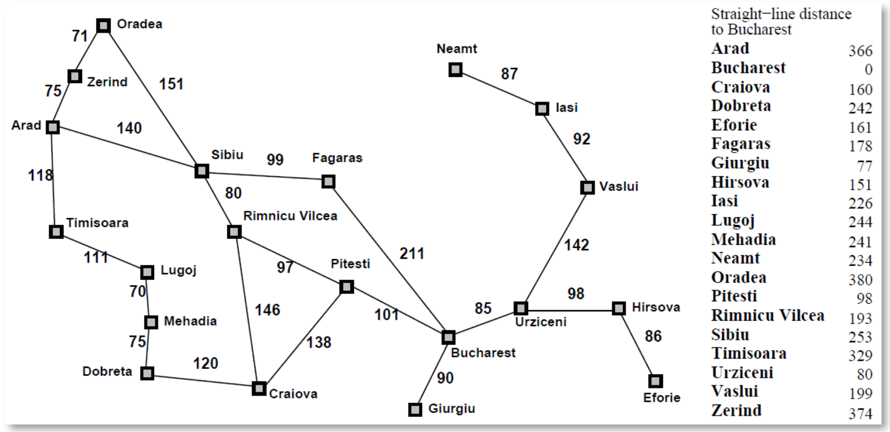


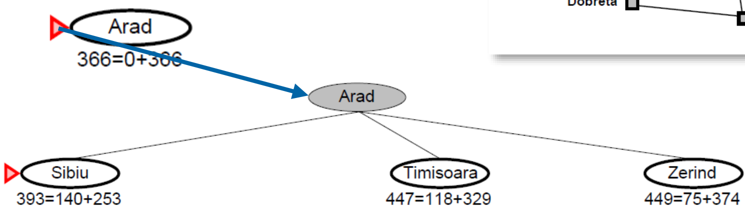


Straight−line distance to Bucharest

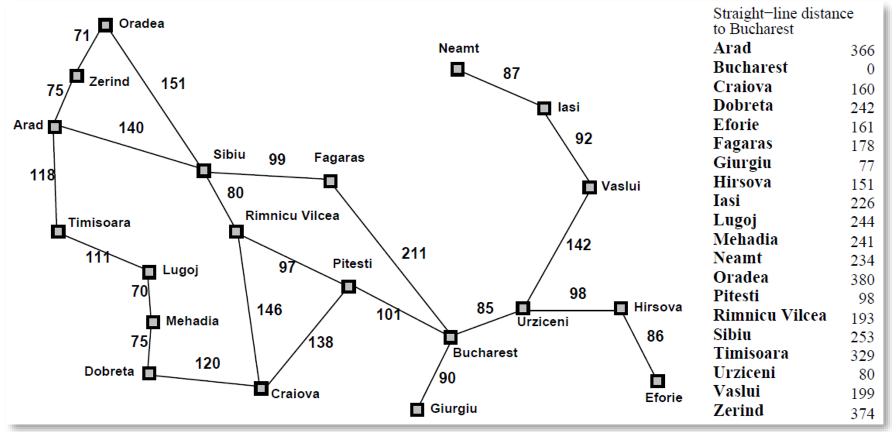
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

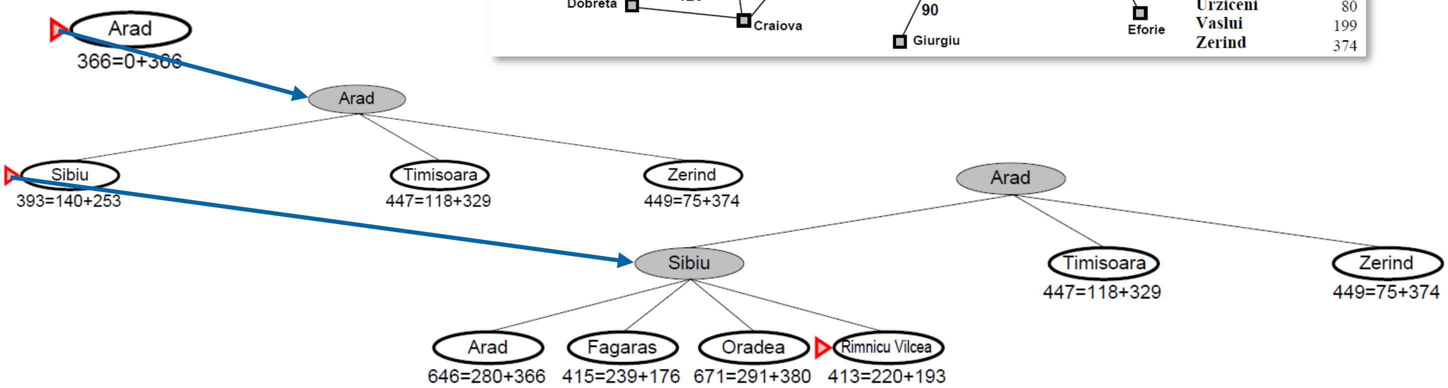
A* Example


Oradea
Zerind
Arad
Sibiu
Fagaras
Timisoara
Rimnicu Vilcea
Lugoj
Pitesti
Mehadia
Dobreta
Craiova
Bucharest
Giurgiu
Neamt
Iasi
Vaslui
Urziceni
Hirsova
Eforie
71
75
151
140
118
99
80
111
97
211
70
146
101
75
138
120
85
90
87
92
142
98
86
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

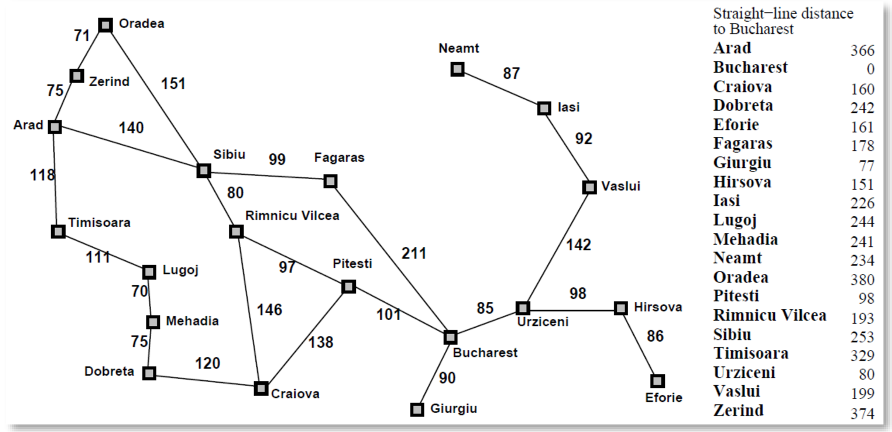

Arad
366=0+366
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

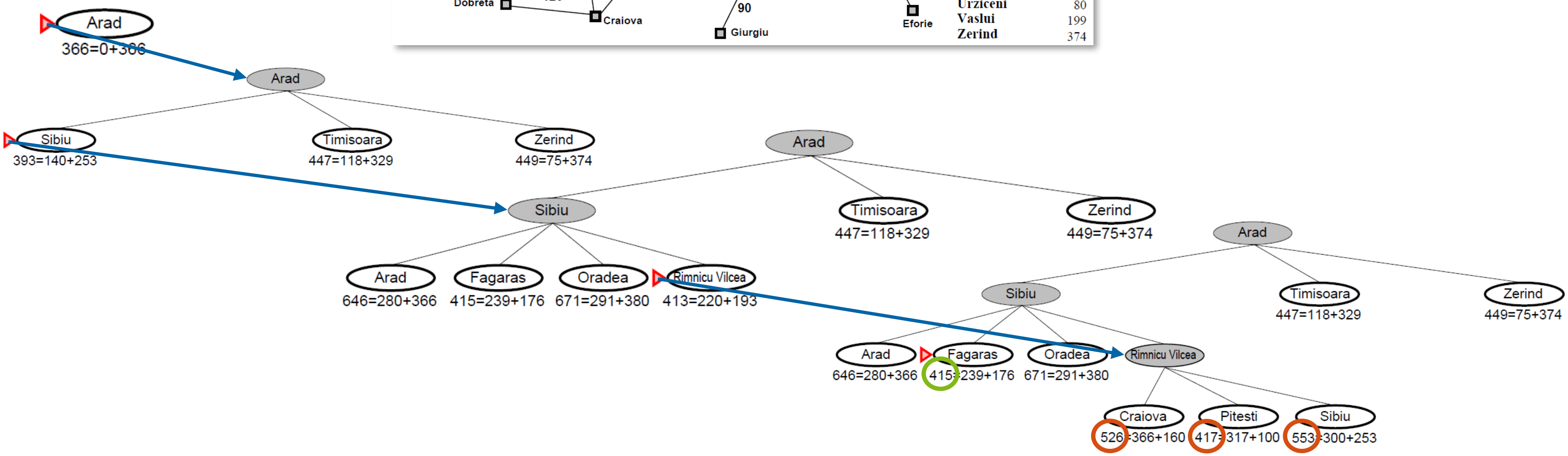
A* Example


Oradea
Zerind
Arad
Sibiu
Fagaras
Timisoara
Rimnicu Vilcea
Lugoj
Pitesti
Mehadia
Dobreta
Craiova
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
71
75
151
140
118
99
80
111
70
97
211
146
101
75
138
120
85
90
98
86
142
92
87
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

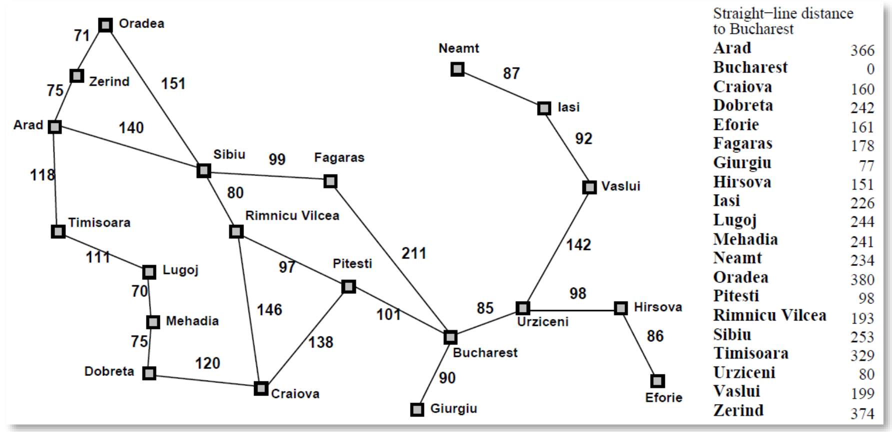

Arad
366=0+366
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

A* Example

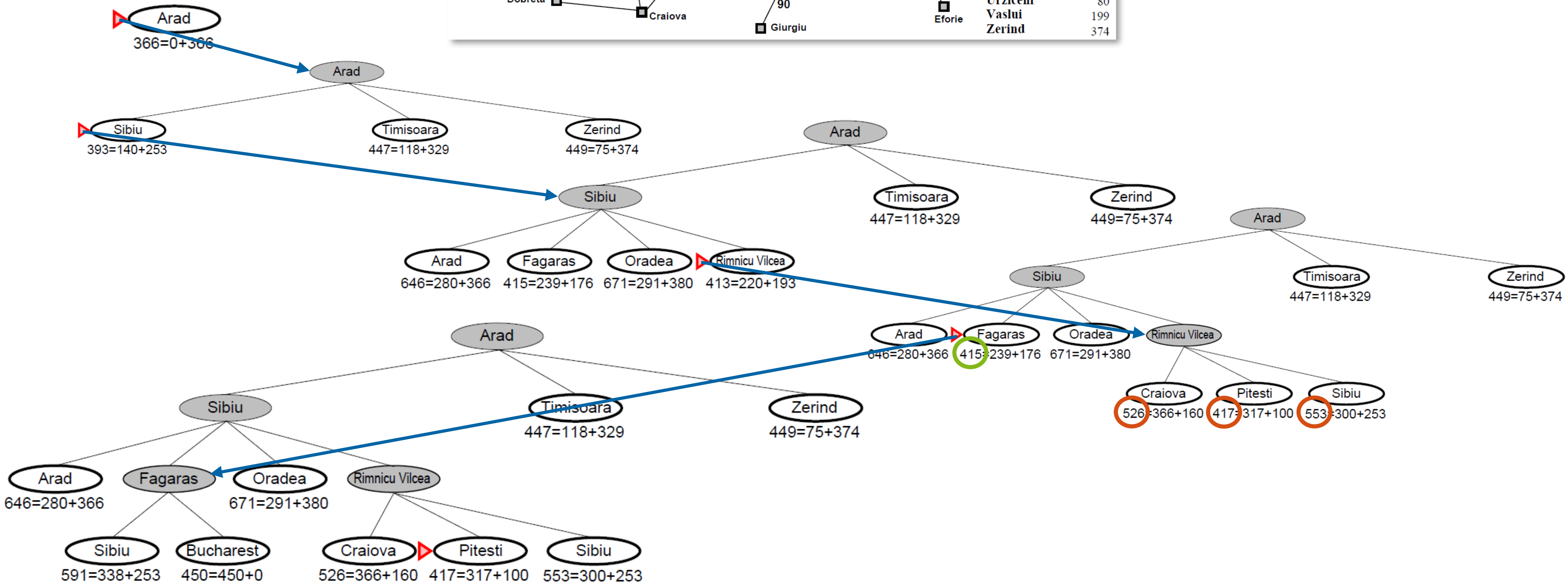

Oradea
Zerind
Arad
Sibiu
Fagaras
Timisoara
Rimnicu Vilcea
Lugoj
Pitesti
Mehadia
Dobreta
Craiova
Bucharest
Giurgiu
Neamt
Iasi
Vaslui
Urziceni
Hirsova
Eforie
71
75
151
140
118
99
80
211
111
97
70
146
101
75
138
120
85
90
87
92
142
98
86
Straight−line distance
to Bucharest
Arad 366
Bucharest 0
Craiova 160
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


Arad
366=0+366
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Example



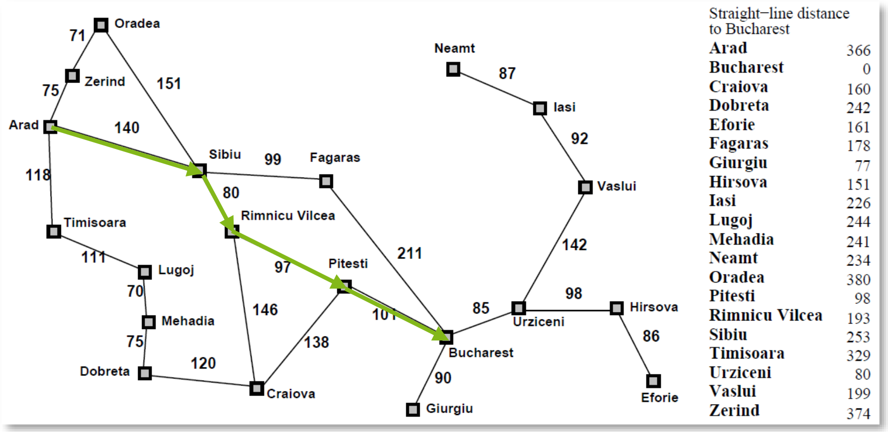
Oradea
Zerind
Arad
Sibiu
Fagaras
Timisoara
Rimnicu Vilcea
Lugoj
Pitesti
Mehadia
Dobreta
Craiova
Bucharest
Giurgiu
Neamt
Iasi
Vaslui
Urziceni
Hirsova
Eforie
71
75
151
140
118
99
80
111
97
211
70
146
101
75
138
120
85
90
87
92
142
98
86
Straight−line distance
to Bucharest
Arad 366
Bucharest 0
Craiova 160
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

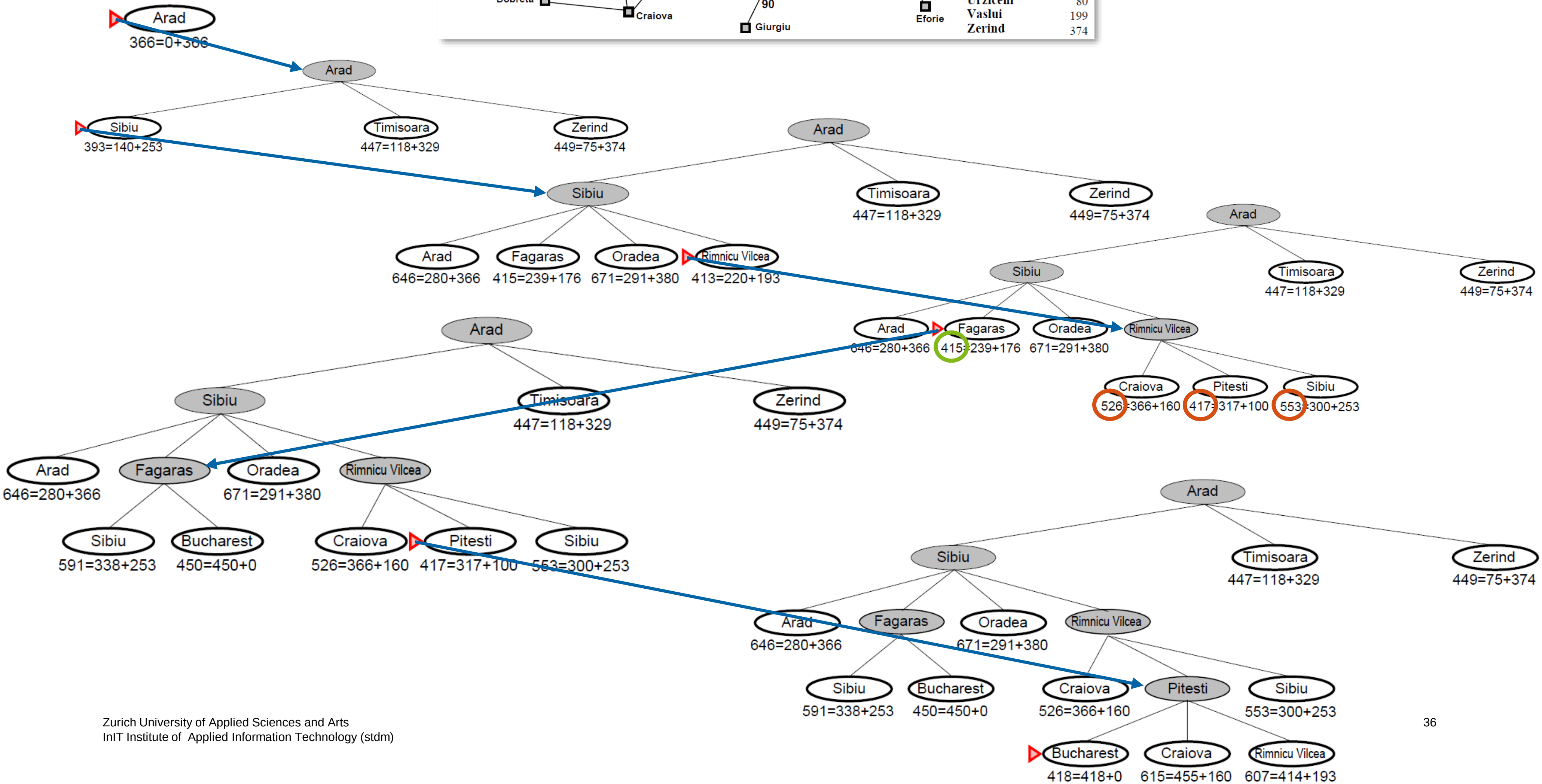


Arad
366=0+366
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Example

Zurich University
of Applied Sciences





Succeeding with search

Learning to search

- **Learn a heuristic** function: use inductive supervised learning on features of a state
- **Alternative: construct** a **metalevel state space**, consisting of all internal states of search program
  Example: For A* searching for a route in Romania, the search tree is its internal state
- Actions in metalevel space: computations that alter the metalevel state
  In the example: Expanding a node
- Solution in metalevel space: a path as depicted on the last slide
  ➔ can be input to machine learning algorithms to avoid unnecessary expansions

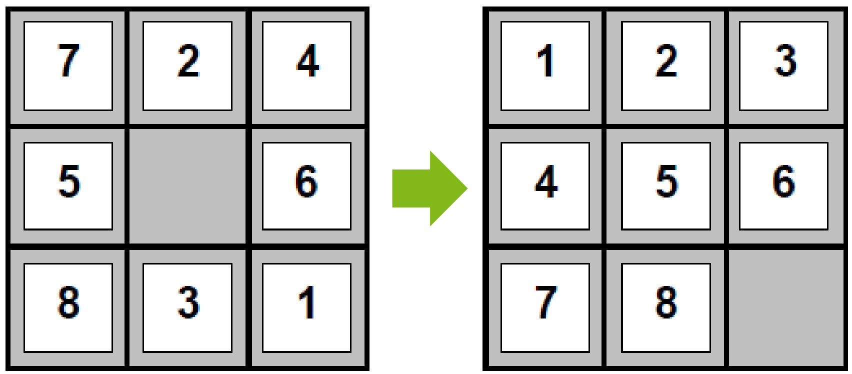
Practical advice

- **A* is impractical** for large scale problems
- Practical, **robust choice**: **SMA***
- **Have good heuristic functions!** A well-designed heuristic would have $b^* \approx 1$
  (b^* is the effective branching factor)

A closer look on heuristic functions
Example: 8-puzzle

Two proposals – which is better?

- $h_1(n) =$ **number of misplaced** tiles
- $h_2(n) =$ total **Manhattan distance** (i.e., no. of horizontal/vertical squares from desired location of each tile)



Start State S

Goal State

$$h_1(S) = 6$$
$$h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$$

Dominance
The 8-puzzle example continues

If $h_2(n) \geq h_1(n)\ \forall n$ ➔ h_2 dominates h_1 and **is better for search**

Typical search costs

Algorithm	#nodes expanded with $d = 14$	#nodes expanded with $d = 24$
Iterative deepening	3'473'941	~54'000'000'000
A* (h_1)	539	39'135
A* (h_2)	113	1'641

Simple improvement
- Given any admissible heuristics h_a, h_b:
- $h(n) = \max\big(h_a(n), h_b(n)\big)$ is also admissible and dominates h_a, h_b

Relaxed problems
Improving heuristics intelligently

Relaxation as a key

- Admissible **heuristics** can be **derived from** the **exact solution cost of a relaxed version** of the problem
- A relaxed problem has fewer constraints on the actions
- Relaxation can be automatized!
  E.g., «Absolver» by (Prieditis, 1993) found best heuristic for 8-puzzle, first heuristic for Rubik's cube

Examples of relaxed 8-puzzle rules

- If **each tile can move anywhere** (in 1 step), then $\boldsymbol{h_1(n)}$ **gives** the **shortest** solution
- If each tile can move **to any adjacent** square, then $\boldsymbol{h_2(n)}$ **gives** the **shortest** solution

Intuition

- Removing constraints adds edges to the state graph
- Additional edges might provide „**short cuts**"
- The optimal solution cost of a relaxed problem ("short cut") can be no greater than the optimal solution cost of the real problem

Zurich University
of Applied Sciences

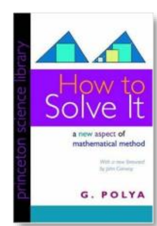
Where's the intelligence?
Man vs. machine

Uninformed search
- In the **abstraction** of the problem
- In the **choice of algorithm** that is optimal for the problem at hand
- In the **systematic exploration** of the state space graph

Heuristic search
- Additionally, in the **heuristic** function

Originally written in German during his research stay at ETH

→ see also: Polya, *«How to solve it - a new aspect of mathematical method»*, 1945



Exercise: Missionaries & cannibals (AIMA ex. 3.9)

Three missionaries and **3 cannibals** are on one side of a river, along with **a boat** that **can hold one or two** people. Find a way to **get** everyone **to the other side, without ever leaving** a group of **missionaries** in one place **outnumbered by the cannibals** in that place.

- Formulate the problem precisely:
  Make only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.
- Implement and solve the problem optimally:
  Use an appropriate search algorithm. Is it a good idea to check for repeated states?
- Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

Review

- **Search** as an approach to AI **exists in its current form** more or less **since AI's inception**
- **Extensions** of search algorithms exist **to non-deterministic** and **partially observable** environments as well as **online** search

- **Problem formulation** usually **requires abstracting** away real-world details to define a state space that can feasibly be explored
- **Iterative deepening** search **uses only linear space** and not much more time than other uninformed algorithms
- **Graph search** can be **exponentially more efficient** than tree search

- **Good heuristics** can **dramatically reduce** search **cost**
- **A*** search **expands lowest** $g + h$
  ➔ complete and optimal, also optimally efficient (up to tie-breaks, for forward search)

- **Admissible heuristics** can be **derived from** exact solution of **relaxed problems**

APPENDIX

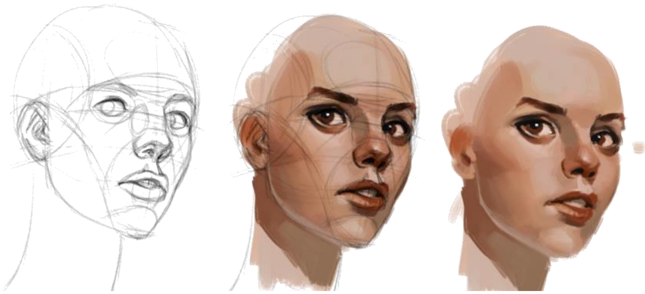
Fun fact: implement depth-first search in a maze by keeping your left hand on the wall.

On modeling and abstraction

Quoted from *AIMA*, p. 68-69, sec. 3.1.2

- *A **model*** [is] an abstract mathematical description […] and not the real thing
- The process of removing detail from a representation is called **abstraction**
- The abstraction is ***valid*** if we can **expand** any abstract solution into a solution in the more detailed world
- The abstraction is ***useful*** if carrying out each of the actions in the abstraction is **easier** than the original problem
- The choice of a **good abstraction** thus involves **removing as much detail as possible while retaining validity** and **ensuring that the abstract actions are easy** to carry out

➔ Were it not for the ability to construct useful abstractions, intelligent agents would be completely swamped by the real world



Recap on complexity theory

Problems are classified to be part of (attention: only intuitive “definitions”)

- **P** – can be solved in polynomial time by a deterministic algorithm
  → deemed to be solvable «efficiently»
- **NP** – can only be solved $\underbrace{\text{efficiently (i.e., in polynomial time)}}$ by guessing the solution
  (i.e., by a non-deterministic algorithm)

When people talk about **efficient computation**, this **always means** (at most) **polynomial time**: $efficient \sim polynomial\ time$.

More terminology

- **NP-hard** – a problem x is said to be NP-hard if **all problems in NP can be reduced** to (i.e., converted into / stated as) x (i.e., can be solved by an algorithm for x) efficiently
  → Example: Traveling salesman problem (i.e., any problem in NP is at most as hard as x)
- **NP-complete** – a problem x is said to be NP-complete if it is NP-hard and in NP
  → Example: The satisfiability problem (SAT) – is there an assignment of truth values to make a given formula of propositional logic true? (→ see V06 and AIMA ch. 7.5)

…which is all good (i.e., we don't have to care for efficiency) if $P = NP$ (tremendously unlikely!)

Further reading

- AIMA appendix A.1 (< 3 pages!)
- J. Koehler's lecture slides on complexity and AI: https://user.enterpriselab.ch/~takoehle/teaching/ai/ProblemComplexity.pdf
- Some more intuition: http://stackoverflow.com/questions/1857244/what-are-the-differences-between-np-np-complete-and-np-hard

Pseudocode for general tree- and graph search

```
function Tree-Search(problem, frontier) returns a solution, or failure
    frontier ← Insert(Make-Node(Initial-State(problem)), frontier)
    loop do
        if frontier is empty then return failure
        node ← Remove-Front(frontier) #choice of picked node defined by strategy
        if Goal-Test(problem) applied to State(node) succeeds return node
        frontier ← InsertAll(Expand(node, problem), frontier)
```

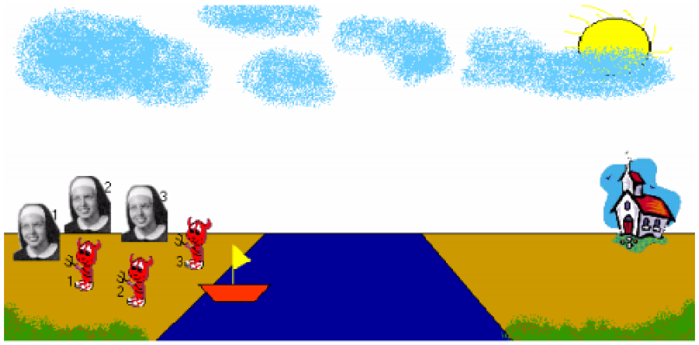
```
function Graph-Search(problem, frontier) returns a solution, or failure
    frontier ← Insert(Make-Node(Initial-State(problem)), frontier)
    explored ← empty
    loop do
        if frontier is empty then return failure
        node ←Remove-Front(frontier) #choice of picked node defined by strategy
        explored ← Insert(node, explored)
        if Goal-Test(problem) applied to State(node) succeeds return node
        frontier ← InsertAll(Expand(node, problem), frontier) only if not in frontier or explored set
```

➔ ***Bold italic*** font shows the additions that handle repeated states in graph search

Missionaries & cannibals (contd.)

States
- $\theta = (M, C, B)$ signifies the number of missionaries, cannibals, and boats on the left bank
- The start state is (3,3,1) and the goal state is (0,0,0)

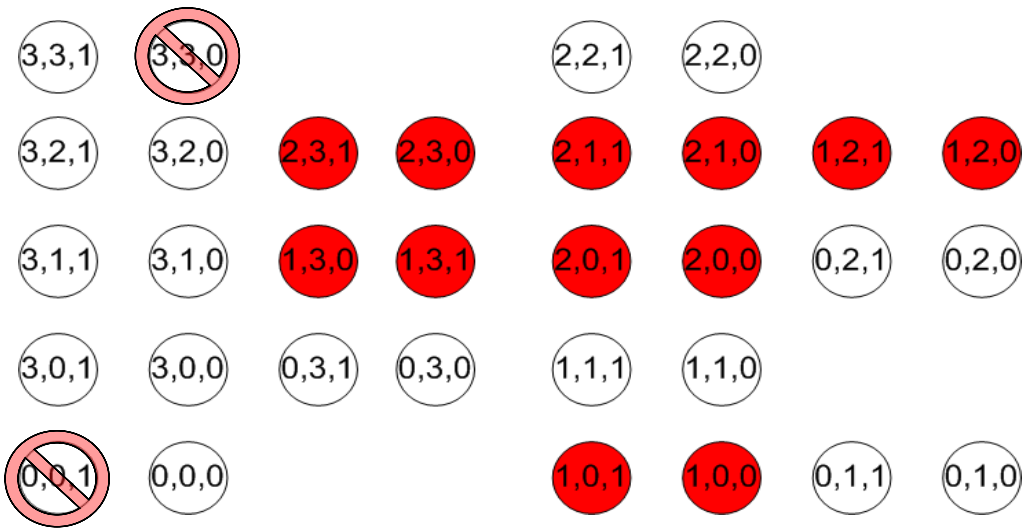
Actions (successor function)
- 10 possible, but only 5 available each move due to boat
- One cannibal/missionary crossing L→R: subtract (0,1,1) or (1,0,1)
- Two cannibals/missionaries crossing L→R: subtract (0,2,1) or (2,0,1)
- One cannibal/missionary crossing R→L: add (1,0,1) or (0,1,1)
- Two cannibals/missionaries crossing R→L: add (2,0,1) or (0,2,1)
- One cannibal and one missionary crossing: add/subtract (1,1,1)



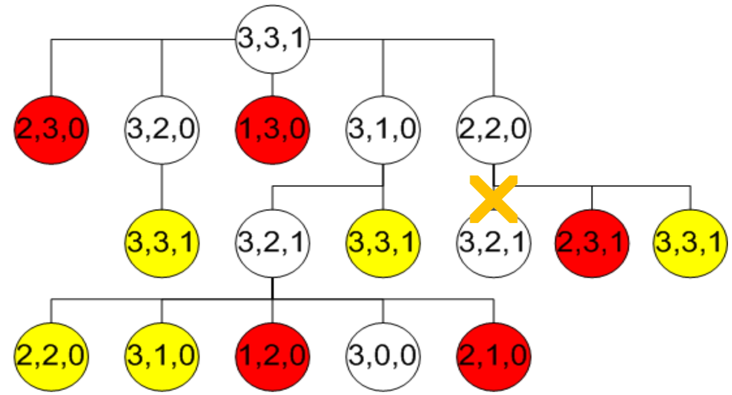
Source: http://www.cse.msu.edu/~michmer3/440/Lab1/cannibal.html

Missionaries & cannibals states

- Assumes that passengers have to get out of the boat after the trip
- Red states = missionaries get eaten



Breadth-first search (4 iterations) on missionaries & cannibals



States are generated by applying

- +/- (1,0,1)
- +/- (0,1,1)
- +/- (2,0,1)
- +/- (0,2,1)
- +/- (1,1,1)

Red states = missionaries get eaten
Yellow states = repeated states

Breadth-first search (final state) on missionaries & cannibals

- Breadth first search expanded 48 nodes
- This is an optimal solution (minimum number of crossings)

- Depth-first search expanded 30 nodes
- ...if repeated states are checked, otherwise we end up in an endless loop