Intelligent Systems

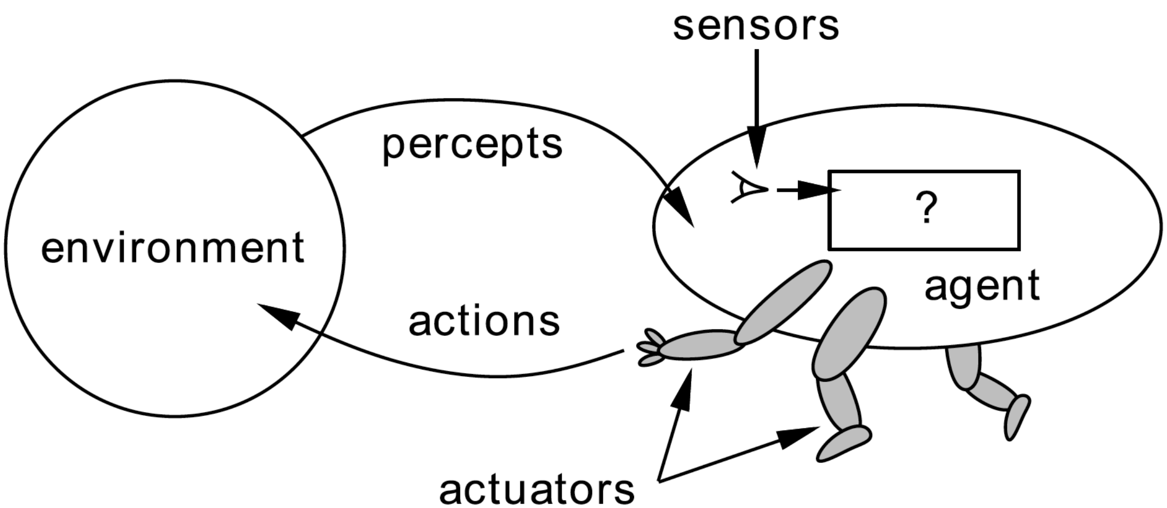
Ինտելեկտուալ տեղեկ. համակարգեր

NUACA/ՃՇՀԱՀ

2017

LECTURE 2

AI = acting rationally
ԱԲ = գործել ռացիոնալ



"For each possible *percept sequence*, a rational agent should select an *action* that is expected to maximize its *performance measure*, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has."

Կամայական մուտքային ազդանշան ստանալուց հետո, ռացիոնալ գործակալն ընտրում է այն գործողությունը որը մեծացնում է իր կատարման չափանիշը, որը տրվում է ազդանշանների կամ նախնական գիտելիքների միջոցով:

Problems

A **problem** can be formulated by specifying:

–The **state space** *("the set of all possible states we might get into") - վիճակների բազմություն*

–The **initial state** *("where we are") - սկզբնական վիճակ*

–A test for the **goal** *("are we where we want to be?") - արդյոք հասել ենք նպատակին*

- *Formally, we define a Boolean function* Goal *on the set of states*

–Available **actions** at any state *("what can we do?") - հասանելի գործողությունները բազմություն (ամեն վիճակում)*

- *Formally, we define a set-valued function* Actions *on the set of states*

–A **transition model** *("what will that achieve?") - անցում նոր վիճակի*

- *Formally, we define a state-valued function* Result *on the set of (state,action) pairs*

–A **cost measure** for sequences of actions *("is it worth it?") - գործողության արժեք*

- *Formally, we define a real-valued function* Cost *on the set of actions, or sequences*

Problems

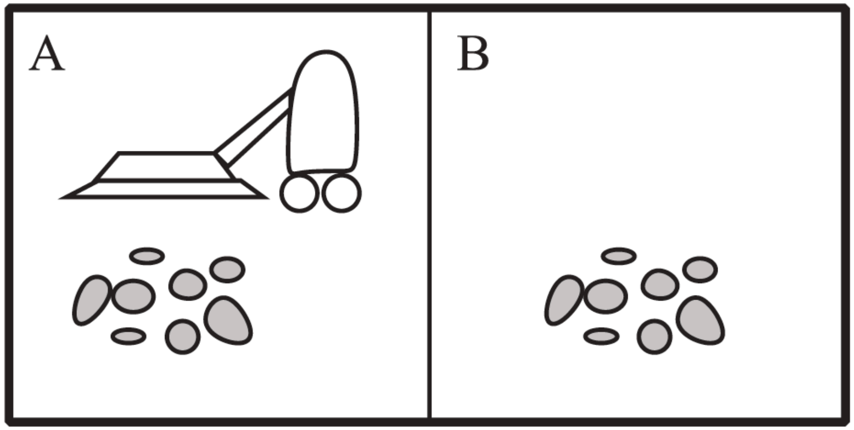
From a problem specification we can derive:

–A **solution** *("a sequence of actions that take us to a goal state") - լուծում*

- *Formally, this is a **path** (ճանապարհ) in the state space, a sequence $s_0,\ a_1, s_1,\ a_2, s_2,\ \ldots, a_n, s_n$ where:*
- $s_0, s_1, s_2, \ldots, s_n$ *are **states**,* s_0 *is the **initial state**,* s_n *is a **goal state*** ($\text{Goal}(s_n)$ = TRUE)
- $a_1,\ a_2, \ldots,\ a_n$ *are* **actions**,
- $a_i \in \text{Actions}(s_{i-1})$ and $s_i = \text{Result}(s_{i-1};\ a_i)$ for each 1 ≤ i ≤ n

– An **optimal solution** *("a solution with minimal cost") – օպտիմալ լուծում*

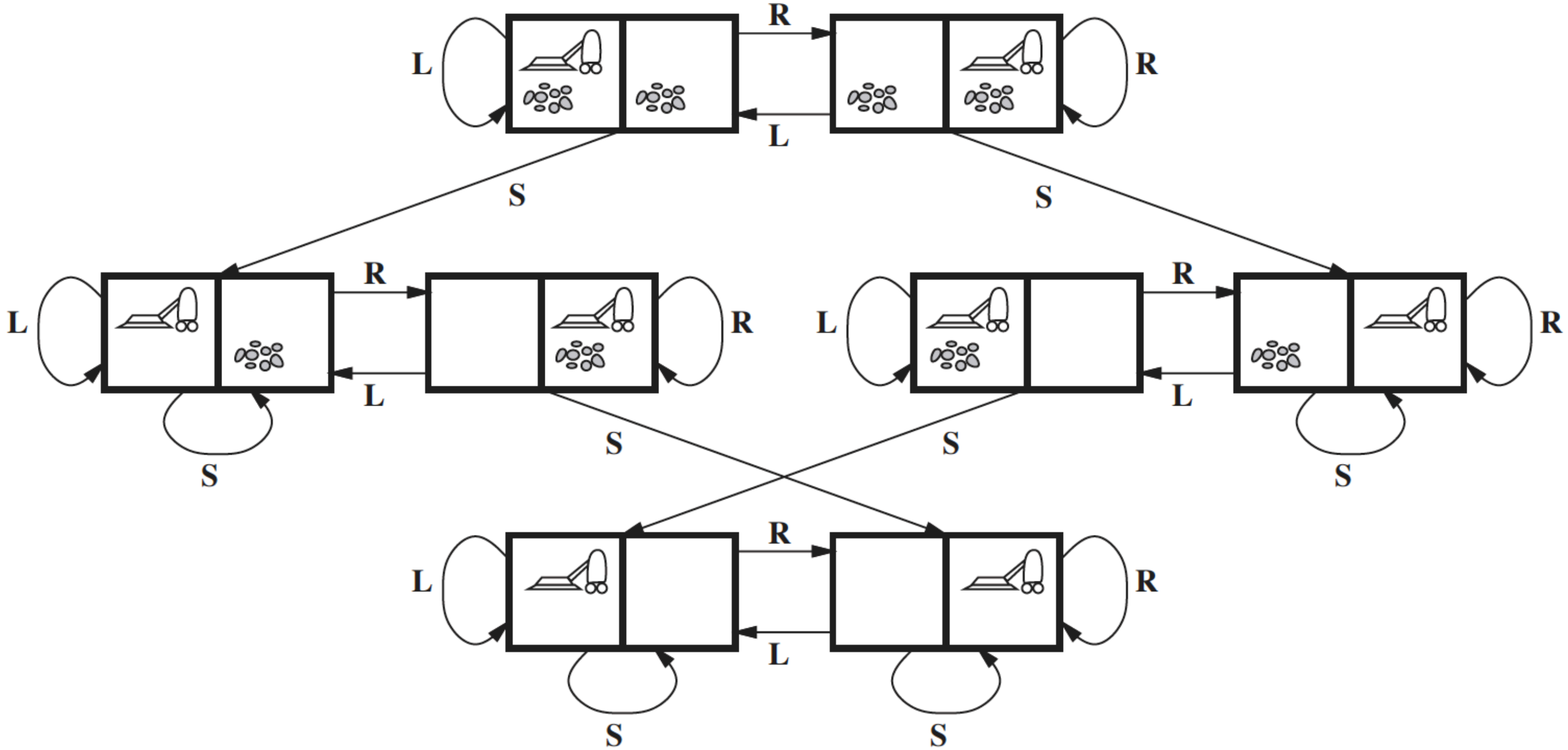
Օրինակ
Vacuum World



– The **initial state** *("Location A, Dirty A, Dirty B")*
– A test for the **goal** *("Is it clean?")*
– Available **actions** at any state *("Left, Right, or Clean")*
– A **transition model** *("Move left or right if possible, clean up dirt")*
– A **cost measure** for actions *("? power consumption")*

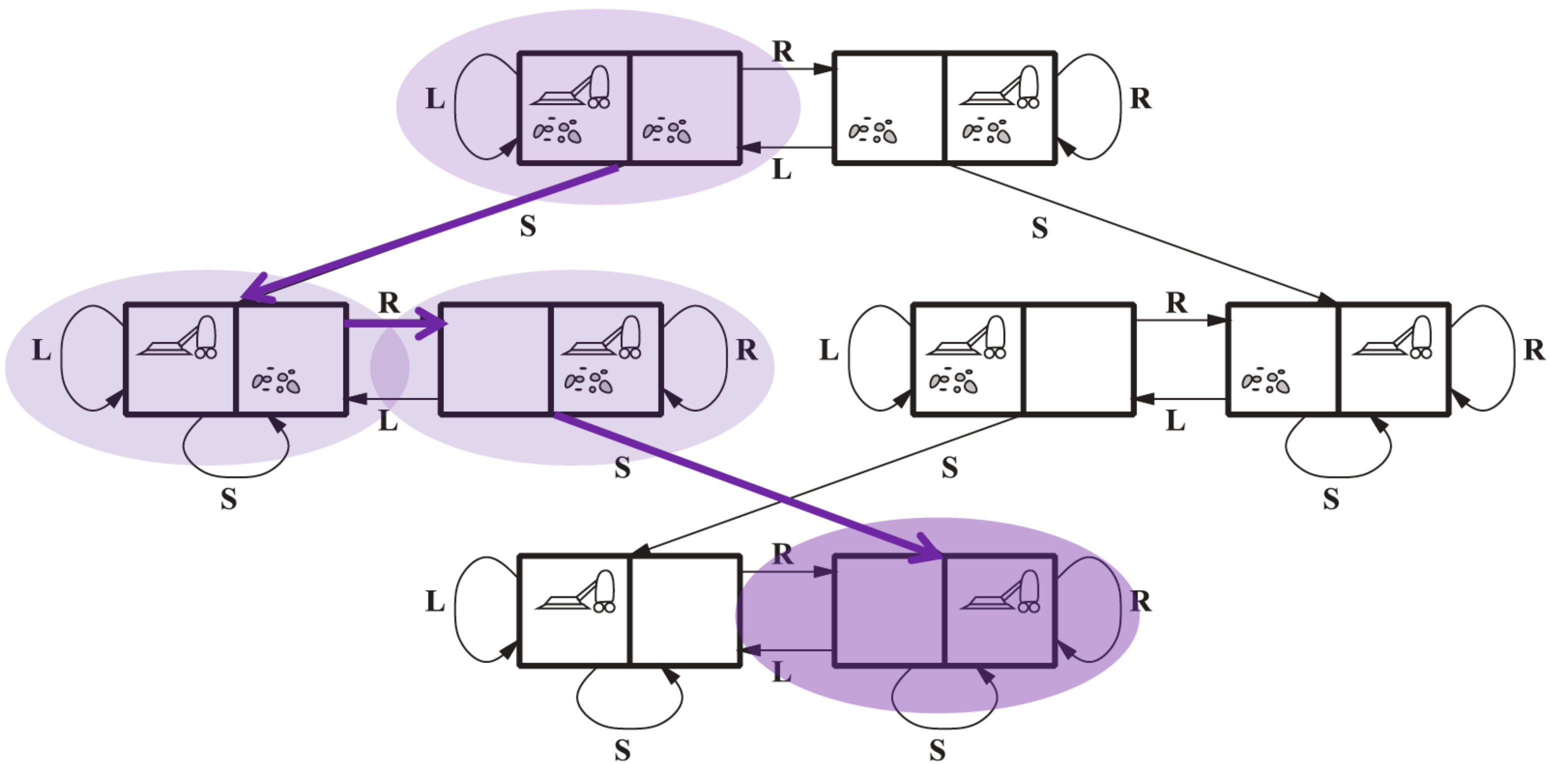
Օրինակ
Vacuum World

The **state space** ("the set of all possible states we might get into")



Օրինակ
Vacuum World

The **state space** ("the set of all possible states we might get into")



A **solution** *("a sequence of actions that take us to a goal state")*

Օրինակ
8-puzzle

7	2	4
5		6
8	3	1

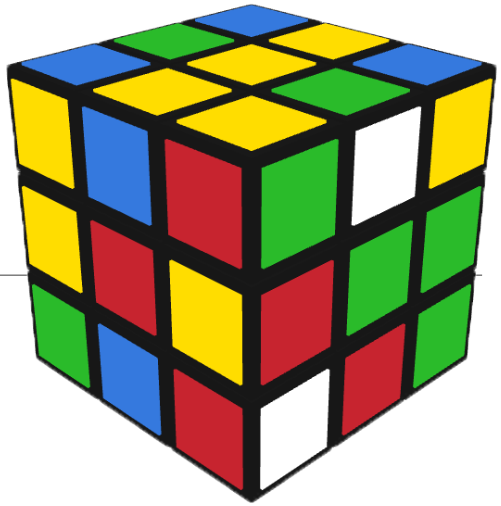
Start State

	1	2
3	4	5
6	7	8

Goal State

- – **States** *("Locations of eight tiles and blank space")*
- – The **initial state** *("Any state")*
- – A test for the **goal** *("Does the state match goal configuration?")*
- – Available **actions** at any state *("Move the blank space right/left/up/down")*
- – A **transition model** *("Move if possible")*
- – A **cost measure** for actions *("Each step costs 1")*

Օրինակ
Rubik's Cube



–The **initial state** *("Scrambled")*

–A test for the **goal** *("Is it unscrambled?")*

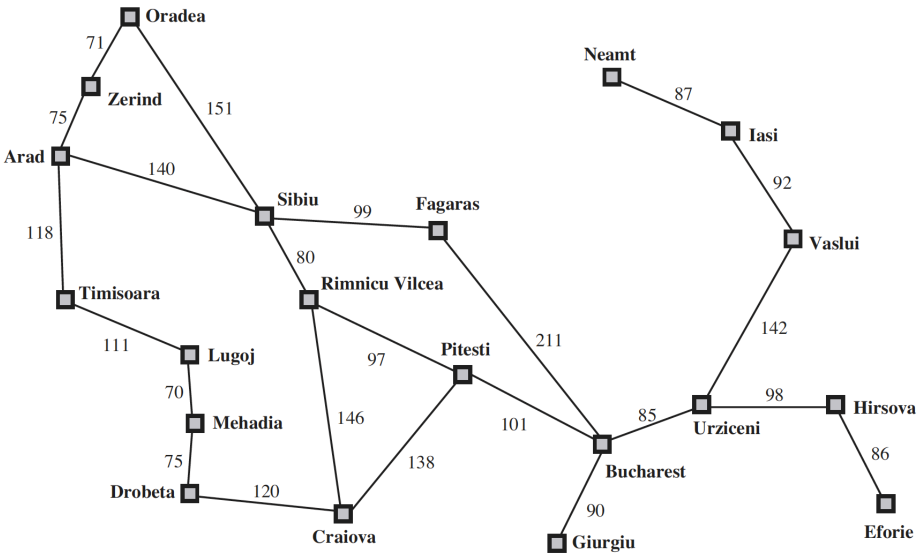
–Available **actions** at any state *("(6 x 3) different twists")*

–A **transition model** *("this twist changes the state like this")*

–A **cost measure** for actions *("quarter turns? or face turns?")*

901,083,404,981,813,616 *distinct* possible positions (up to symmetry)

Օրինակ
Route-finding - Romania Map



–**States** *("Current location")*

–The **initial state** *("Some city")*

–A test for the **goal** *("Are we in Bucharest?")*

–Available **actions** at any state *("Go to a neighboring city")*

–A **transition model** *("Update current location")*

–A **cost measure** for actions *("Distance / fuel?")*

Simplifying Assumptions Պարզեցնող ենթադրություններ

In all these examples we have assumed that:

–The environment is **static** - ստատիկ միջավայր

- (i.e., nothing changes while we choose and execute actions)

– The environment is **fully observable** - ամբողջովին տեսանելի

- (i.e. the agent has full knowledge of the state it is in)

–The environment is **deterministic** - որոշիչ/դետերմինիստիկ

- (i.e. the result of a given action in a given state is fixed)

–The environment is **discrete** - դիսկրետ միջավայր

- (i.e. we can consider discrete states at discrete time steps)

–There is only a **single agent** - մեկ գործակալ

Many real problems are dynamic, partially observable, stochastic, continuous and involve multiple agents

Search
Որոնում

The basic idea is to explore the state space of a problem by generating the states that are reachable from the current state (known as expanding the state) and systematically examining them in some order

Հիմնական գաղափարն այն է ուսումնասիրել խնդրի վիճակների բազմությունը ստեղծելով վիճակներ, որոնք հասանելի են ներկա վիճակից (վիճակի ընդհարձակում) եւ կանոնավոր կերպով ուսումնասիրել դրանք որոշակի հերթականությամբ:

Tree-search algorithm

frontier: a collection of nodes waiting to be explored

node: a data structure with information about a state and the path to it

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

set *frontier* to {{node(**initial state**, *empty path*)}}

repeat

choose a node from *frontier* (and remove it)

if the node contains a **goal state**

return the **solution**

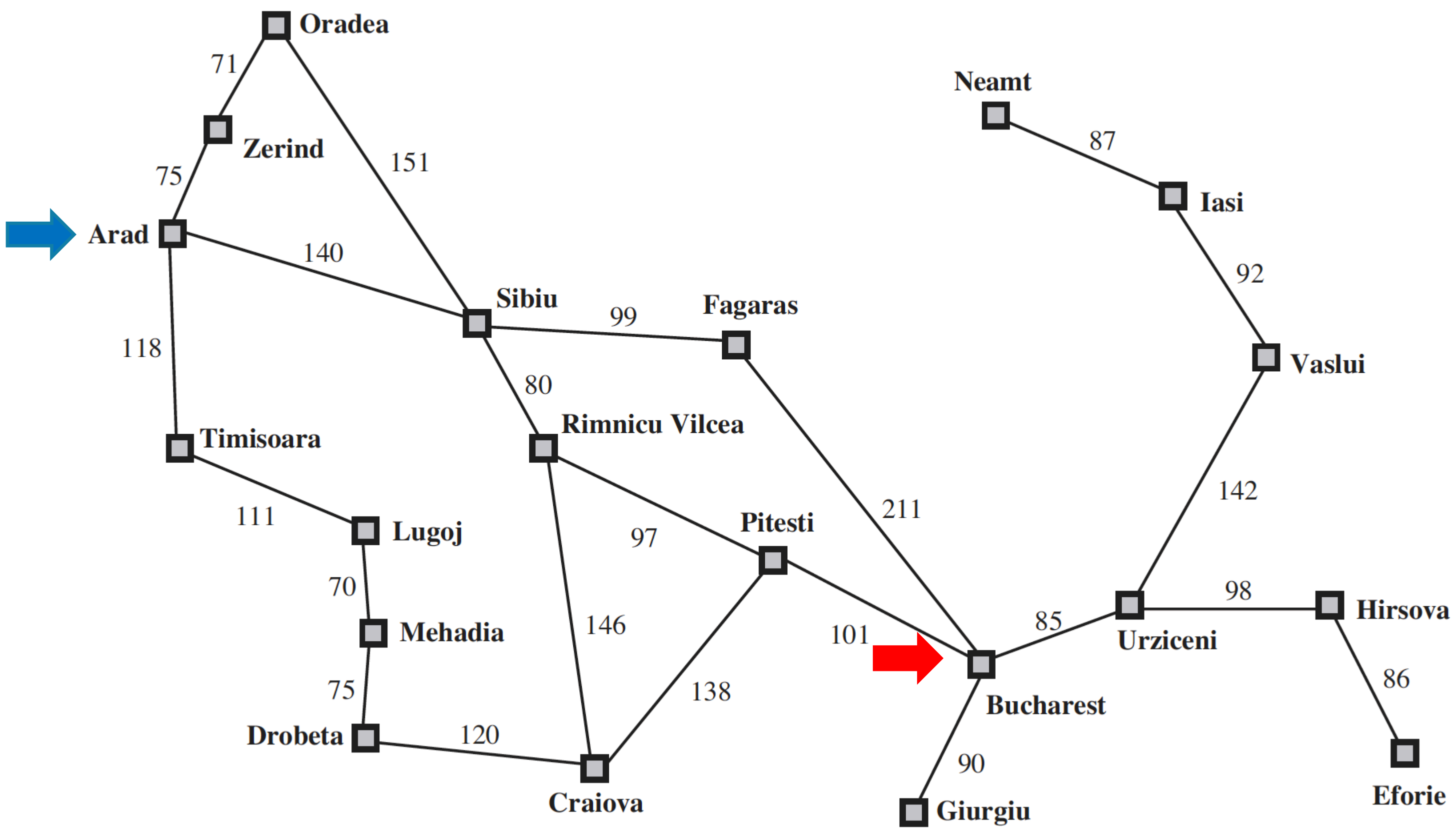
expand the chosen node, adding the resulting nodes to the *frontier*

until *frontier* is empty

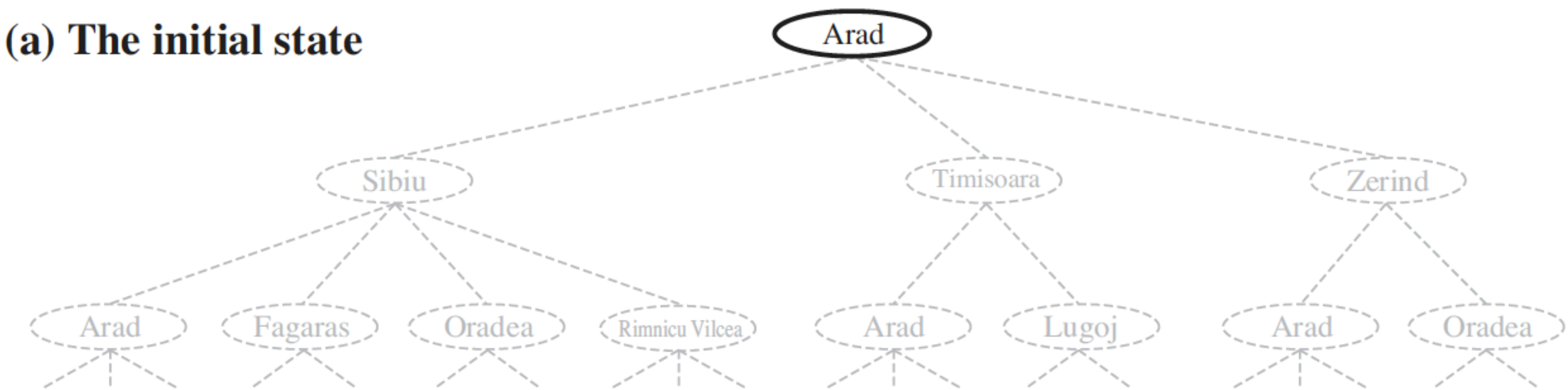
return failure

which node is chosen is determined by the search strategy...

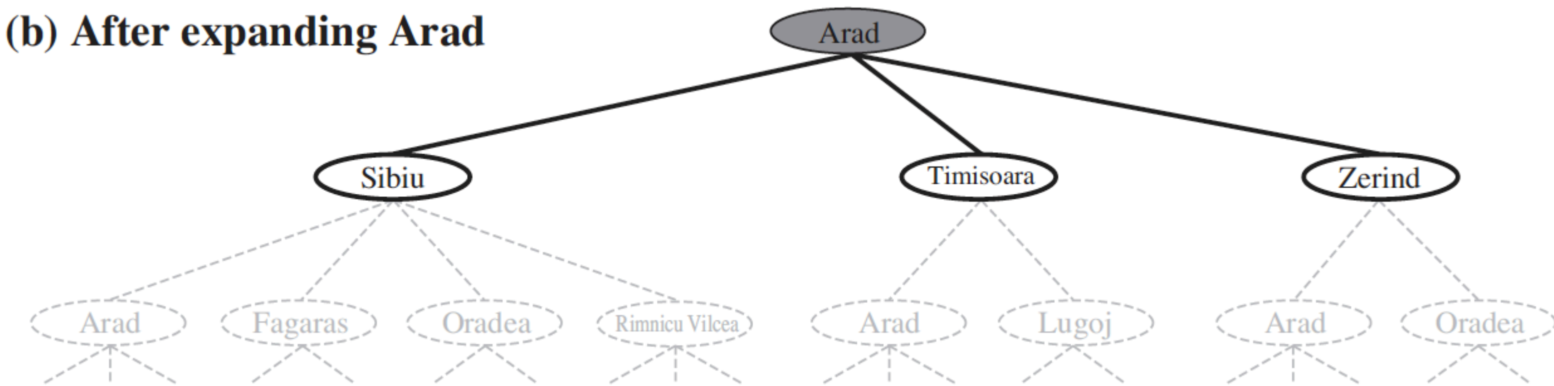
Route-finding
Arad to Bucharest



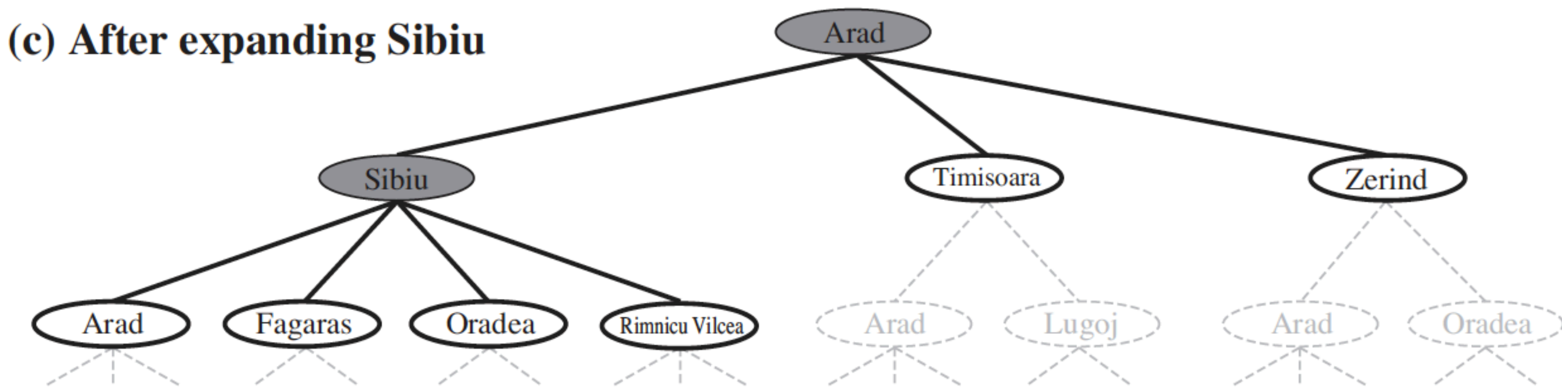
Route-finding
Search tree



Route-finding
Search tree



Route-finding
Search tree



Node/հանգույց

Nodes are the data structures from which the search tree is constructed

For each node **n** of the tree, we have a structure that contains **4** components:

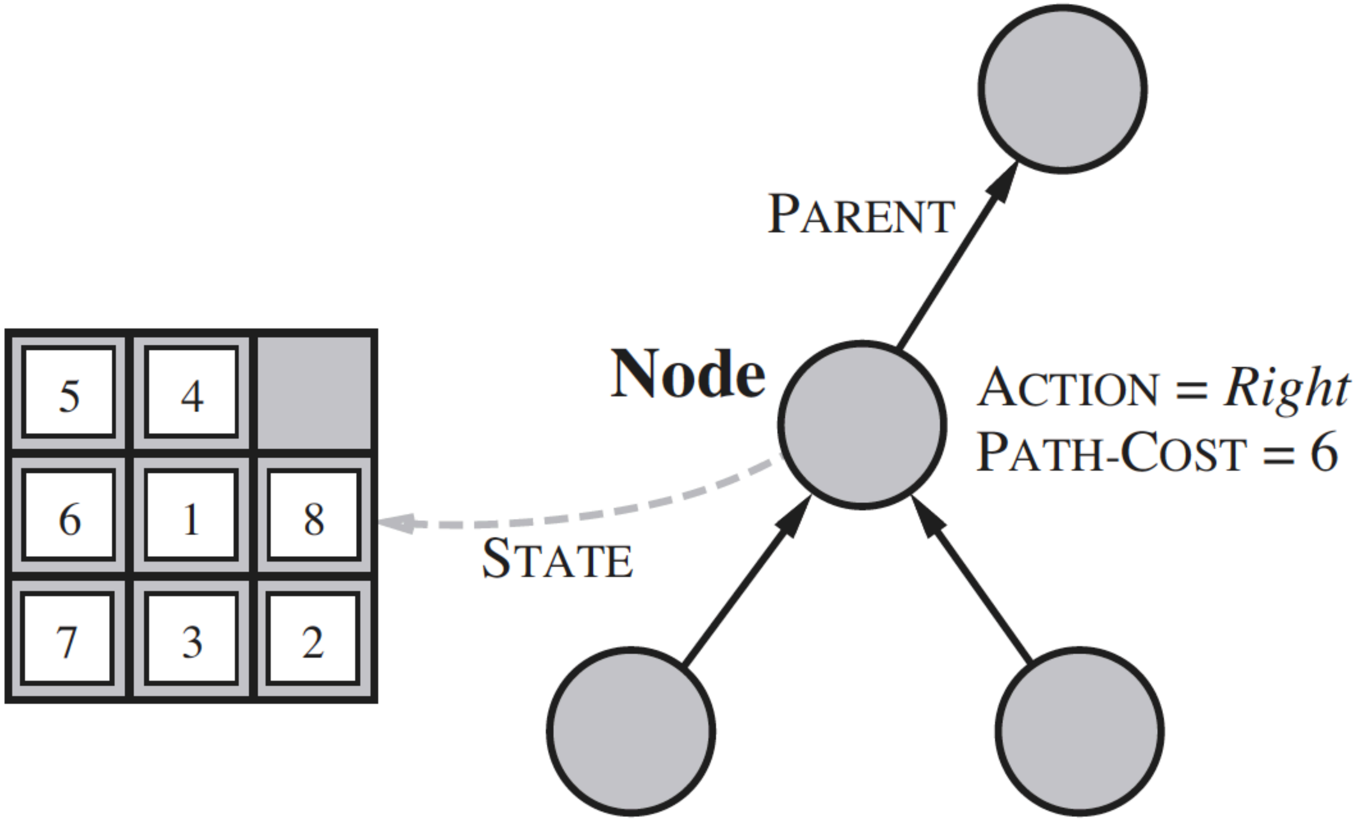
n.STATE: the state in the state space to which the node corresponds;

n.PARENT: the node in the search tree that generated this node;

n.ACTION: the action that was applied to the parent node;

n.PATH-COST: the cost of the path from the initial state to the node.

Node/հանգույց



Tree-search

The basic idea is to explore the state space of a problem by generating the states that are reachable from the current state (known as expanding the state) and systematically examining them in some order

function TREE-SEARCH(*problem*) returns a *solution* or *failure*
    set *frontier* to {{node(*initial state*, *empty path*)}}

    repeat
        choose a node from *frontier* (and remove it)
        if this node contains a goal state then **return** *solution*

        expand this node, adding resulting nodes to the *frontier*

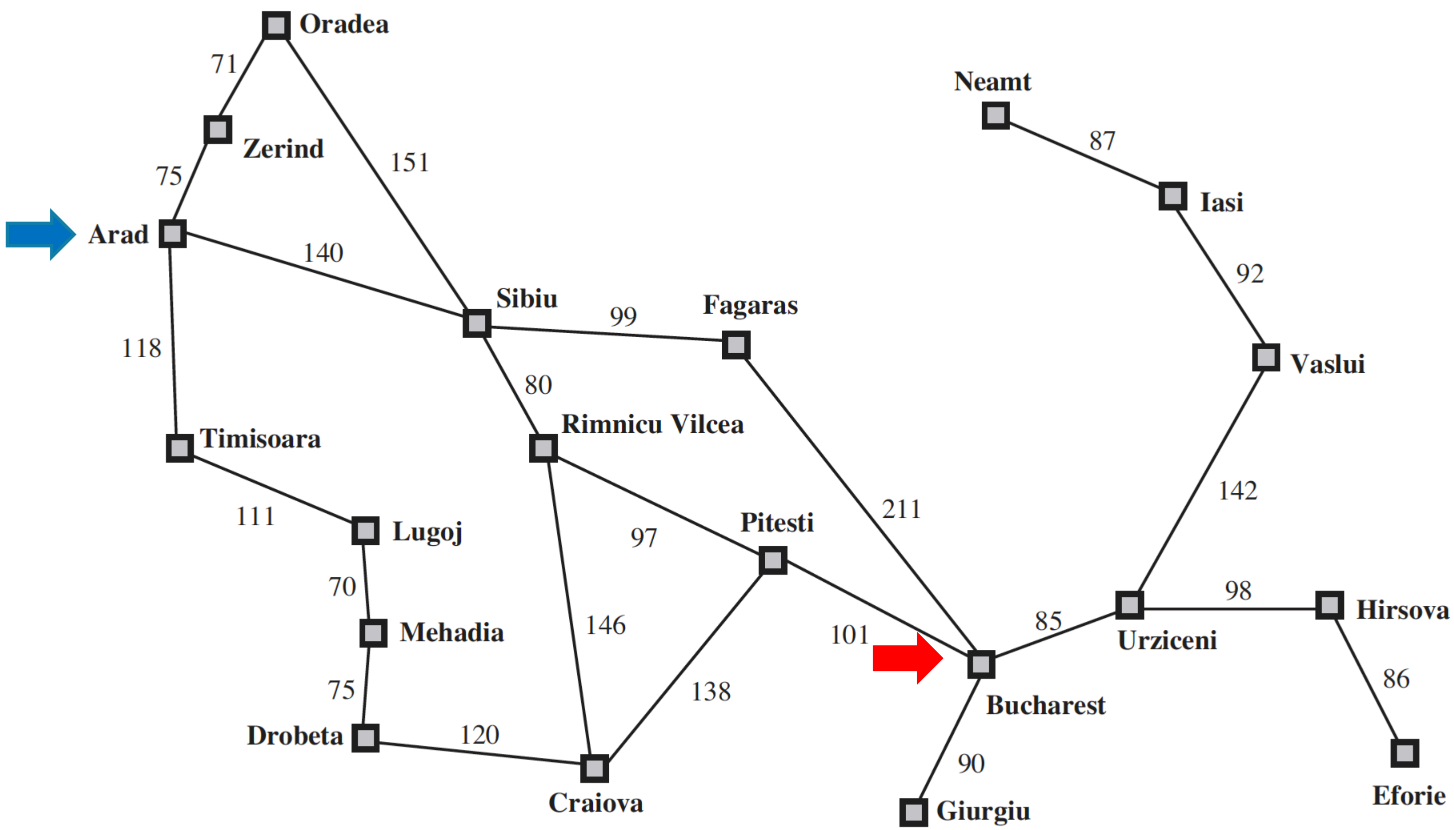
    until *frontier* is empty
    return *failure*

Graph-search

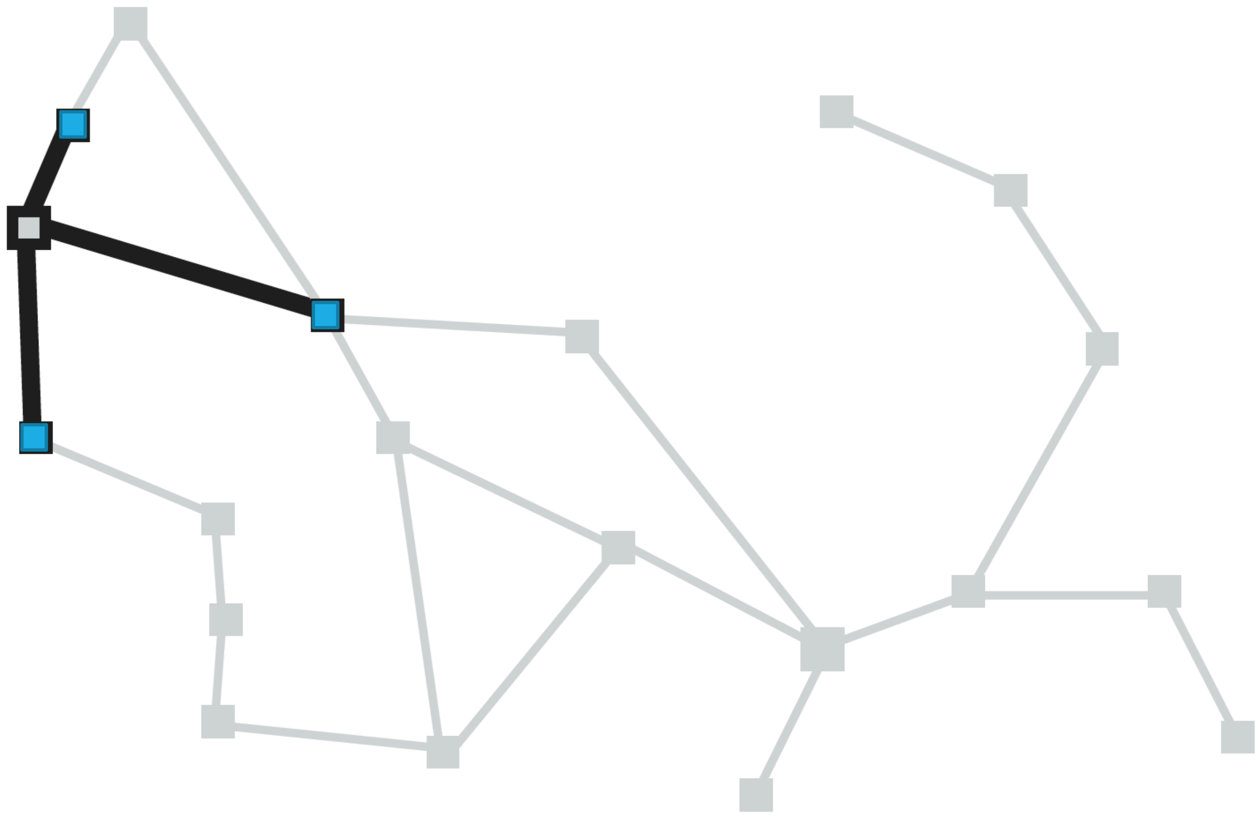
The basic idea is to explore the state space of a problem by generating **new** states that are reachable from the current state (known as expanding the state) and systematically examining them in some order

```
function GRAPH-SEARCH(problem) returns a solution or failure
    set frontier to {{node(initial state, empty path)}}
    set explored_set to an empty set (of states)
    repeat
        choose a node from frontier (and remove it)
        if this node contains a goal state then return solution
        add (state of) this node to the explored_set
        expand this node, adding resulting nodes to the frontier
            (if state not already in the explored_set or frontier)
    until frontier is empty
    return failure
```

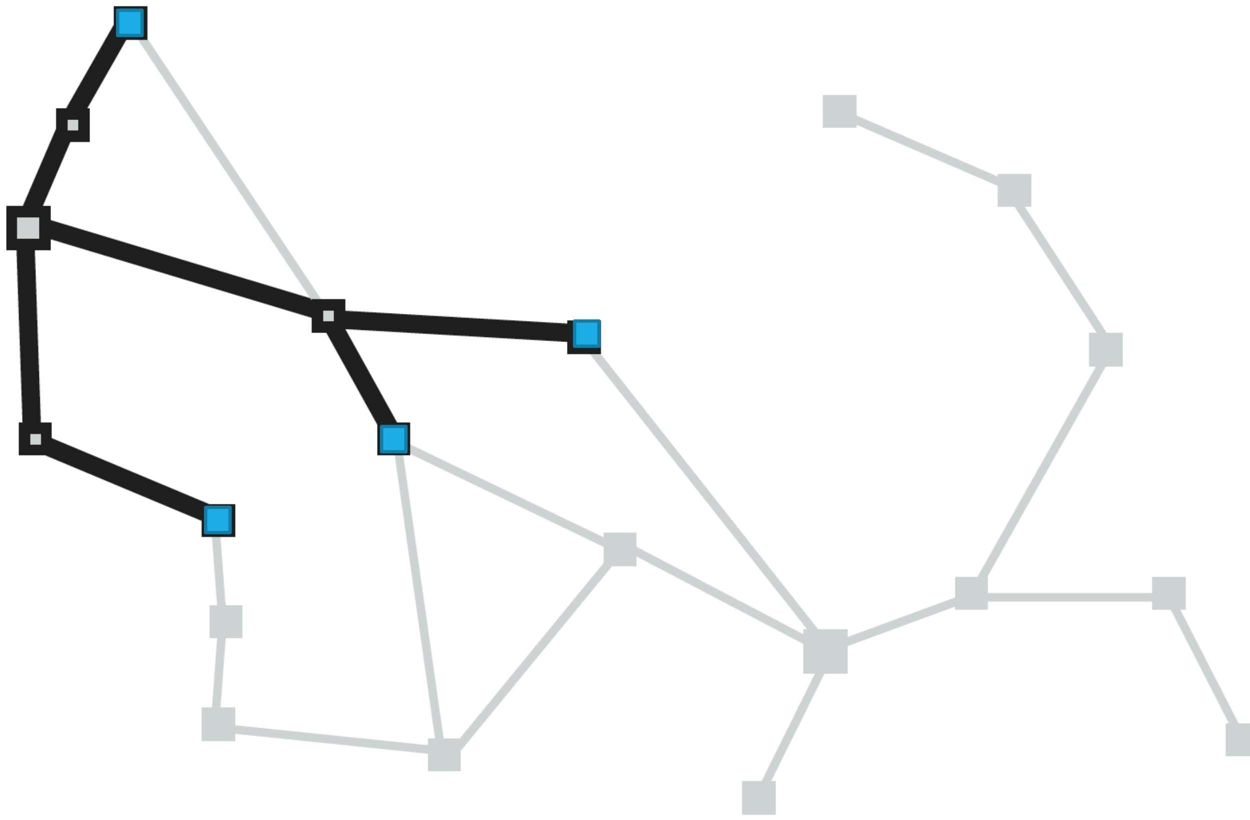
Route-finding with graph-search



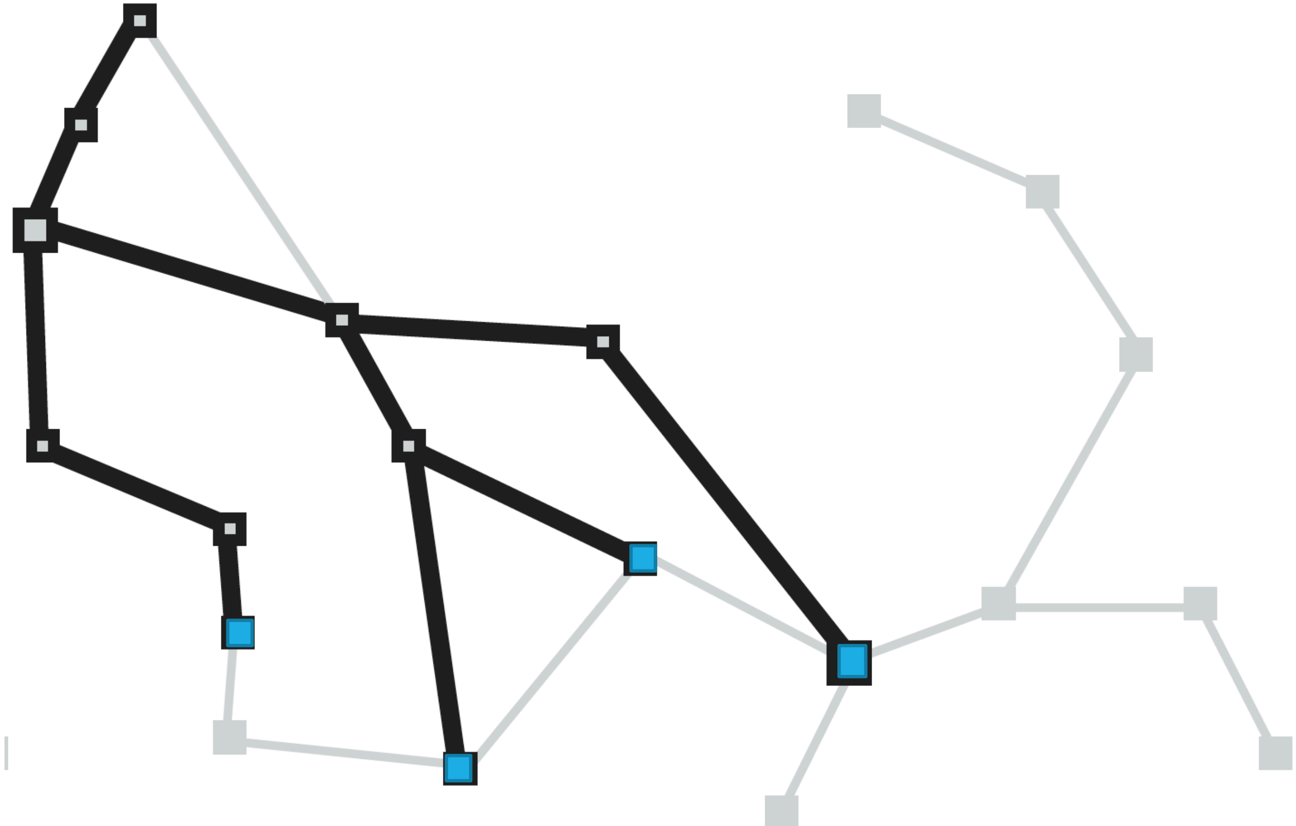
Route-finding with graph-search



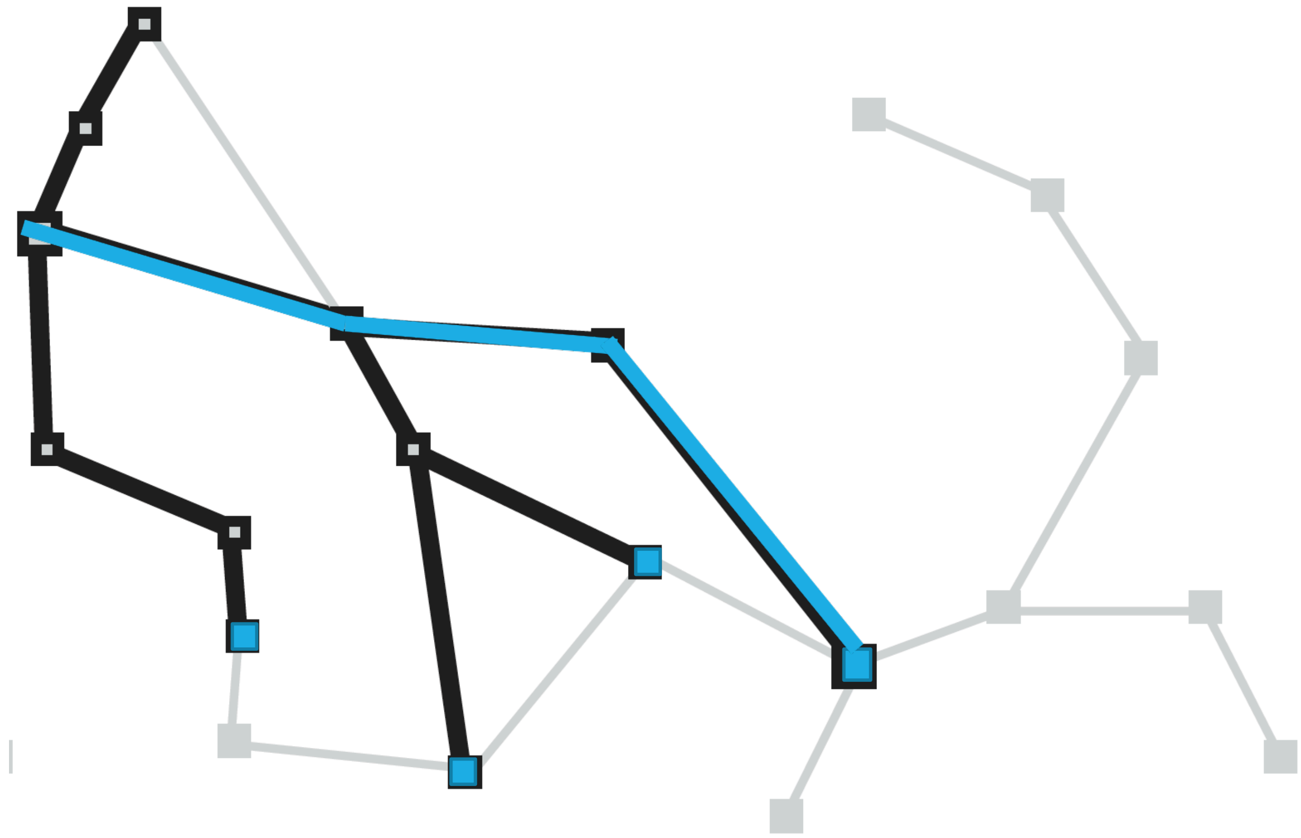
Route-finding with graph-search



Route-finding with graph-search



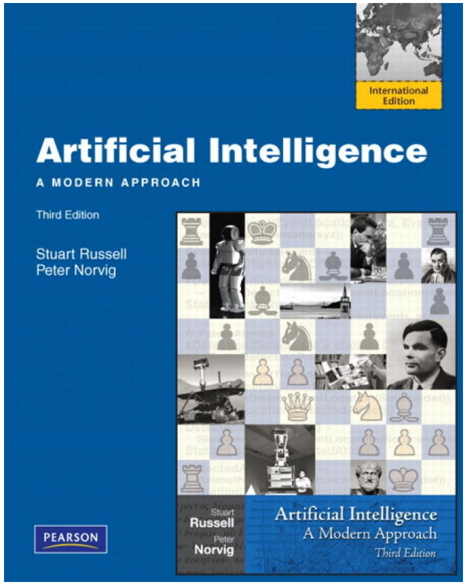
Route-finding with graph-search



Measuring problem-solving performance

1. **Completeness**: Is the algorithm guaranteed to find a solution when there is one? - Լրիվություն
2. **Optimality**: Does the strategy find the optimal solution? - Օպտիմալություն
3. **Time complexity**: How long does it take to find a solution? - Ժամանակի բարդություն
4. **Space complexity**: How much memory is needed to perform the search? – Հիշողության բարդություն

Reading List
Կարդալ



Chapter 3.1 – 3.3

https://github.com/samvelyan/Intelligent-Systems

Next Lecture

1. (Repeat) algorithm complexities.
2. Uninformed search strategies.
3. Coding! We'll try to solve 8-puzzle and route-finding problems! (repeat Java)

Questions?
Հարցե՞ր