Heuristic Search and Evolutionary Algorithms

启发式搜索与演化算法

Chao Qian （钱超）

Associate Professor, Nanjing University, China

Email: qianc@nju.edu.cn

Homepage: http://www.lamda.nju.edu.cn/qianc/

课程相关信息

课程时间地点：周五下午16:00-18:00

课程主页：

http://www.lamda.nju.edu.cn/HSEA21/

课程讨论QQ群：972364375

助教：薛轲、刘丹璇

每个ppt的最后附有相关参考文献

答疑时间：周五下午14:00-16:00、逸A-502

Outline of this course

- **Part 1: Traditional heuristic search algorithms**

  (Assignment 1: 15%)

- **Part 2: Evolutionary algorithms** (Assignment 2: 15%)

- **Part 3: Theoretical analysis of evolutionary algorithms** (Assignment 3: 15%)

- **Part 4: Design of evolutionary algorithms**

  (Assignment 4: 15%)

Final exam: 40%

南京大学 人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE, NANJING UNIVERSITY

南京大學

Heuristic Search and Evolutionary Algorithms

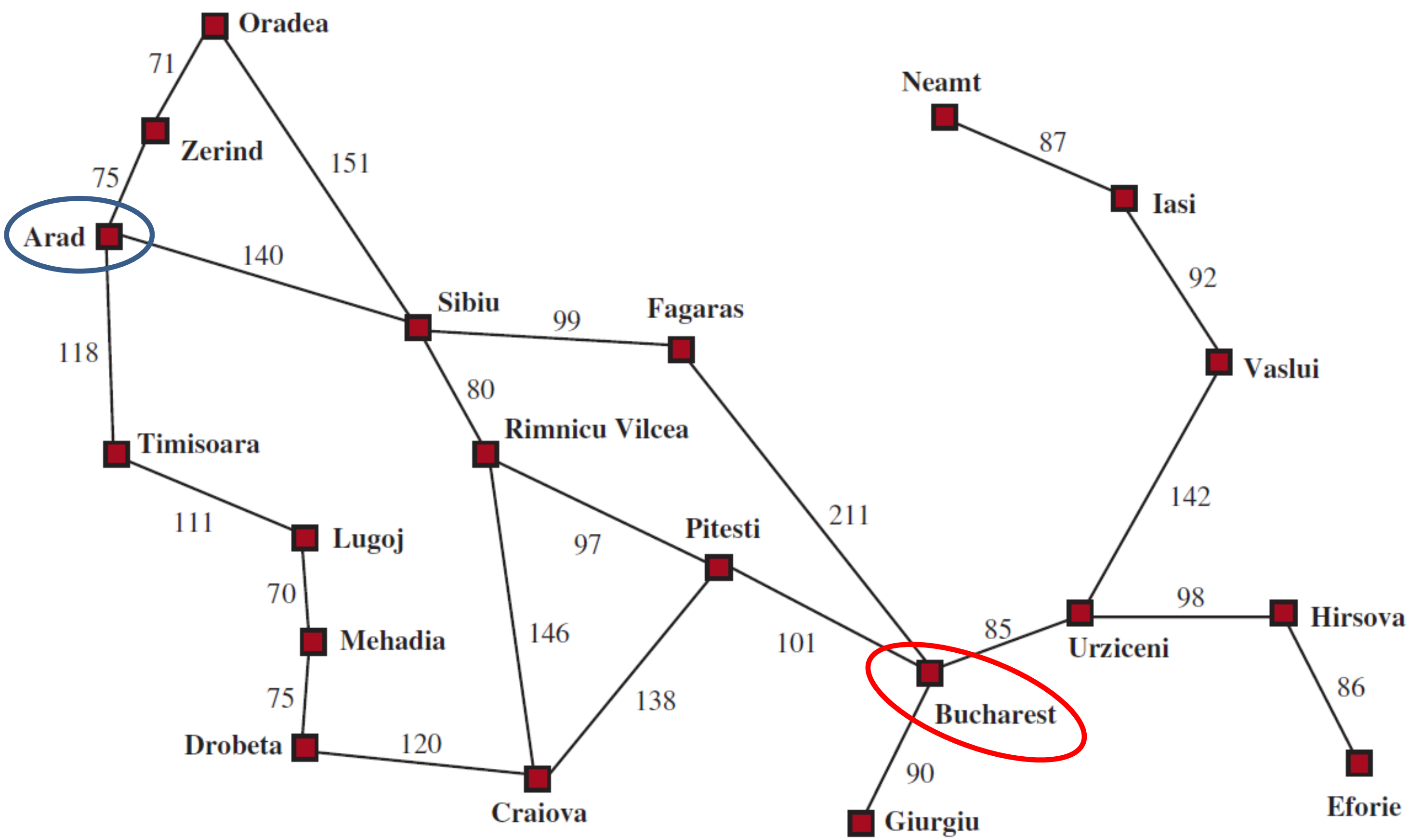
Lecture 1: Search

Chao Qian （钱超）

Associate Professor, Nanjing University, China

Email: qianc@nju.edu.cn
Homepage: http://www.lamda.nju.edu.cn/qianc/

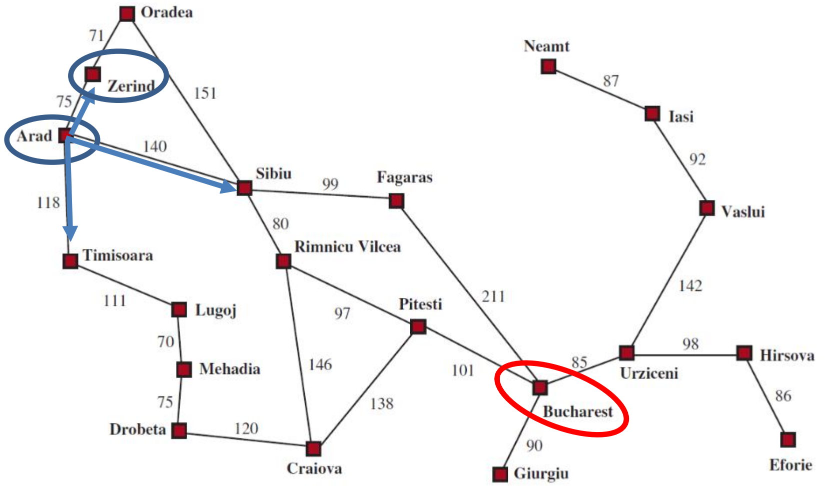
Search example – route finding



Search problem

A search problem can be defined formally by five components:

- Initial state *e.g., In(Arad)*
- Actions *e.g., Go(Sibiu), Go(Timisoara), Go(Zerind)*
- Transition model *e.g., Result(In(Arad), Go(Zerind))=In(Zerind)*
- Goal test *e.g., Is(In(Bucharest))*
- Path cost *e.g., the sum of action costs*



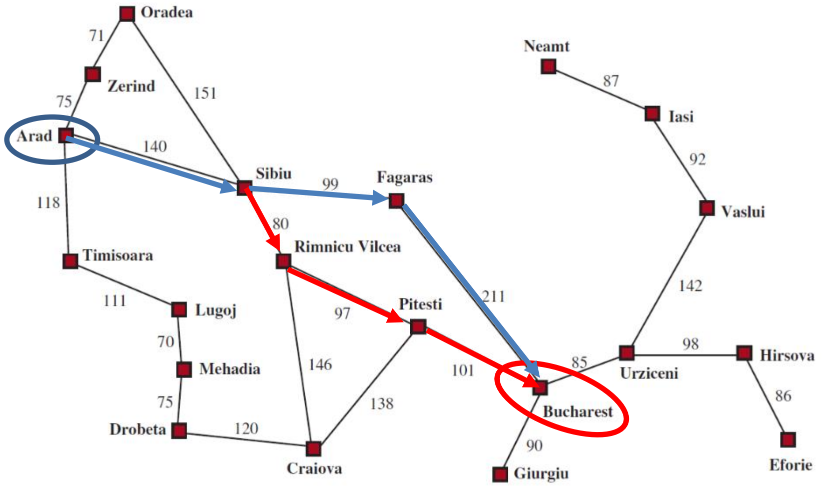
Search problem

A search problem can be defined formally by five components:

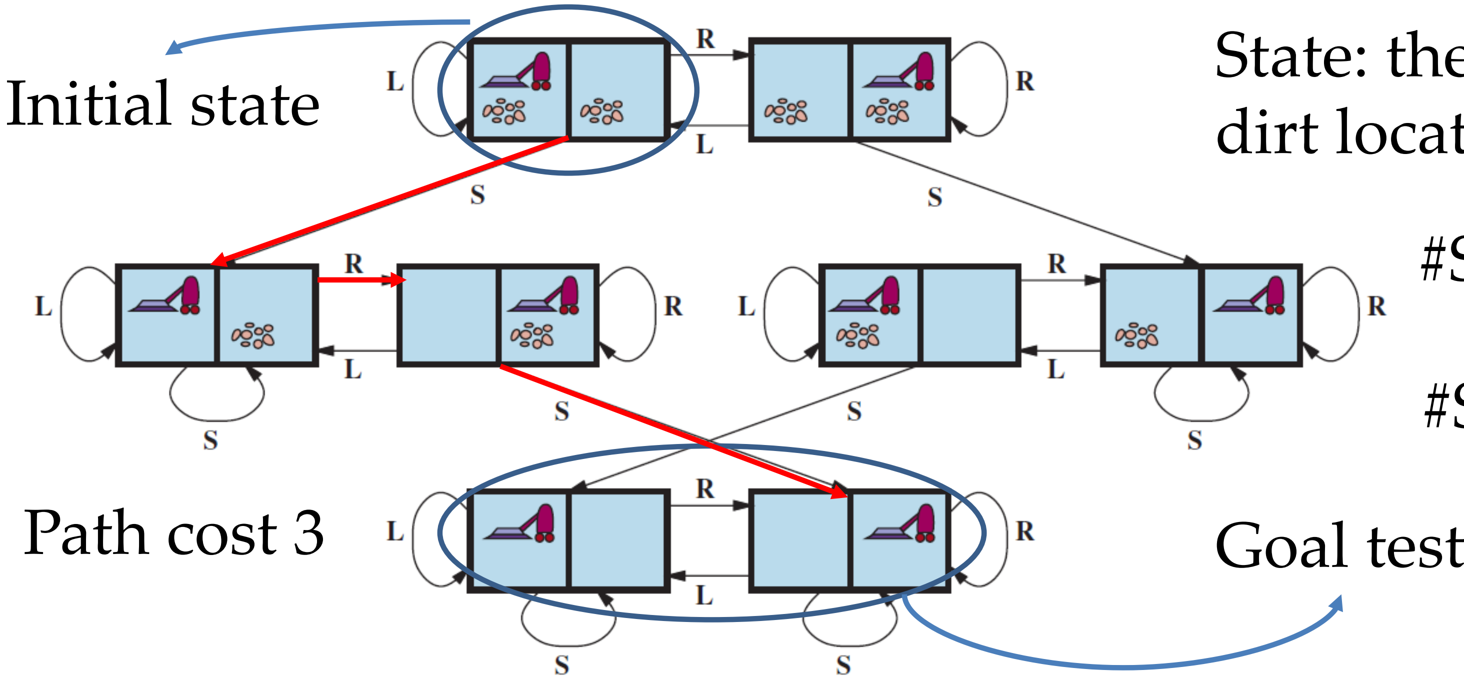
- Initial state — *e.g., In(Arad)*
- Actions — *e.g., Go(Sibiu), Go(Timisoara), Go(Zerind)*
- Transition model — *e.g., Result(In(Arad), Go(Zerind))=In(Zerind)*
- Goal test — *e.g., Is(In(Bucharest))*
- Path cost — *e.g., the sum of action costs*

Solution: a path (i.e., an action sequence) from the initial state to the goal state

Optimal solution: a path with the lowest cost



More examples – vacuum world



State: the agent and dirt location

#States: $2 \cdot 2^2$

#States: $n \cdot 2^n$

Actions: *Left (L), Right (R), Suck (S)*

Transition model: *e.g., Result(Initial state, L) = Initial state*

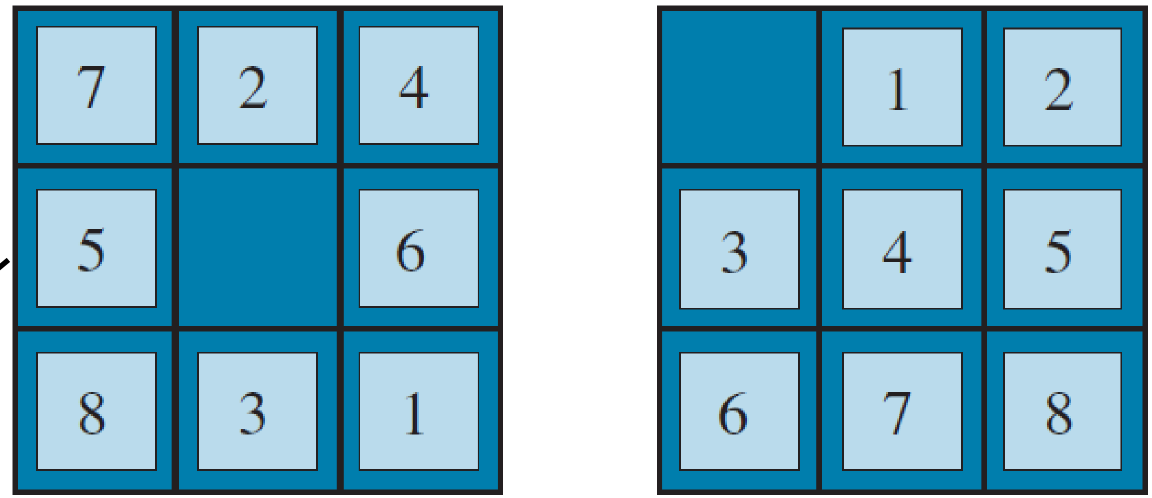
Path cost: *the number of actions on the path*

More examples – 8-puzzle

Left

Result(Start state, Left)

7	2	4
	5	6
8	3	1



Start State Goal State

Actions: *movements of blank space, i.e., Left, Right, Up and down*

Path cost: *the number of actions on the path*

More examples – integer construction

Problem: starting with the number 4, apply a sequence of factorial, square root, and floor operations to reach any desired positive integer

- Initial state: *4*
- Actions: *factorial, square root, and floor operations*
- Transition model: *e.g., Result(4, factorial)=24*
- Goal test: *Is(the desired positive integer)*
- Path cost: *the number of actions on the path*

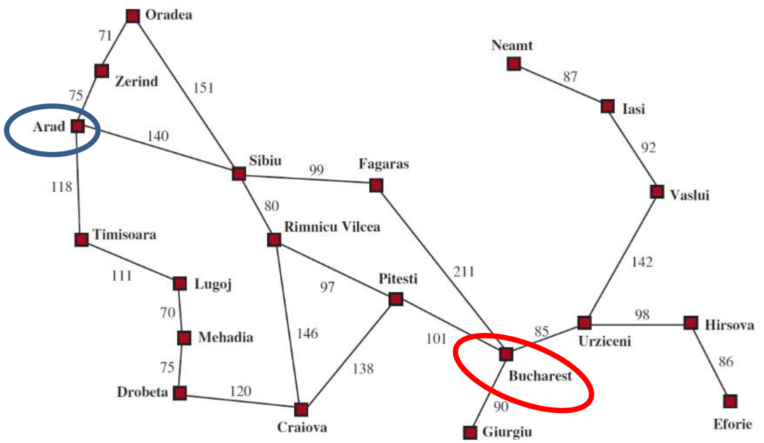
$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

Path cost 8

Infinite state space: positive numbers

More examples – route finding

Problem: find the shortest path between two cities

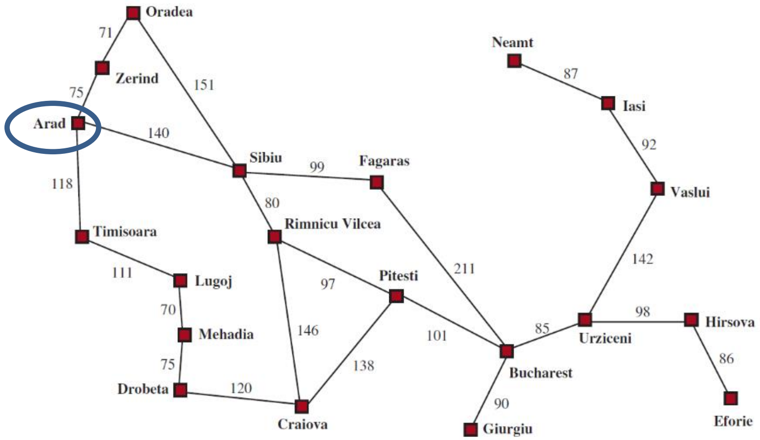


State: *e.g., In(Oradea)*

- Initial state — *e.g., In(Arad)*
- Actions — *e.g., Go(Sibiu), Go(Timisoara), Go(Zerind)*
- Transition model — *e.g., Result(In(Arad), Go(Zerind))=In(Zerind)*
- Goal test — *e.g., Is(In(Bucharest))*
- Path cost — *e.g., the sum of action costs*

More examples – touring

Problem: find the shortest route to visit each city at least once, starting and ending in the same city

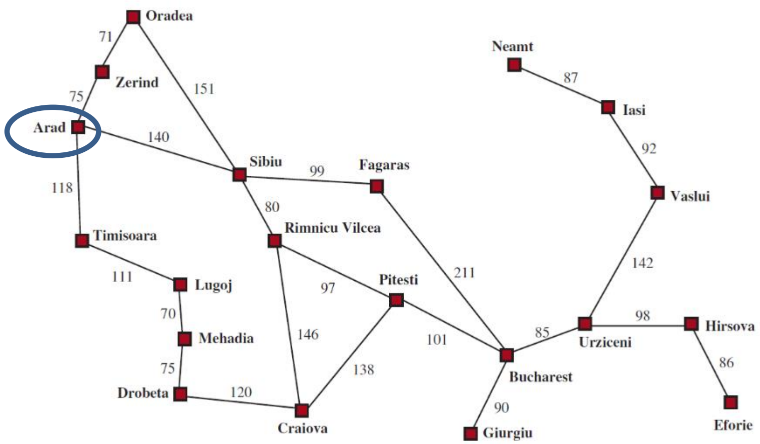


State: *e.g., In(Oradea), Visited({Arad, Zerind, Oradea})*

- Initial state — *e.g., In(Arad), Visited({Arad})*
- Actions — *e.g., Go(Sibiu), Go(Timisoara), Go(Zerind)*
- Transition model — *e.g., Result(In(Arad), Visited ({Arad}), Go(Zerind))*
  =In(Zerind), Visited({Arad, Zerind})
- Goal test — *e.g., Is(In(Arad), Visited({all the cities}))*
- Path cost — *e.g., the sum of action costs*

More examples – traveling salesman

Problem: find the shortest route to visit each city exactly once, starting and ending in the same city



State: *e.g., In(Oradea), Visited({Arad, Zerind, Oradea})*

- Initial state *e.g., In(Arad), Visited({Arad})*
- Actions: *can go to non-visited cities, and return to the origin city at last*
- Transition model *e.g., Result(In(Arad), Visited ({Arad}), Go(Zerind))*
  =In(Zerind), Visited({Arad, Zerind})
- Goal test *e.g., Is(In(Arad), Visited({all the cities}))*
- Path cost *e.g., the sum of action costs*

Search problem

A search problem can be defined formally by five components:

- Initial state
- Actions
- Transition model
- Goal test
- Path cost

Solution: a path (i.e., an action sequence) from the initial state to the goal state

Optimal solution: a path with the lowest cost

Are search problems difficult?

Complexity classes

- Classify problems according to their complexities
- Class: a set of problems
- P, NP, NP-complete, NP-hard

A decision problem is a mapping from all possible inputs into the set {yes, no}

$$f : I \rightarrow \{1,0\}$$

Example of decision problems

- Travelling salesman problem

  Given a weighted graph, a specific vertex (i.e., city), and a positive number k, is there a tour with cost at most k?

  > a tour: starting and ending at the specific vertex after having visited each other vertex exactly once

- Graph coloring

  Given a undirected graph G and a positive integer k, is there a coloring of G using at most k colors?

  > a coloring of G: assigning colors to each vertex of G such that no adjacent vertices get the same color

Decision and optimization problems

There are standard techniques for transforming optimization problems into decision problems

Travelling salesman problem

Optimization version: find the shortest route to visit each city exactly once, starting and ending in the same city

⇩ Try different k

Decision version: given a positive number k, is there such a route with cost at most k?

The class P

The class P contains decision problems that can be solved in polynomial time by a deterministic algorithm

- For any input, the algorithm runs for polynomial time
- For any positive input, the algorithm output “yes”
- For any negative input, the algorithm output “no”

The class NP

Nondeterministic algorithm

```
void nondetA(String input)
    String s=genCertif();
    Boolean CheckOK=verifyA(input,s);
    if (checkOK)
        Output “yes”;
    return;
```

Step 1: guess a solution

Step 2: verify the solution

If yes, output “yes”

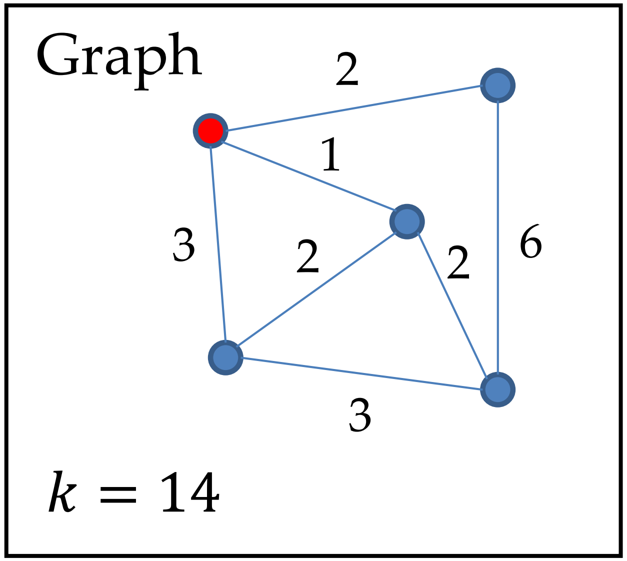
Otherwise, no output

Given the same input, the algorithm may behave differently in different executions

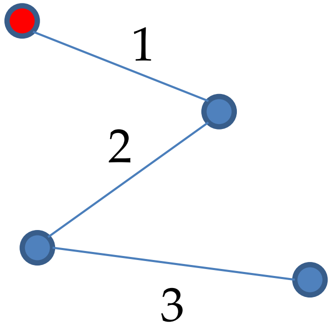
Nondeterministic traveling salesman

- Travelling salesman problem

  Given a weighted graph, a specific vertex (i.e., city), and a positive number k, is there a tour with cost at most k?



Guess:



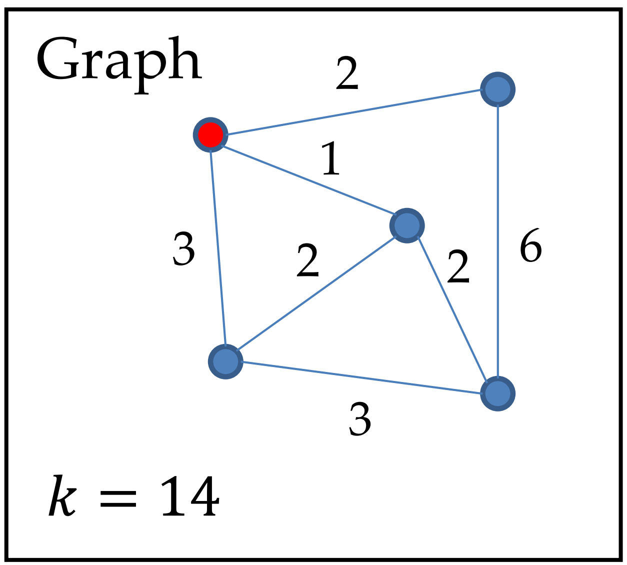
Verify: not a tour

No output

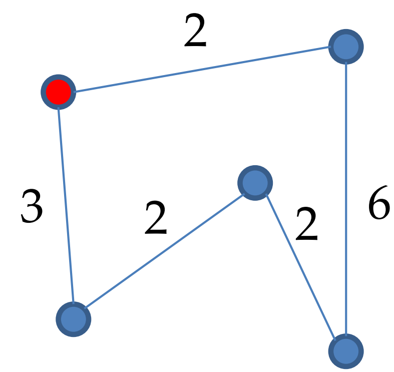
Nondeterministic traveling salesman

- Travelling salesman problem

  Given a weighted graph, a specific vertex (i.e., city), and a positive number k, is there a tour with cost at most k?



Guess:



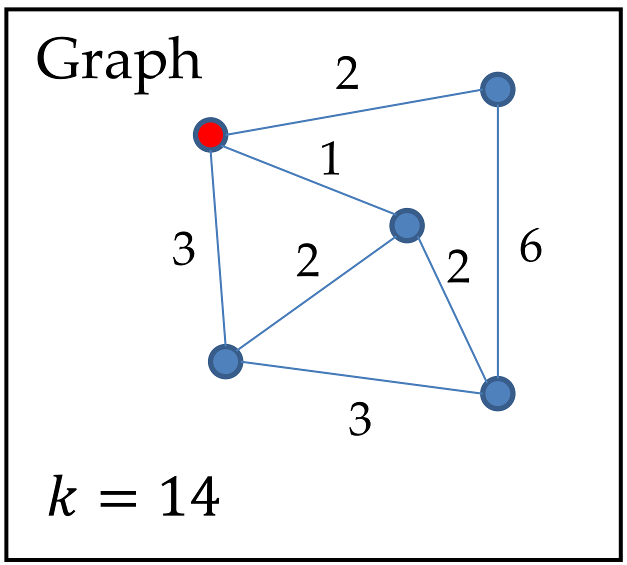
Verify: a tour with cost 15

No output

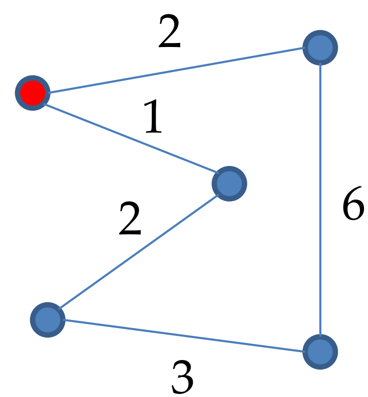
Nondeterministic traveling salesman

- Travelling salesman problem

Given a weighted graph, a specific vertex (i.e., city), and a positive number k, is there a tour with cost at most k?



Guess:



Verify: a tour with cost 14

Output: “yes”

The class NP

The class NP contains decision problems for which there is a polynomial bounded nondeterministic algorithm

- For any positive input, there is some execution of the non-deterministic algorithm which outputs "yes" in polynomial time

$P \subseteq NP$

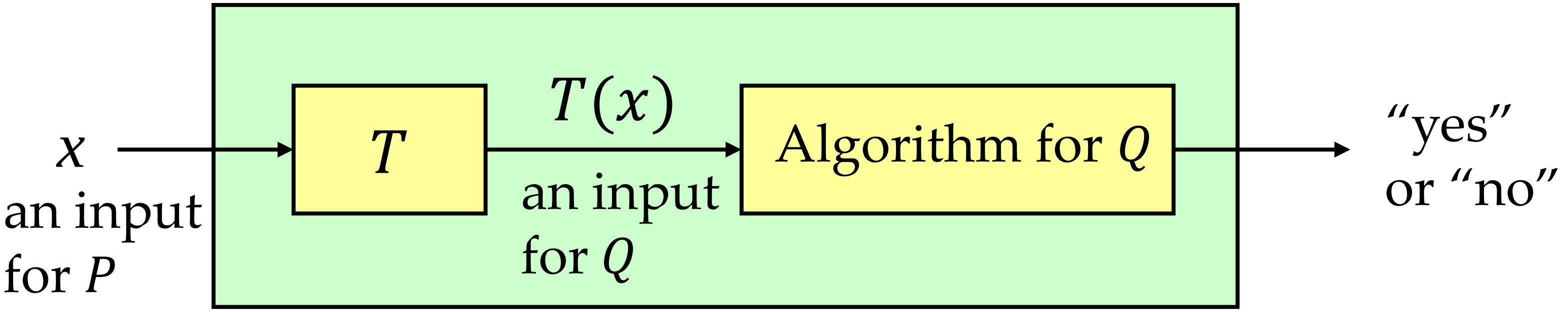
```
void nondetA(String input)
    String s=genCertif();
    Boolean CheckOK=verifyA(input,s);
    if (checkOK)
        Output "yes";
    return;
```

the deterministic polynomial-time algorithm

The class NP-hard

- Let T be a function mapping from the input set of a decision problem P into the input set of Q

- A decision problem P is polynomially reducible to Q if there exists a function T satisfying:
  - ✓ T can be computed in polynomial time
  - ✓ x is a "yes" input for P iff $T(x)$ is a "yes" input for Q



Algorithm for P

The class NP-hard

- A decision problem P is polynomially reducible to Q if there exists a function T satisfying:
  - ✓ T can be computed in polynomial time
  - ✓ x is a "yes" input for P iff $T(x)$ is a "yes" input for Q

Q is at least as hard as P

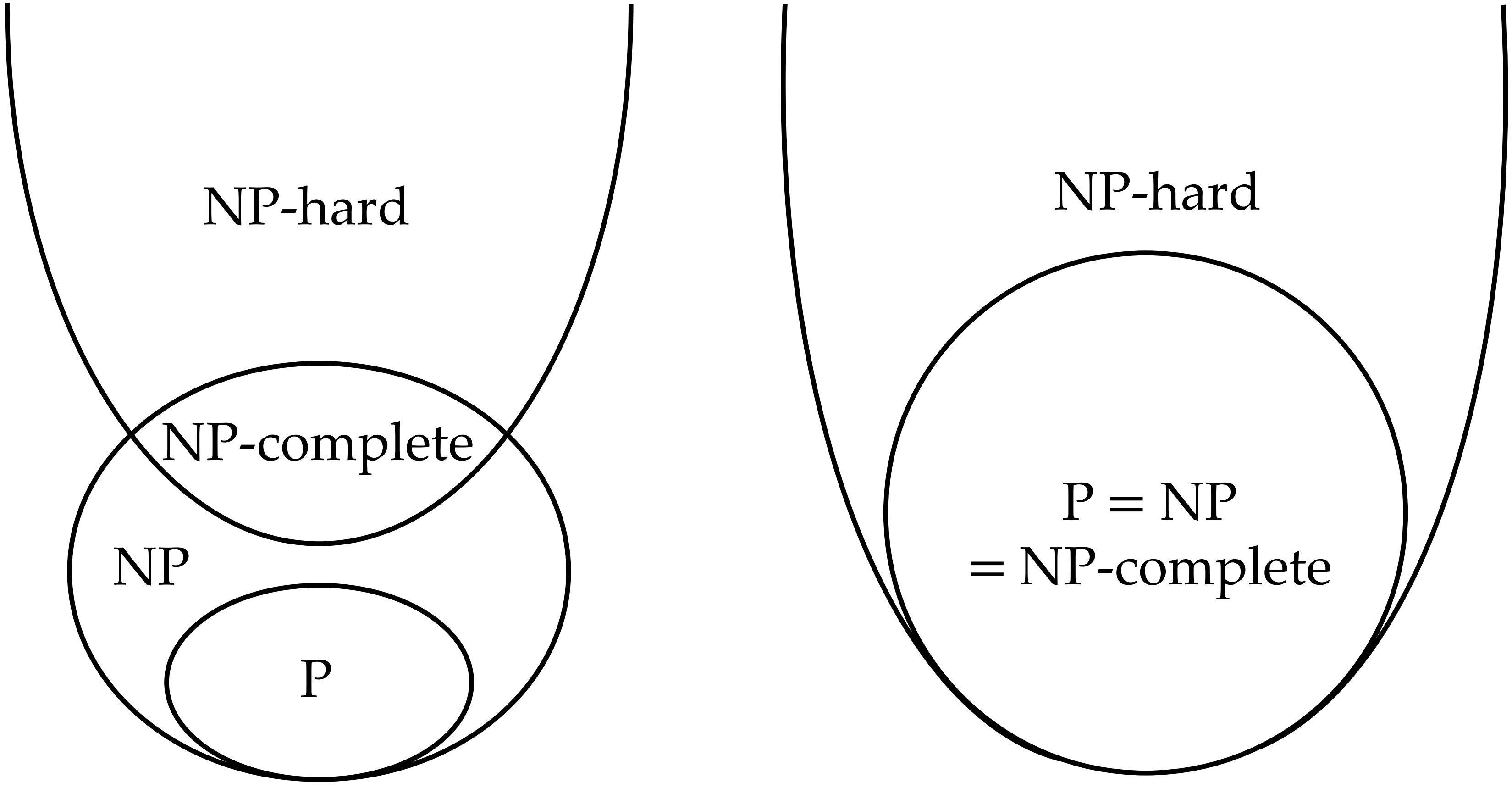
- A problem Q is in NP-hard if every problem P in NP is polynomially reducible to Q

Q is at least as hard as any problem in NP

- A problem is in NP-complete if it is in both NP and NP-hard

the hardest problems in NP

P, NP, NP-complete and NP-hard



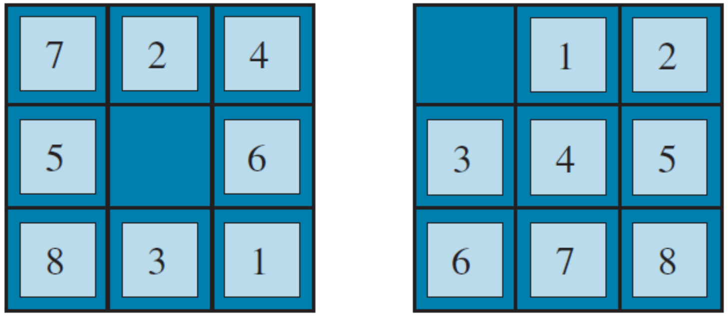
P ≠ NP

P = NP

Hard search problem

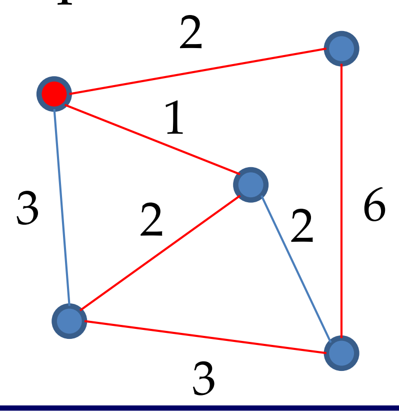
Many search problems are NP-hard, e.g.,

- *n*-puzzle: NP-complete



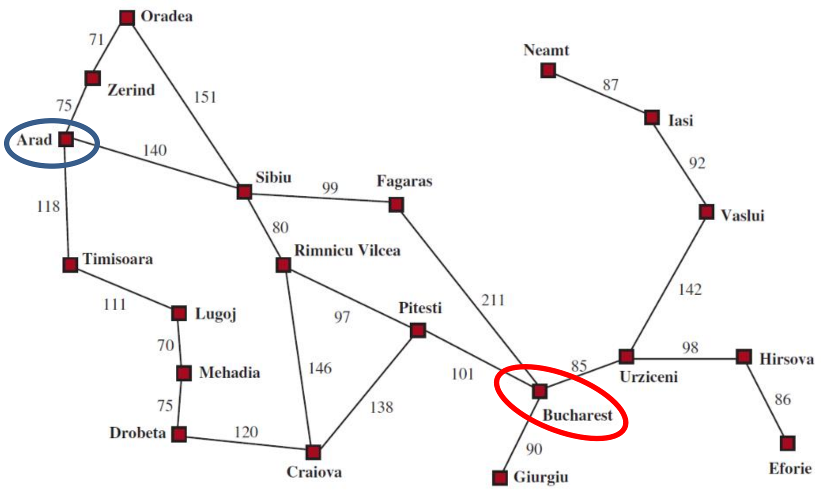
Start State Goal State

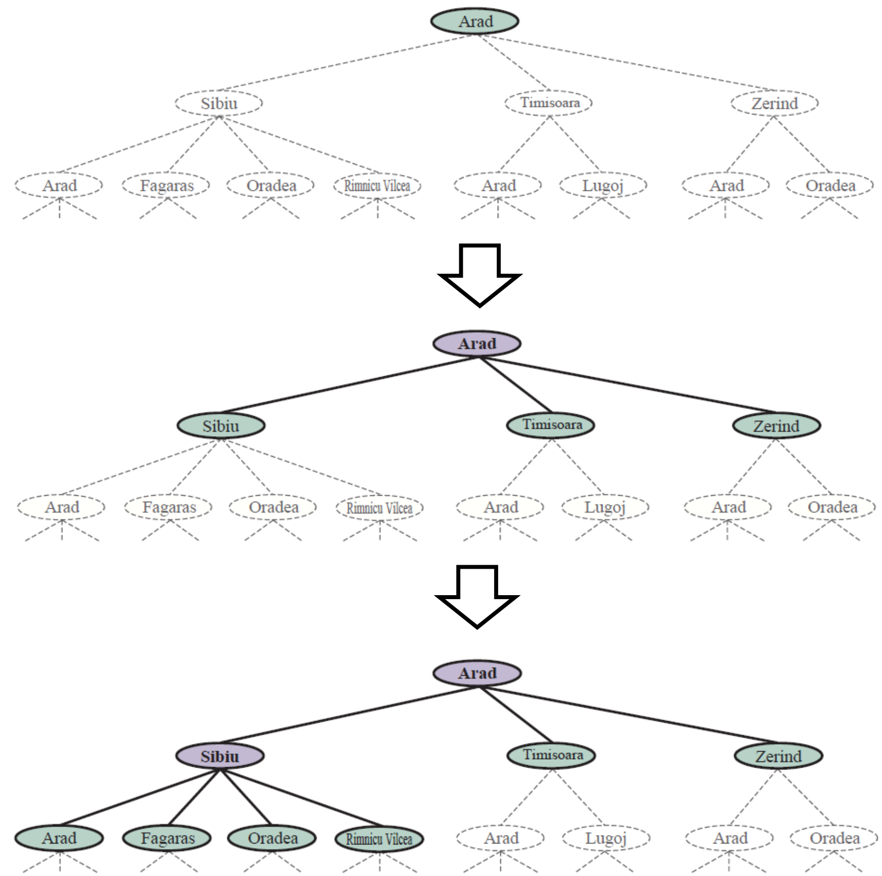
- Travelling salesman problem: NP-hard



Search algorithms

Route finding: the shortest path from Arad to Bucharest



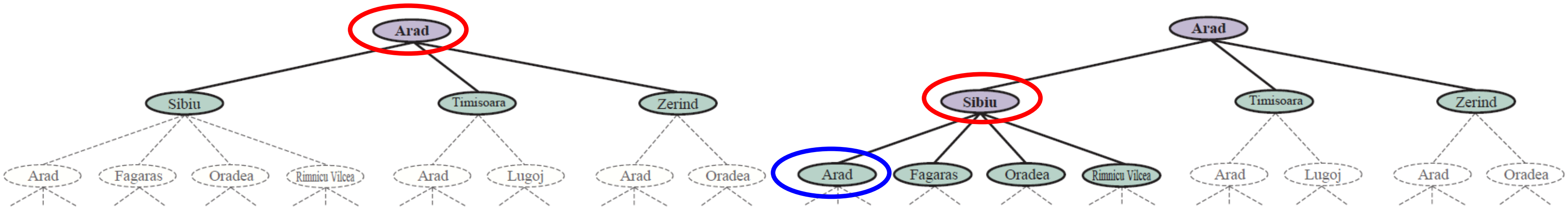


Search tree: the possible action sequences starting from the initial state

Branch: action   Node: state

Tree-search algorithms

function Tree-search(*problem*) **returns** a solution or failure
initialize the frontier using the initial state of *problem*
loop do
if the frontier is empty **then return** failure
choose a leaf node and remove it from the frontier
if the node contains a goal state, **return** the corresponding solution
expand the chosen node, adding the resulting nodes to the frontier



The chosen node: Arad

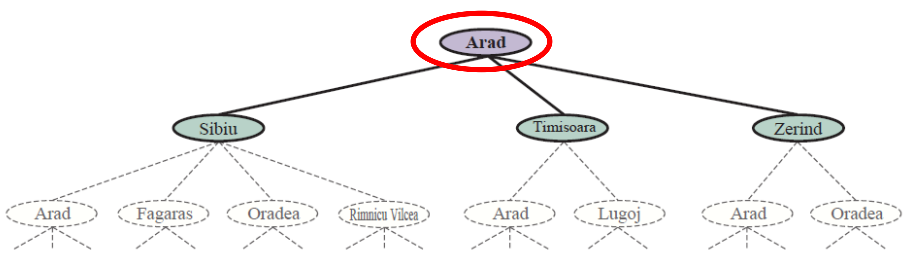
Frontier: Sibiu, Timisoara, Zerind

The chosen node: Sibiu

Frontier: Arad, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind

Graph-search algorithms

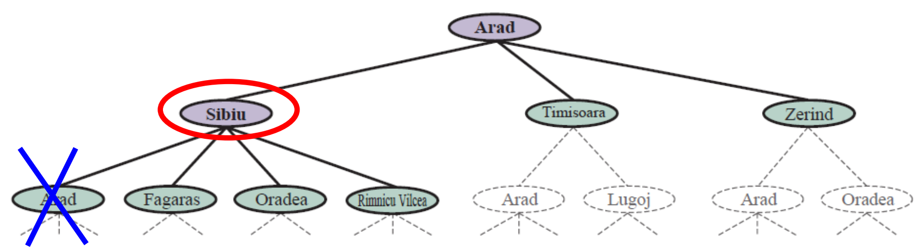
function Graph-search(*problem*) **returns** a solution or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state, **return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set



The chosen node: Arad

Explored set: Arad

Frontier: Sibiu, Timisoara, Zerind



The chosen node: Sibiu

Explored set: Arad, Sibiu

Frontier: Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind

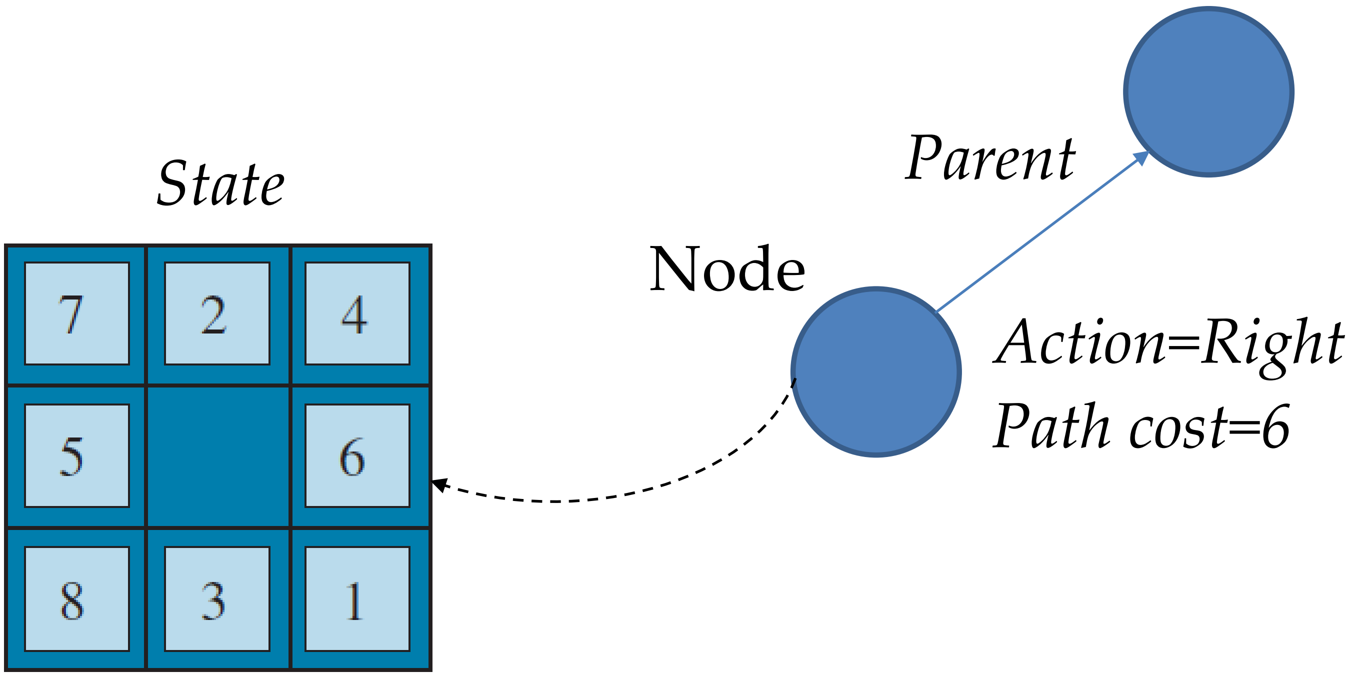
Search algorithms

- Different search algorithms: how to choose a node from the frontier for expansion
  - ✓ Breadth-first search: expand the shallowest node
  - ✓ Depth-first search: expand the deepest node

- Each search algorithm has two implementations
  - ✓ Tree-search
  - ✓ Graph-search

Some notes on implementation

- Data structure of a node of the search tree



- The frontier and explored set can be implemented with a queue and a hash table, respectively

Performance evaluation criteria

A search algorithm's performance can be evaluated in four ways:

- Completeness

  Is the algorithm guaranteed to find a solution when there is one?

- Optimality

  Is the solution found by the algorithm optimal?

- Time complexity

  How long does the algorithm find a solution?

  measured by the number of nodes generated during the search

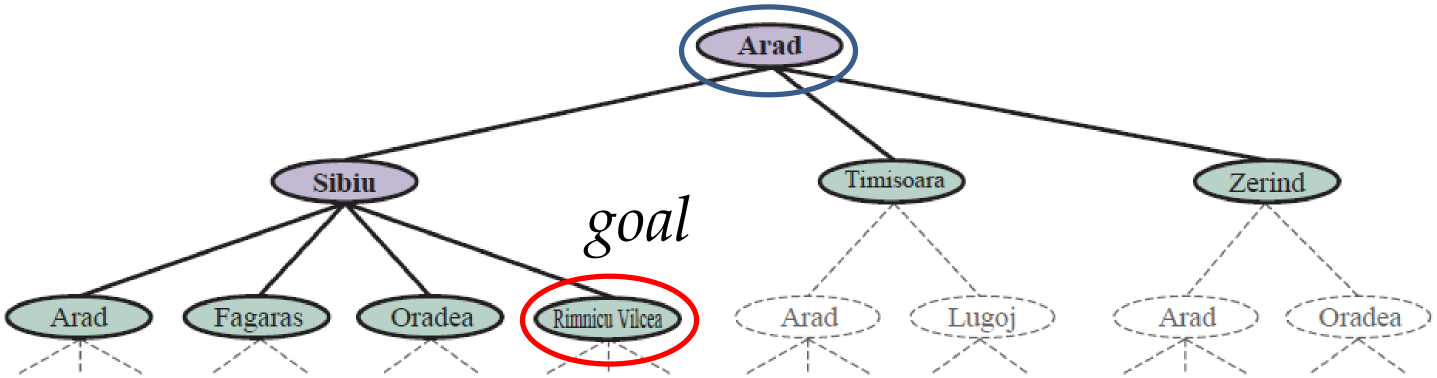
- Space complexity

  How much memory is needed until finding a solution?

  measured by the maximum number of nodes stored in memory

Performance evaluation criteria

- Time and space complexity are usually characterized by three quantities:
  - ✓ The branching factor b, i.e., the maximum number of successors of any node
  - ✓ The depth d of the shallowest goal node
  - ✓ The maximum length m of any path



$b = 4$

$d = 2$

Asymptotic notations

- Let f and g be two positive functions defined on integers, i.e., $f, g: \mathrm{N} \rightarrow \mathrm{R}^+$

- $f \in O(g)$ if there exist positive constants c and n_0 such that

$$\forall n \geq n_0: f(n) \leq c \cdot g(n)$$

$$\lim_{n\rightarrow\infty} \frac{f(n)}{g(n)} < \infty$$

- $f \in o(g)$ if for any positive constant c, there exists positive constant n_0 such that

$$\forall n \geq n_0: f(n) < c \cdot g(n)$$

$$\lim_{n\rightarrow\infty} \frac{f(n)}{g(n)} = 0$$

Asymptotic notations

- Let f and g be two positive functions defined on integers, i.e., $f, g: \mathrm{N} \rightarrow \mathrm{R}^{+}$
- $f \in \Omega(g)$ if $g \in O(f)$ $$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$
- $f \in \omega(g)$ if $g \in o(f)$ $$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$
- $f \in \Theta(g)$ if $f \in O(g)$ and $f \in \Omega(g)$ $$0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

Asymptotic notations

- Let f and g be two positive functions defined on integers, i.e., $f, g: \mathrm{N} \rightarrow \mathrm{R}^+$

$f \in O(g)$	$f \leq g$
$f \in o(g)$	$f < g$
$f \in \Omega(g)$	$f \geq g$
$f \in \omega(g)$	$f > g$
$f \in \Theta(g)$	$f = g$

Asymptotic notations - example

$$\forall \alpha > 0: \log n \in o(n^{\alpha})$$

$$\lim_{n\to\infty} \frac{\log n}{n^{\alpha}} = \frac{1}{\ln 2} \lim_{n\to\infty} \frac{\ln n}{n^{\alpha}} = \frac{1}{\ln 2} \lim_{n\to\infty} \frac{1}{n \cdot \alpha n^{\alpha-1}} = 0$$

L'Hospital's rule

For any positive integer k, $\forall c > 1: n^k \in o(c^n)$

$$\lim_{n\to\infty} \frac{n^k}{c^n} = \frac{k}{\ln c} \lim_{n\to\infty} \frac{n^{k-1}}{c^n} = \frac{k!}{(\ln c)^k} \lim_{n\to\infty} \frac{1}{c^n} = 0$$

Asymptotic notations - example

$$2^n \in o(n!)$$

$$\lim_{n\to\infty}\frac{n!}{2^n} = \lim_{n\to\infty}\frac{n!}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}\cdot\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n}$$

$$= \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n} = \lim_{n\to\infty}\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n = \infty$$

Stirling's approximation

$$\Downarrow$$

$$n! \in \omega(2^n)$$

Asymptotic notations - properties

- Transitivity

$$f(n) \in O\big(g(n)\big) \land g(n) \in O\big(h(n)\big) \Rightarrow f(n) \in O\big(h(n)\big)$$

- Reflexivity

$$f(n) \in O\big(f(n)\big) \quad f(n) \in \Omega\big(f(n)\big) \quad f(n) \in \Theta\big(f(n)\big)$$

- Order of sum functions

$$O\big(f(n) + g(n)\big) = O(\max\{f(n), g(n)\})$$

Summary

- What is search
- Problem complexity: P, NP, NP-hard, NP-complete
- Tree-search and graph-search
- Performance evaluation criteria
- Asymptotic notations

References

- S. J. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Chapter 3.1-3.3, Third edition.

- T. H. Cormen, et al. Introduction to Algorithms. Chapter 3.1 and 34, Second edition.