

# Informed State Space Search

## B4B36ZUI, LS 2018

**Branislav Božanský, Martin Schaefer, David Fiedler, Jaromír Janisch**  
{name.surname}@agents.fel.cvut.cz  
Artificial Intelligence Center, Czech Technical University

# State Space

## Formulation



- **Problem**
- **Initial state** –  $s_0$
- **Successor function** –  $x \in S \rightarrow succ(x) \subseteq 2^S$
- **Goal test** –  $x \in S \rightarrow goal(x) = T \mid F$
- **Arc cost** –  $c(x, succ(x))$
  
- **Solution** is set of actions leading from initial state to a goal state

# Tree Search Algorithm

## Formulation



**function** TREE-SEARCH(*problem*, *fringe*) **returns** a solution, or failure

*fringe* ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

**loop do**

**if** *fringe* is empty **then return** failure

*node* ← REMOVE-FRONT(*fringe*)

**if** GOAL-TEST(*problem*, STATE(*node*)) **then return** *node*

*fringe* ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

---

**function** EXPAND(*node*, *problem*) **returns** a set of nodes

*successors* ← the empty set

**for each** *action*, *result* **in** SUCCESSOR-FN(*problem*, STATE[*node*]) **do**

*s* ← a new NODE

    PARENT-NODE[*s*] ← *node*; ACTION[*s*] ← *action*; STATE[*s*] ← *result*

    PATH-COST[*s*] ← PATH-COST[*node*] + STEP-COST(*node*, *action*, *s*)

    DEPTH[*s*] ← DEPTH[*node*] + 1

    add *s* to *successors*

**return** *successors*

# Tree Search Algorithm

## Formulation



```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE(node)) then return node
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

BFS

Insert at the end

DFS

Insert at the beginning

*s* ← a new NODE

PARENT-NODE[*s*] ← *node*; ACTION[*s*] ← *action*; STATE[*s*] ← *result*

PATH-COST[*s*] ← PATH-COST[*node*] + STEP-COST(*node*, *action*, *s*)

DEPTH[*s*] ← DEPTH[*node*] + 1

add *s* to *successors*

**return** *successors*

# Graph Search

Using a closed list

**function** GRAPH-SEARCH(*problem*, *fringe*) **returns** a solution, or failure

*closed* ← an empty set

*fringe* ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

**loop do**

**if** *fringe* is empty **then return** failure

*node* ← REMOVE-FRONT(*fringe*)

**if** GOAL-TEST(*problem*, STATE[*node*]) **then return** *node*

**if** STATE[*node*] is not in *closed* **then**

    add STATE[*node*] to *closed*

*fringe* ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

**end**

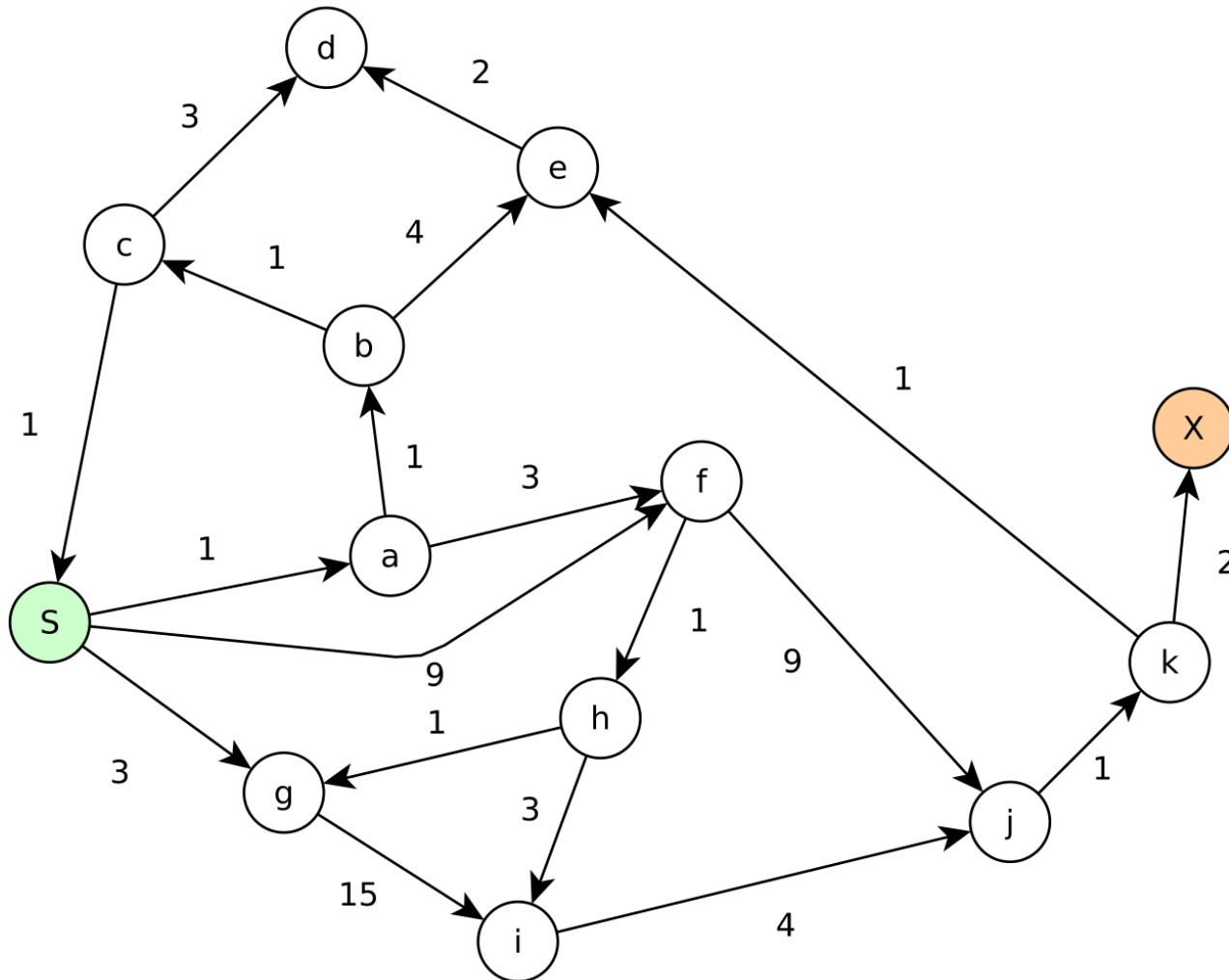
# Searching the State Space

## Algorithms



- Breadth first search **BFS**
- Depth first search **DFS**
- Depth limited search (DFS with search limit  $l$ )
- Iterative deepening search (Iteratively increase  $l$ )
- Uniform cost search

# Uniform-Cost Search Exercise



# Informed Search Problems



- **Problem**
- **Initial state** –  $s_0$
- **Successor function** –  $x \in S \rightarrow succ(x) \in 2^S$
- **Goal test** –  $x \in S \rightarrow goal(x) = T \mid F$
- **Arc cost** –  $c(x, succ(x))$

**Heuristic**  $s \in S: h(s) \rightarrow R$

- $g(s)$ : cost to reach the state  $s$
- $h(s)$ : estimated cost to get from state  $s$  to goal state



# Heuristic



Wikipedia: “A **heuristic function**, or simply a **heuristic**, is a [function](#) that ranks alternatives in various [search algorithms](#) at each branching step based on the available information ([heuristically](#)) in order to make a decision about which branch to follow during a search.”

# Heuristic

- Evaluation function for each node  $h(N)$
- Value is independent of the current search tree
- Expressing desirability
  - Estimates the cost from  $N$  to a goal state  $G$
  - $h(N) \geq 0$

Goal of heuristic design:  
As close to the real cost as  
possible!

# Best-first Search Algorithms

- Evaluation function  $f(n)$  for each state/node

$$f(n) = g(n) + h(n)$$

COST

HEURISTIC

- → Selecting **best** node first – “best-first search”

Uniform cost search:  $h(N) = 0$

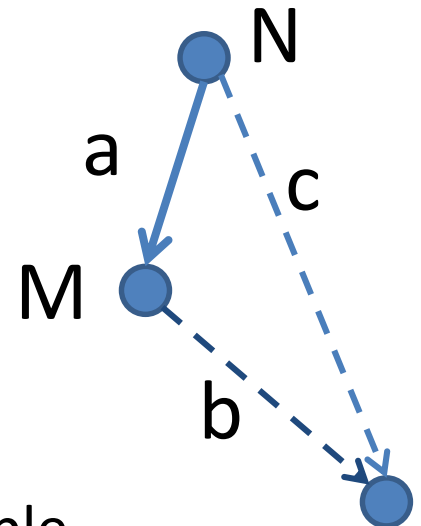
Greedy search:  $g(N) = 0$ ,  $h(N)$  arbitrary

A search:  $g(N)$ ,  $h(N)$  arbitrary

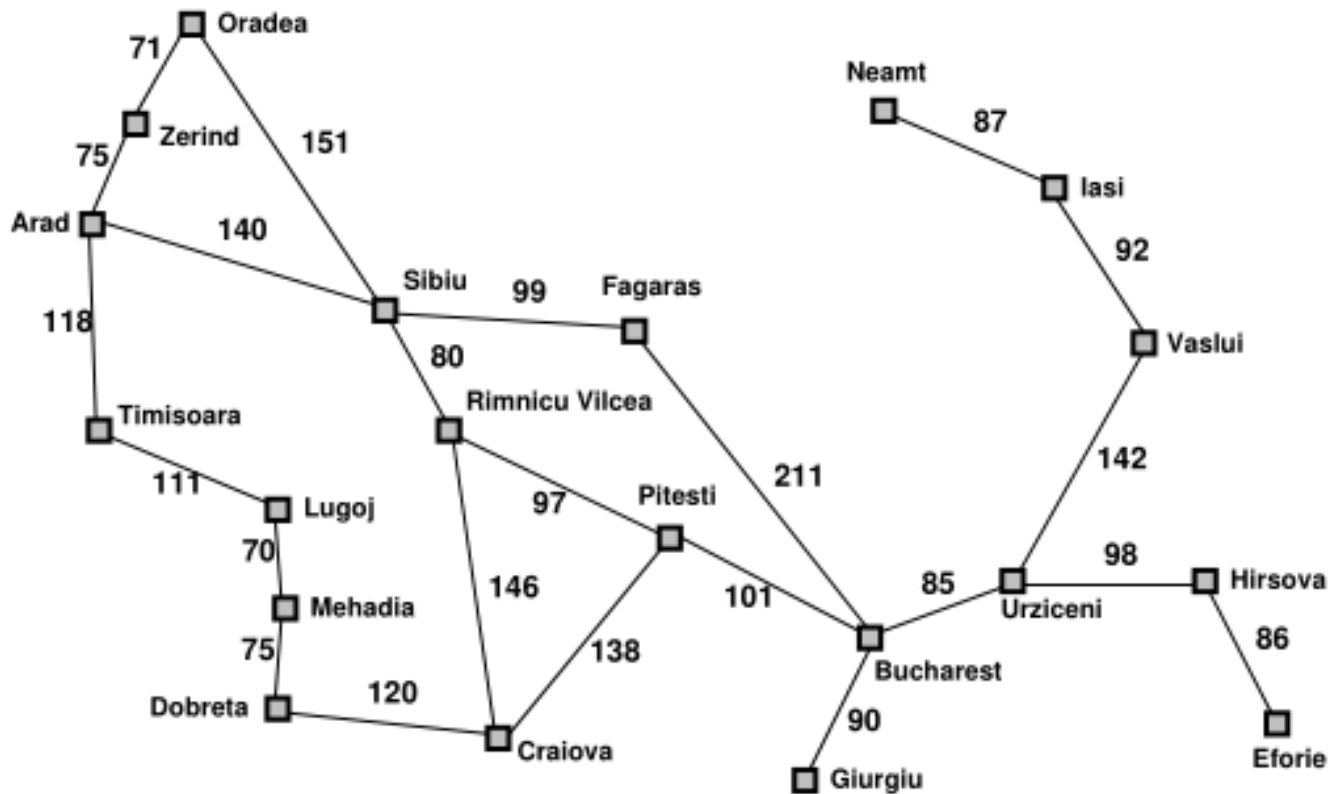
A\* search:  $g(N)$ ,  $h(N)$  admissible

# H(N) – Heuristic function

- We know the cost to the node  $g(n)$  – nothing to tune here
- We don't know the exact cost from  $n$  to goal  $h(n)$  – if we knew, no need to search – **estimate it!**
- H(N) – **admissible** and **consistent** heuristic
- Admissible = *optimistic* – it never overestimates the cost to the goal
  - $0 \leq h(N) \leq h^*(N)$
- Consistent = Triangle inequality is valid
  - $a + b \geq c$
  - $g(N, M) + h(M) \geq h(N)$
  - $\rightarrow$  once a node is expanded, the cost by which it was reached is the lowest possible



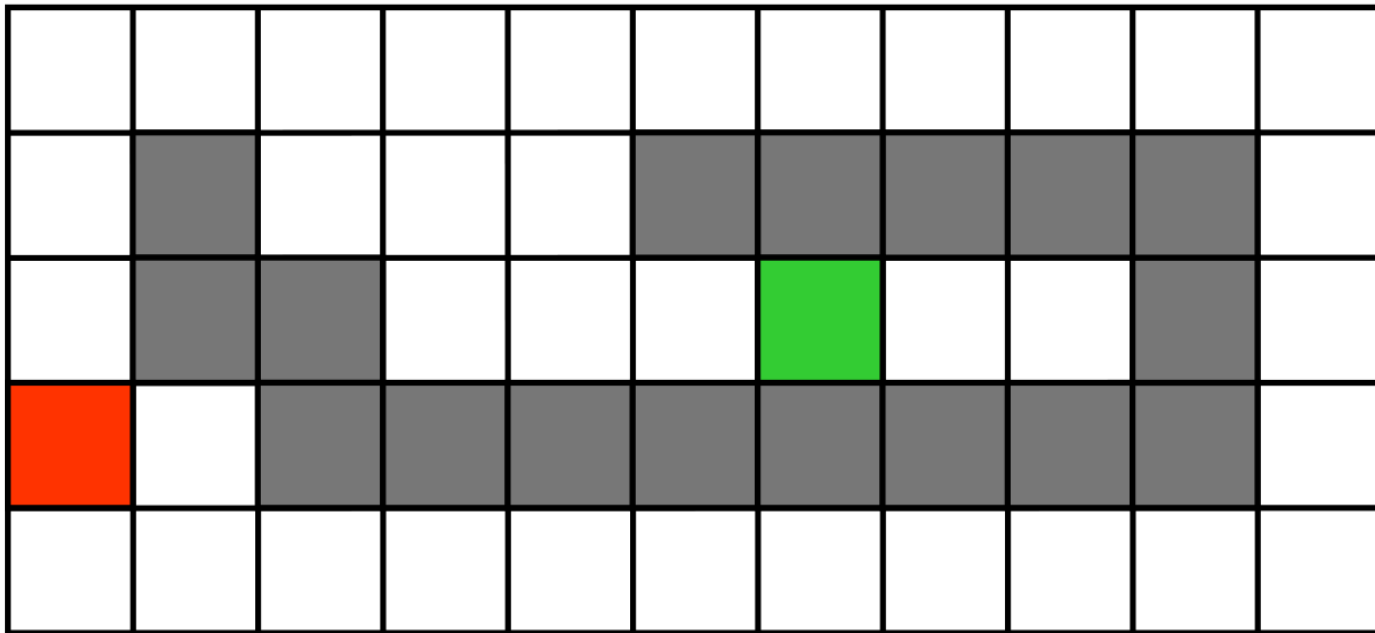
# Informed Search Exercises



Straight-line distance  
to Bucharest

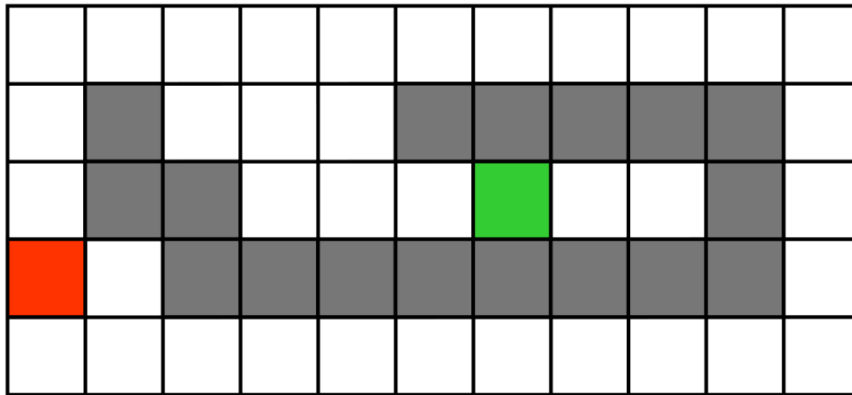
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	178
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	98
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara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

# Path in a maze



What are possible heuristics?

# Path in a maze



- What if we can jump over 1 obstacle?
- What if we can do several jumps in one move?
- What if there are many units that need to reach destination?
- What if we move with a knight (chess piece)?

# Roomba Robot path planning





# The ferryman problem



# Escaping the World Trade Center



- Imagine a huge skyscraper with several elevators. As the input you have:
- set of elevators, where for each you have:
  - - range of the floors that this elevator is operating in
  - - how many floors does this elevator skip (e.g. an elevator can stop only on every second floor, or every fifth floor, etc.)
  - - speed (time in seconds to go up/down one floor)
  - - starting position (number of the floor)



# Escaping the World Trade Center



- Let us assume, that transfer from one elevator to another one takes the same time (given as input -  $t$ ).
  - You are starting in  $k$ th floor and you want to find the quickest way to the ground floor.
  - You can assume that you are alone in the building and elevators do not run by themselves.
- 
1. What are the states?
  2. What is the initial state and the goal state?
  3. What is the cost function?

# Stock Exchange Problem

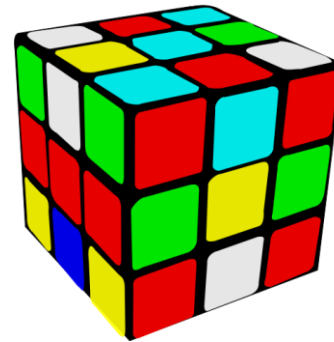


- As the input data you have a set of requests that contains a set of 4-tuples:
- (STOCK\_BUY/STOCK\_SELL, STOCK\_ID, STOCK\_AMOUNT, STOCK\_PRICE)
- that describe a request to either sell or buy given amount of given stock for given price. The price is interpreted as minimal in case the request is to sell stocks and maximal, in case the request is to buy.
- Your task is to find appropriate price for each STOCK\_ID that would maximize the sum of amount of the traded stocks.

# State Space

More examples

- “Perfect” Spam filter
- Spellcheck suggestion design
- Solving a puzzle
- Rubik’s cube
- Monkey & Bananas
- Crossword puzzles
- Knapsack problem
- Traveling Salesman p
- Baking a chicken
- App. Moving with friends



A	I	K	E	N		S	L	A	M	S		F	B	I	
P	H	I	L	O		H	A	B	I	T		L	L	C	
P	O	L	A	R		O	C	O	M	E		Y	O	O	
	P	O	P	T	H	E	Q	U	E	S	T	I	O	N	
			S	H	U		U	N	O		E	N	D	S	
L	A	C	E		P	L	E	D		P	N	G			
I	R	A		S	T	I	R		A	I	E	L	L	O	
M	A	R	L	O	W	E		E	P	S	T	E	I	N	
E	L	P	A	S	O		C	O	P	A		A	N	Y	
			O	D	O		B	O	N	E		S	P	E	X
B	B	O	Y		C	A	M		A	B	A				
W	I	L	L		Y	O	U		M	A	R	R	Y	M	E
A	S	I			A	L	B	U	M		O	B	E	A	H
N	O	N			N	O	L	T	E		N	O	I	S	E
A	N	G			G	R	E	E	N		C	O	R	E	Y

