Základy umělé inteligence

B4B36ZUI, LS 2019

Martin Schaefer, Jaromír Janisch, Tomáš Krajník
{name.surname}@agents.fel.cvut.cz
Artificial Intelligence Center, Czech Technical University

Course Outline

Lectures

- Tuesday 14:30 - 16:00
- Prof. Michal Pěchouček, Doc. Jiří Kléma, Tomáš Krajník

Seminars

- Krajník, Schaefer, Janisch; Kléma, Kunc; Vintr
- Assignments – **a lot of programming**, Java knowledge required

CourseWare

- https://cw.fel.cvut.cz/wiki/courses/b4b36zui/start
- Course materials
- Course requirements

Contacts

- Forum, emails of lecturers on cw

Seminars Credit Requirements

Participation and active work at all seminars (up to 2 absences without an excuse are allowed).

Gathering at least 25 points from the assignments (out of 50).

Submitting all assignments during the term and receiving at least 50% of points from each of the assignment for the content

- You can choose **one assignment**, for which you **don't have to get 50% of points.**
- The points for the fulfilling 50% of the assignment are computed before the deduction of points for late submission penalty
- The late submission penalty is 1 point for each day after deadline

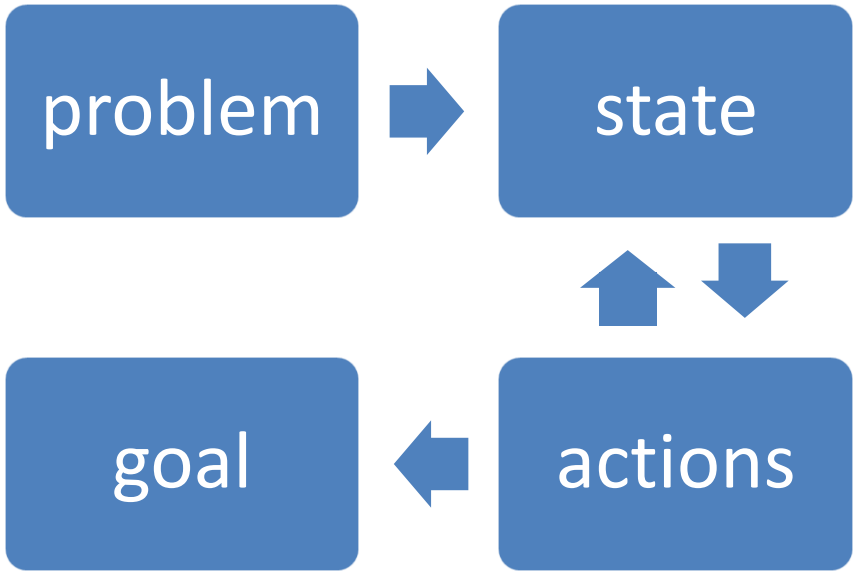
Homework Assignments

A* - finding an optimal path

Constraint Satisfaction Programming

Two-player games

Problem Solving



Search Problem

State space

Problem

Initial state : s_0

Successor function : $x \in S \rightarrow succ(x) \in 2^S$

Goal test : $x \in S \rightarrow goal(x) = T \mid F$

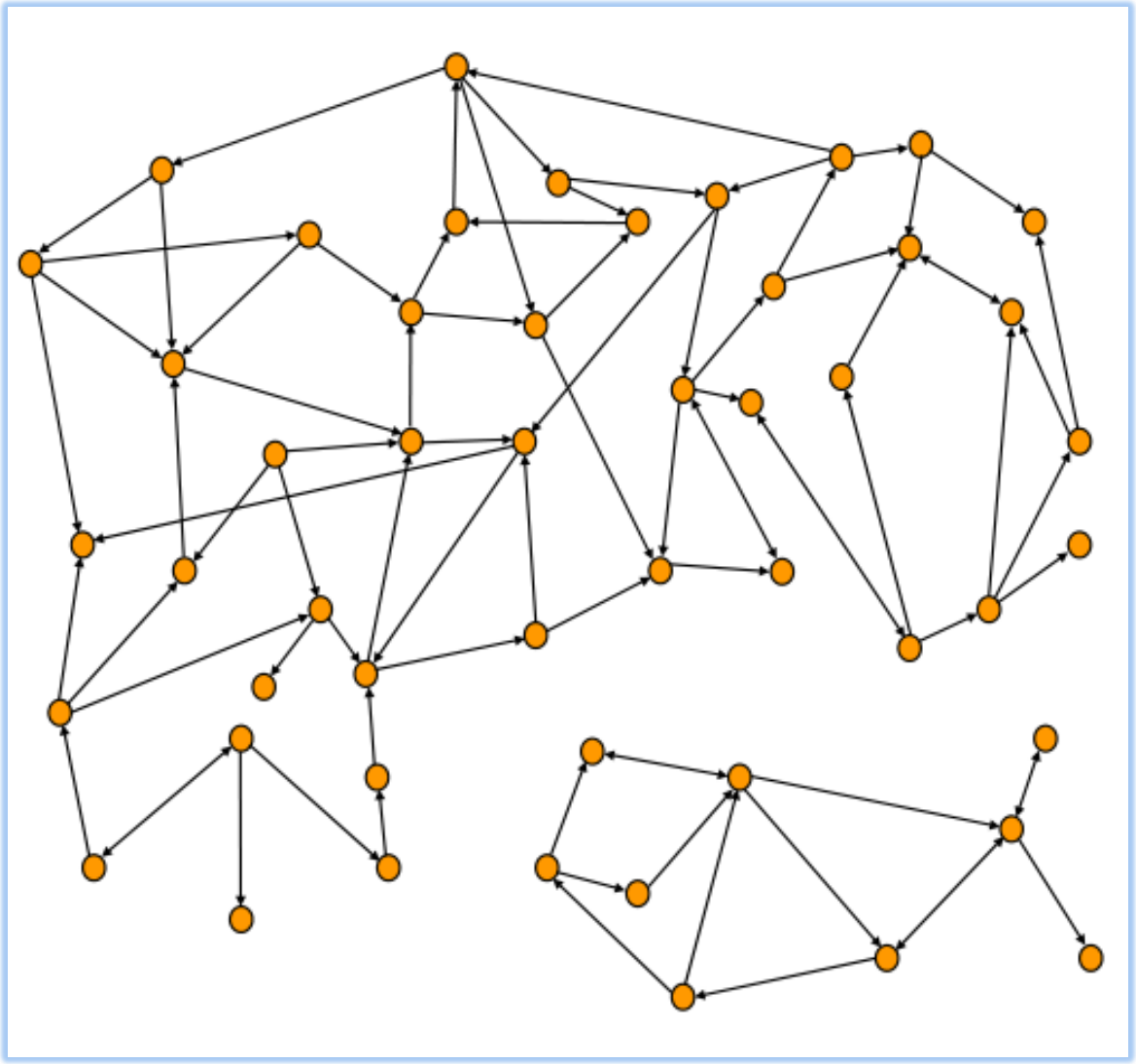
Arc cost : $c(x, succ(x))$

Solution is set of actions leading from initial state to a goal state.

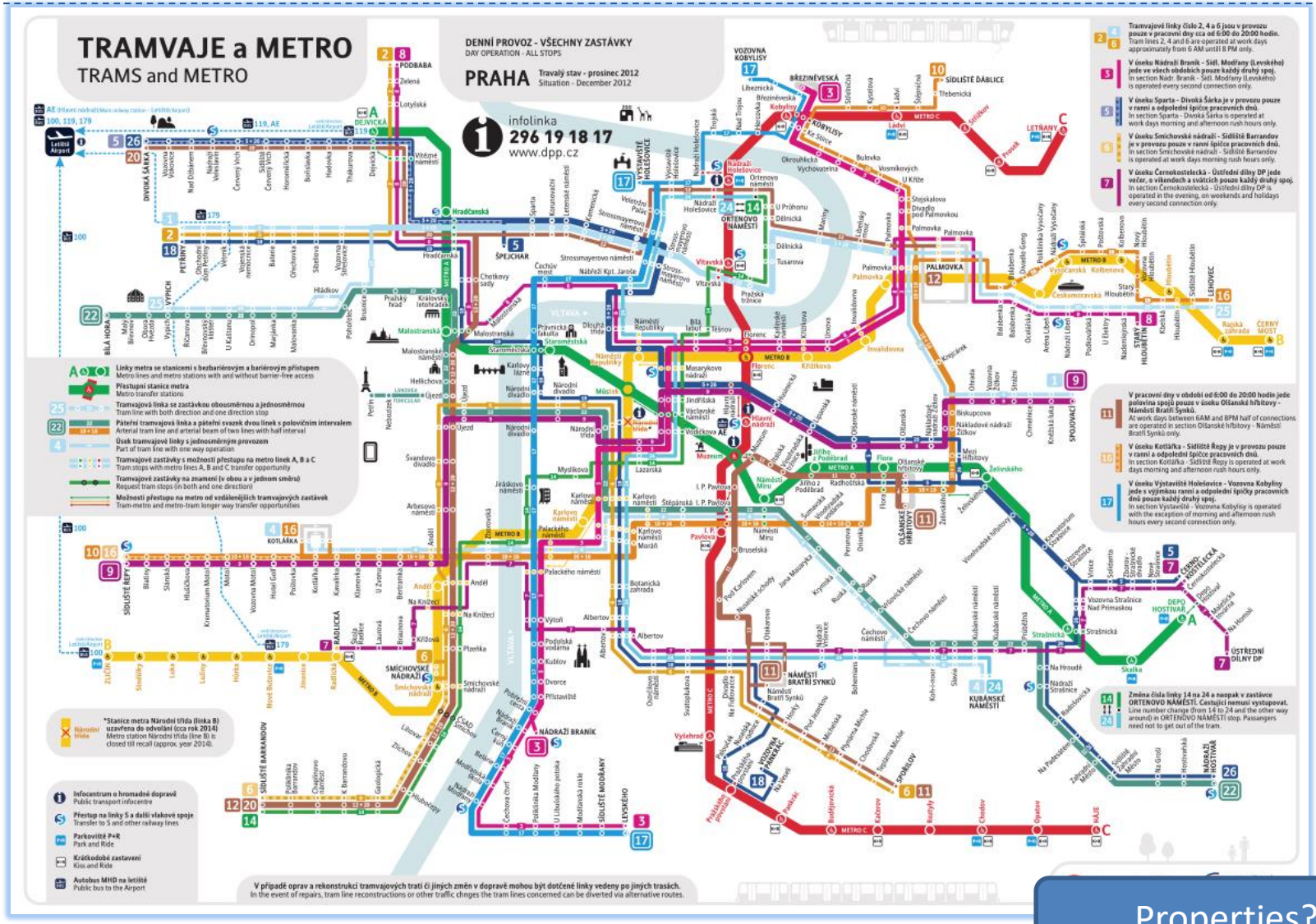
State space is defined by the initial state and successor function

Formalization of the state space is the key to solve (almost) all symbolic AI problems!

State Graph



State Graph


TRAMVAJE a METRO
TRAMS and METRO
DENNÍ PROVOZ - VŠECHNY ZASTÁVKY
DAY OPERATION - ALL STOPS
PRAHA
Travaly stav - prosinec 2012
Situation - December 2012
infolinka
296 19 18 17
www.dpp.cz
Properties?

State Space

Formulation

- **Problem** – shortest path in MHD from KN to Dejvice
- **Initial state** – s_0=Karlovo Namesti
- **Successor function** – $succ(x)$ →all connected stations
- **Goal test** – x=Dejvicka (explicit)
- **Arc cost** – $c(x, y)$=time of transit between stations x,y

- **Solution –** (Karlak-MustekB),(MustekB-MustekA), (MustekA-Staromestska),…, (Hradcanska,Dejvicka)

State Space

Examples

- Traveling problem
- from Karlak to Dejvice
- from Prague to Snezka
- from Prague to Sydney

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 kth 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?

Search Problem

State space

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 : $c(x, succ(x))$

Solution is set of actions leading from initial state to a goal state.

State space is defined by the initial state and successor function

Tree Search Algorithm

Basic Idea

Basic idea:

offline, 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
  end
```

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
```

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*)

BFS/DFS Exercises