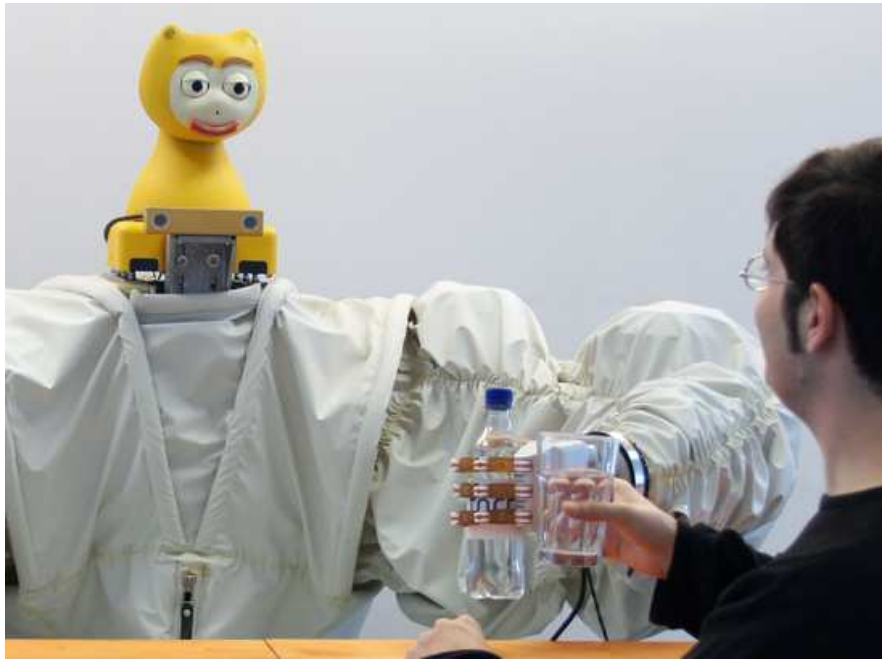
Vorlesung

Grundlagen der Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055



Wintersemester 2012/13 29.10.2012

Chapter 3

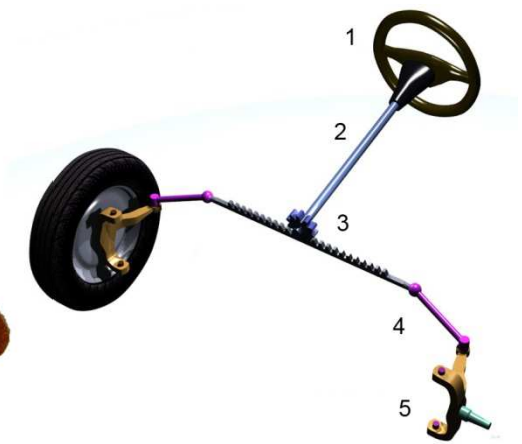
Solving Problems by Searching

What' the problem?

- Assumption: An agent has a goal
  - Described by set of desired states of the world

- How to reach the goal?
- Identify an appropriate formulation of the problem
  - What is the right level of abstraction?

http://upload.wikimedia.org/wikipedia/commons/4/44/Compass_in_a_wooden_frame.png



http://upload.wikimedia.org/wikipedia/commons/b/be/Steer_system.jpg

maps.google.de

Characteristics of the problem

- Observable
  - The agent can determine the current state
- Discrete
  - Only a finite set of possible actions in each state
  - n ways at each crossing
- Environment fully known
  - Knowing the result of each action
  - Next town known for each way
- Deterministic
  - Exactly one result for each action

What‘s the solution?

- A solution to a problem as described is a fixed sequence of actions
  - Final state is the goal state

- After knowing the solution, the action sequence could be carried out
  - Does this always lead to the goal state?

http://wiki.einsatzleiter.net/images/Baum_auf_Stra%C3%9Fe.jpg

Simple problem-solving agent

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*)
returns an action

static: *seq*, an action sequence
state, some description of the current world state
goal, a goal
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)
if seq = *failure* **then** return *NoOp*
action ← FIRST(*seq*)
seq ← REST(*seq*)
return *action*

Formal description of a problem

Components

- Initial state
- Set of possible actions for each state s: ACTIONS(s)
- Transition model RESULT(s,a) defining the successor state
  - This defines the state space
  - Forms a directed graph
  - Path: A sequence of states connected by a sequence of actions
- Goal test, sometimes only properties of a goal state given
  - E.g. checkmate

Formal description of a problem

Measuring effectiveness and efficiency:

- Does the method find a solution at all?
- Is it a good solution (low path cost)?
  - Path cost function (e.g. sum of costs for each action)
  - Measure of quality of a solution
  - Step costs defined by $c(s_i, a, s_j)$

But that‘s not all:

- What is the search cost?
- The total cost = search cost + path cost
  - may not be commensurate!

Formulating problems

Abstraction needed: Real world is absurdly complex

- As few details as possible
  - “Turn right“ instead of “change the angle of the steering wheel by 37.3 deg. within a period of 5 sec.“
- *Valid* abstraction: Each abstract solution can be expanded to one in a more detailed world
- *Useful* abstraction: Execution of an action is more simple than in the original problem formulation

Examples

- Good old-fashioned AI (GOFAI): toy problems, games, theorem proving, etc.
  - Chess, checkers
  - n-queens problem
  - Missionaries and cannibals
  - 8-puzzle
  - Traveling salesman problem (TSP)

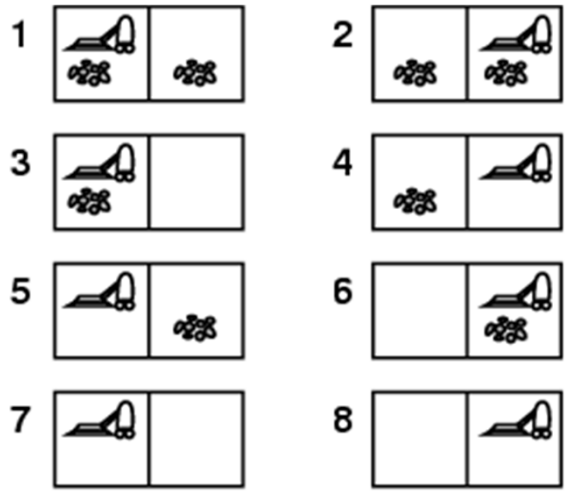
- The techniques can be applied to real-world problems:
  - Route-finding in airline travel planners
  - (real) Travelling Salesman Problem
  - VLSI layout (cell layout and channel routing)
  - “intelligent” manufacturing (assembly sequencing)

Vacuum world: Problem formulation

- States: agent position, state (clean, dirty) of each cell
  - 2 fields only: 2 x 2^2 = 8 states

Possible positions of the agent

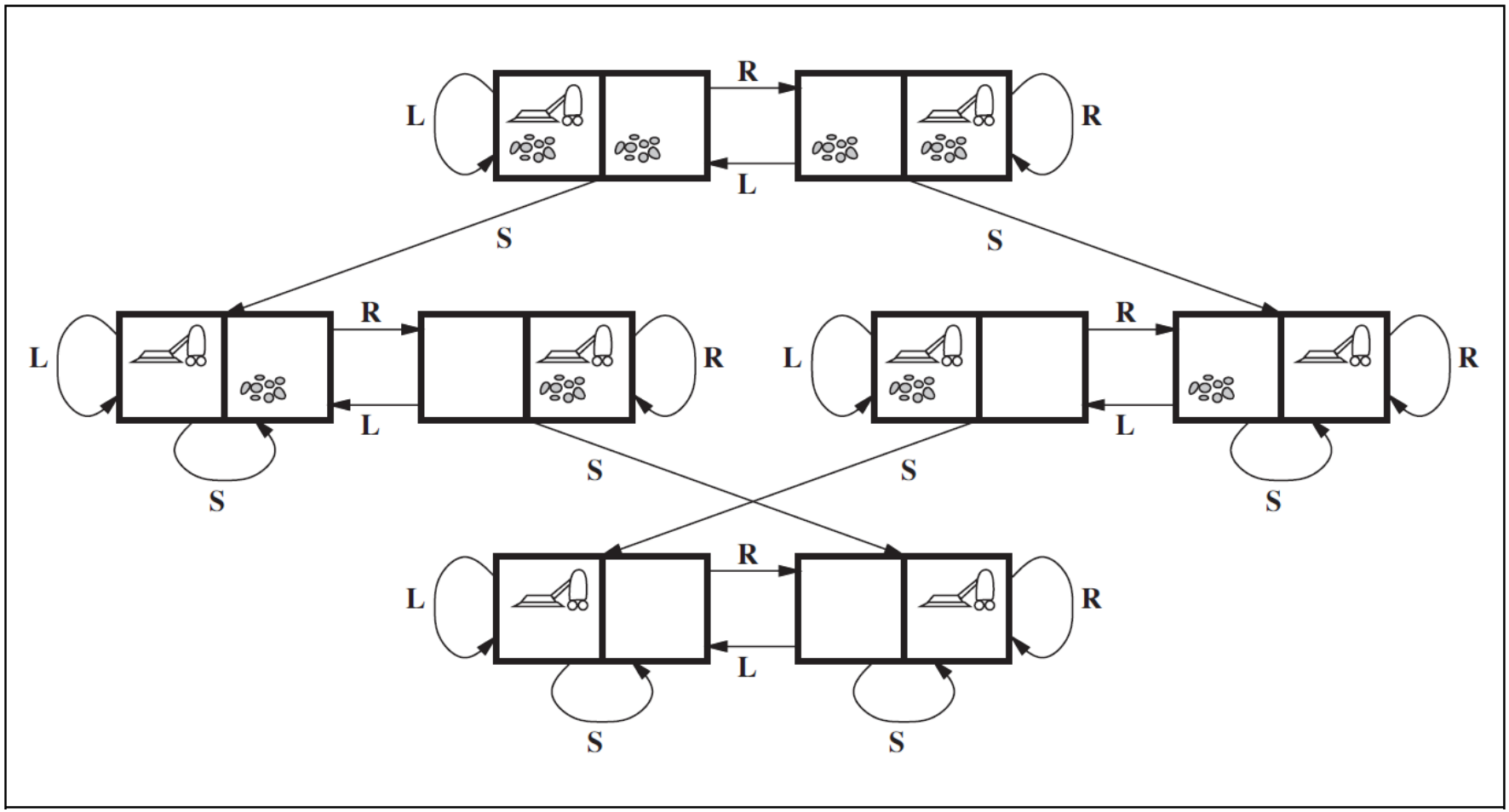
Possible cleaning states of all cells



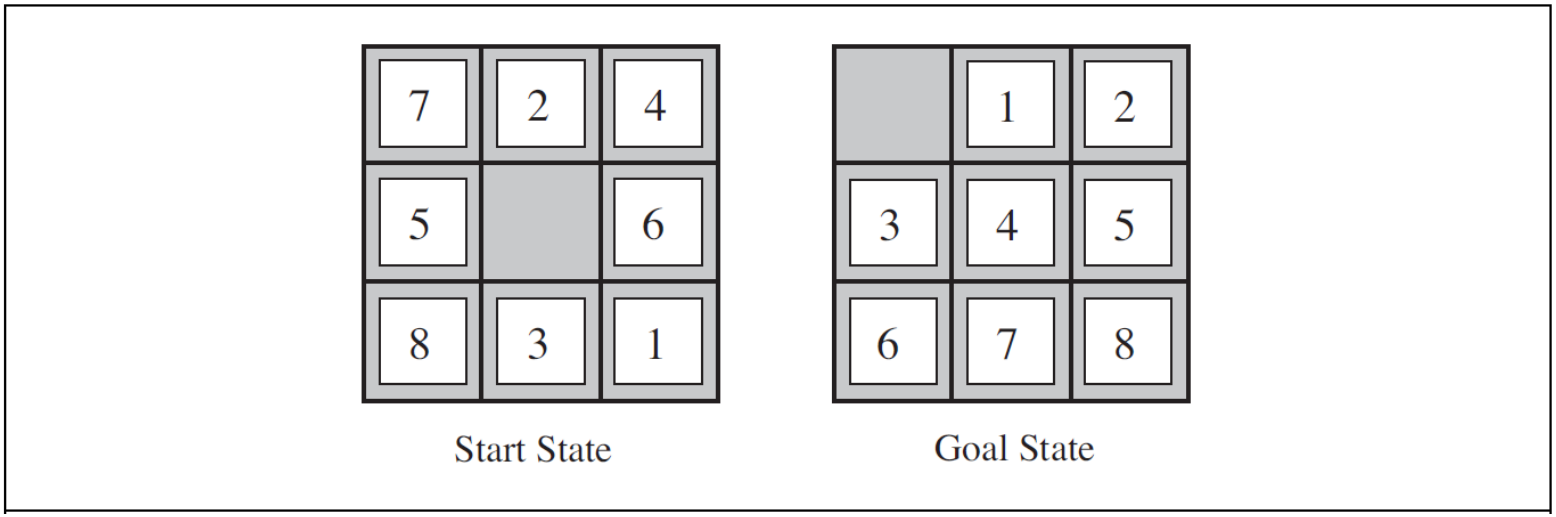
  - n fields: n x 2^n states

- Initial state: can be defined, each field possible
- Actions: left, right, suck
- State transition model shown in graph
- Goal test: all fields clean
- Path cost: each action costs 1 unit

Vacuum world



8-puzzle



Which abstration is valid and useful?

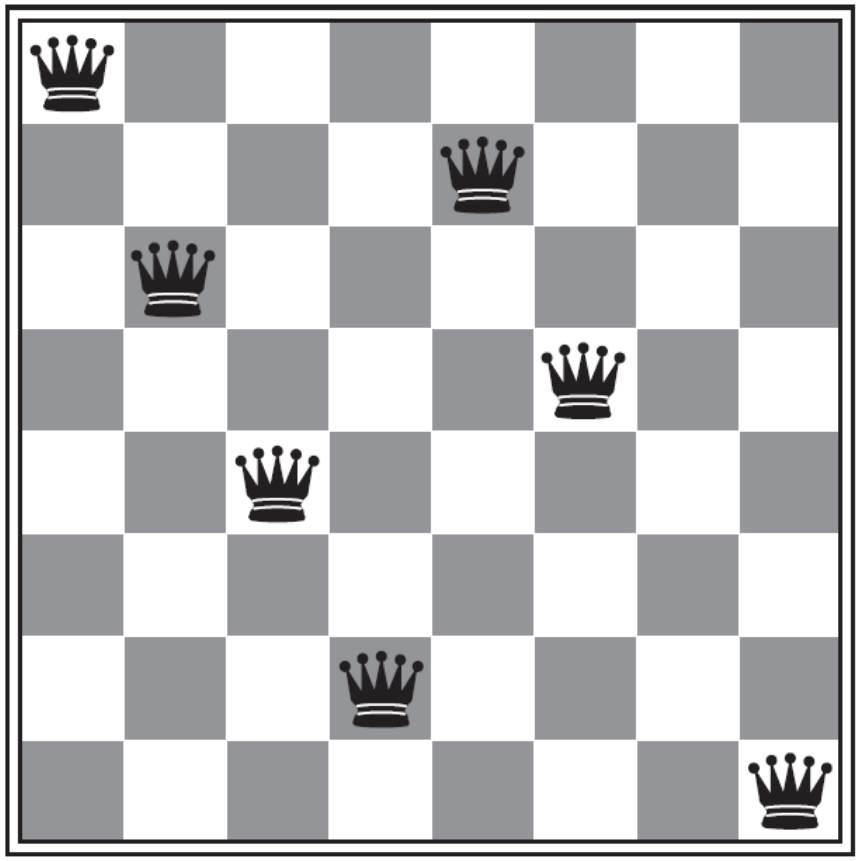
In general: n-puzzle

n	Field size	Number of states
8	3x3	9! / 2 = 181440
15	4x4	1.3e12
24	5x5	1e25

8-puzzle

- States: position of the numbers and the empty field
- Initial state: can be defined
- Actions: easiest formulation:
  Movement of empty field
- State transition model shown in graph
- Goal test: state equal to given goal state?
- Path cost: each action costs 1 unit

Another example: 8 queens



8 queens: Problem formulations

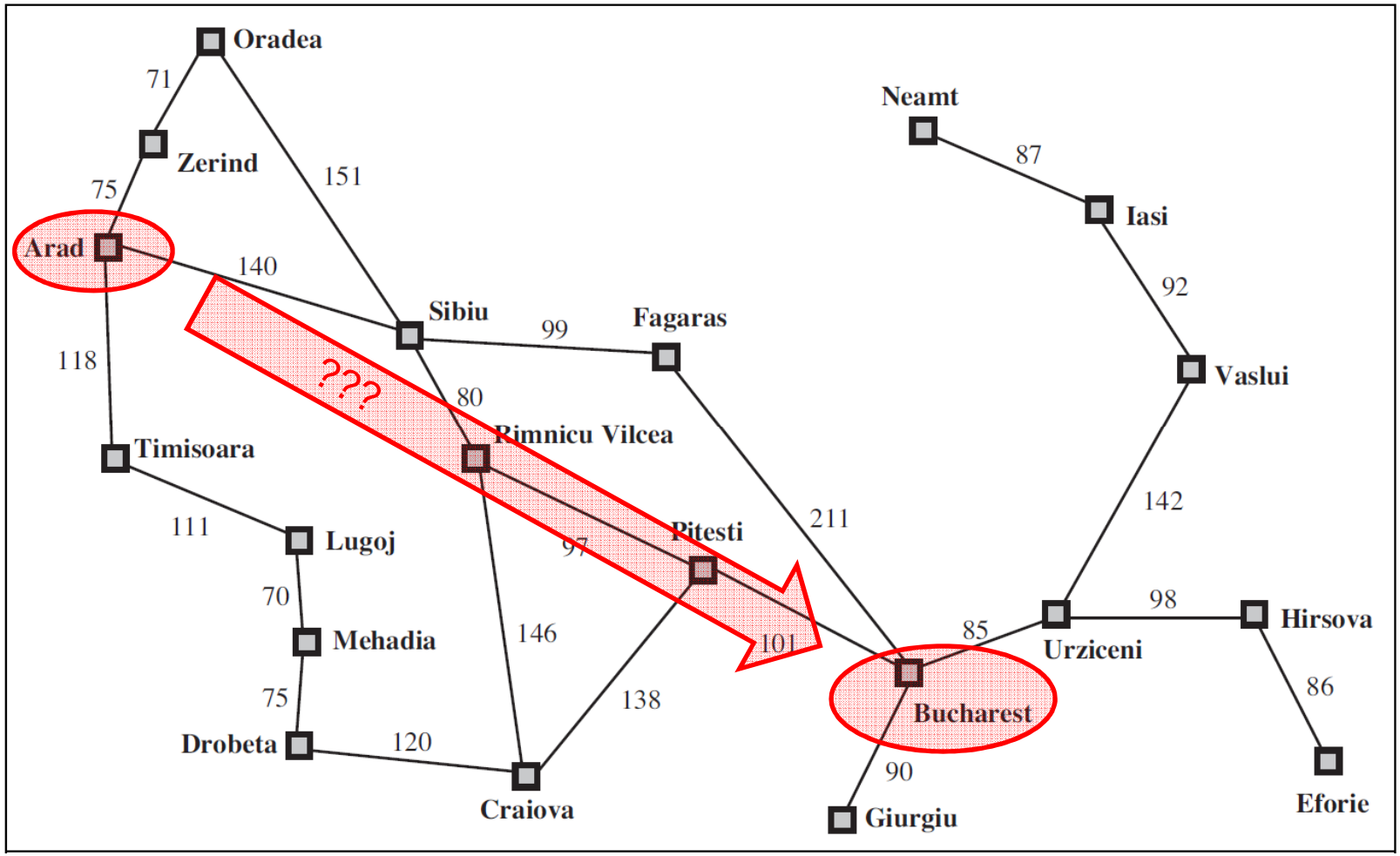
2 possibilities: incremental vs. complete

	Incremental	complete
States	Position of i queens	Position of n queens
Initial state	Empty field	All queens distributed
Action	Add queen	Move queen
Transition	New state, i+1 queens	New state, 8 queens
Goal test	8 queens on the field, none attacked	
Problem size	2057	64x63x62x…x57= 1e14

Good abstraction using incremental view:

- State: i queens, one per column in i most left columns
- Transition: add queen in most left empty column

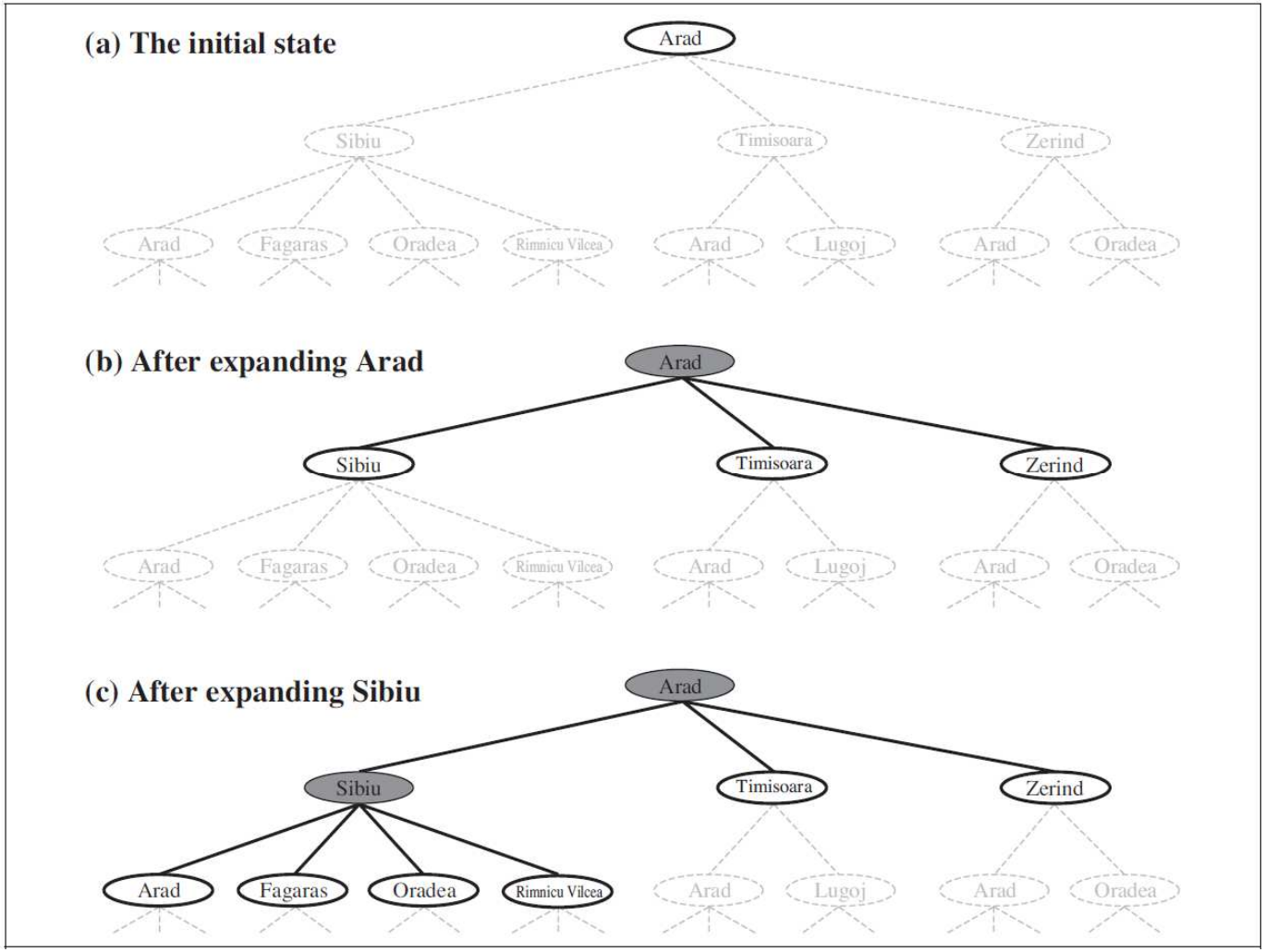
A real-world problem



- How to get from Arad to Bucharest?

How to solve these problems?

- Search trees, nodes represent states



General search algorithm

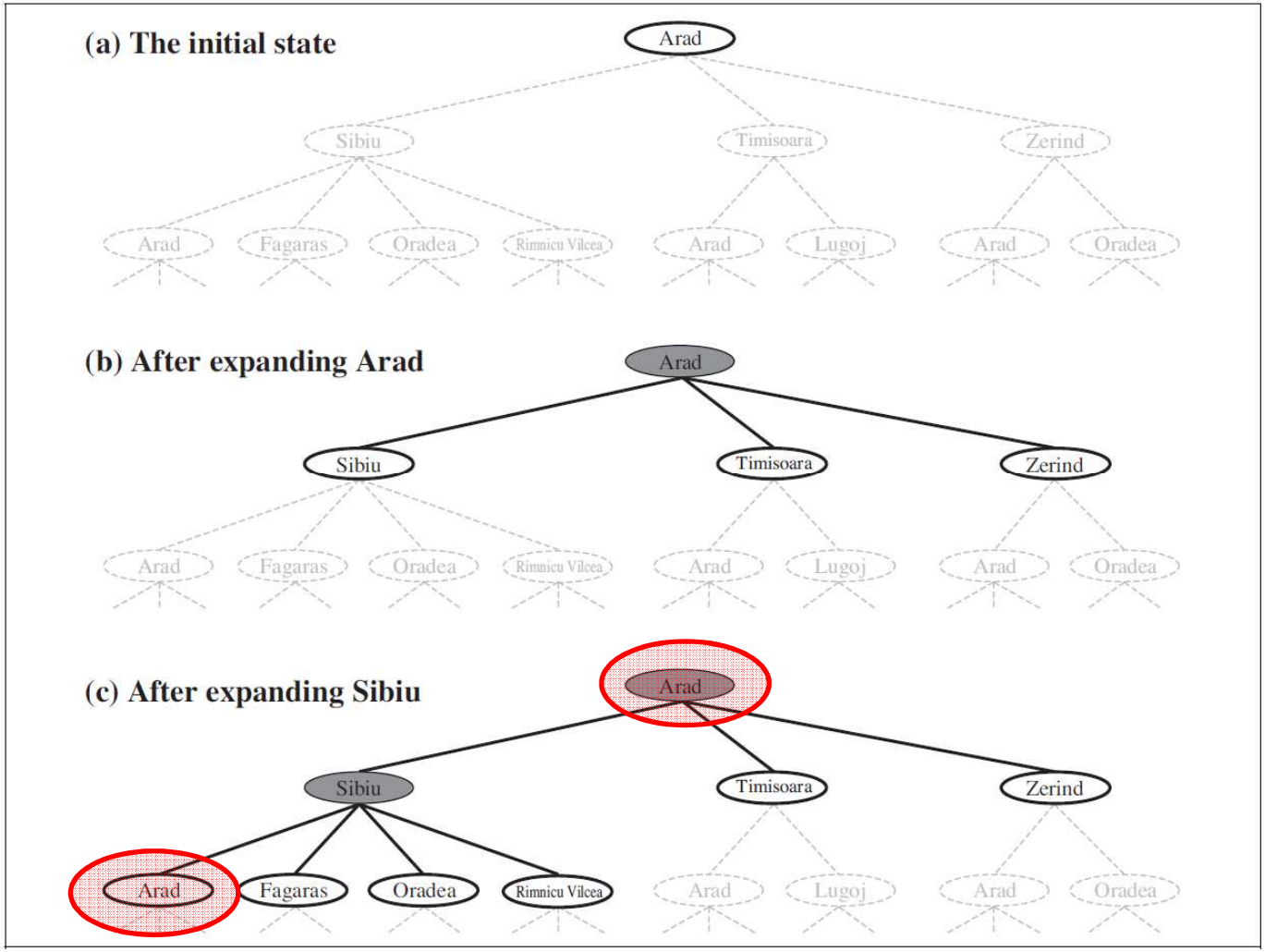
function GENERAL-TREE-SEARCH(*problem*)
returns a solution or an error

static: *open*, the set of frontier nodes, initialized with root node

forever
    if *open* is empty **then return** error
    take a node out of *open*
    if this node contains a goal state **then return** solution
    expand this node (i.e. take all successors)
    add resulting nodes (successors) to *open*

How to solve these problems?

- Search trees, nodes represent states



General search algorithm avoiding loops

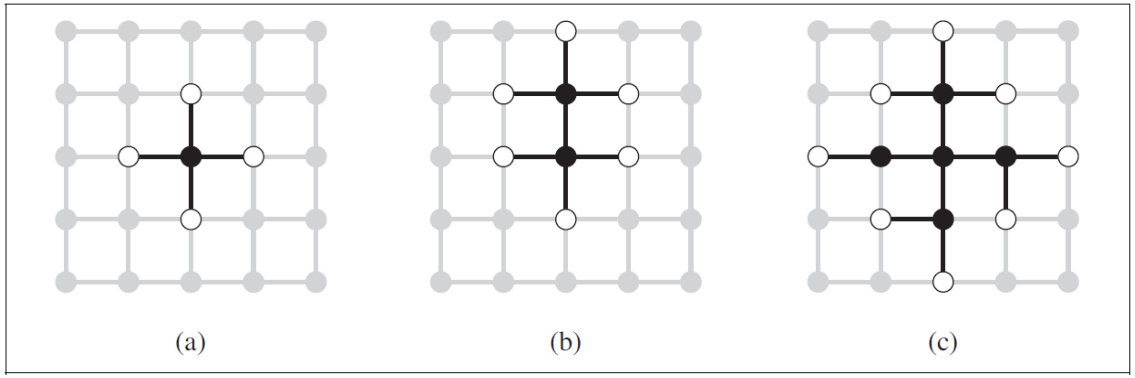
function GENERAL-GRAPH-SEARCH(*problem*)
returns a solution or an error

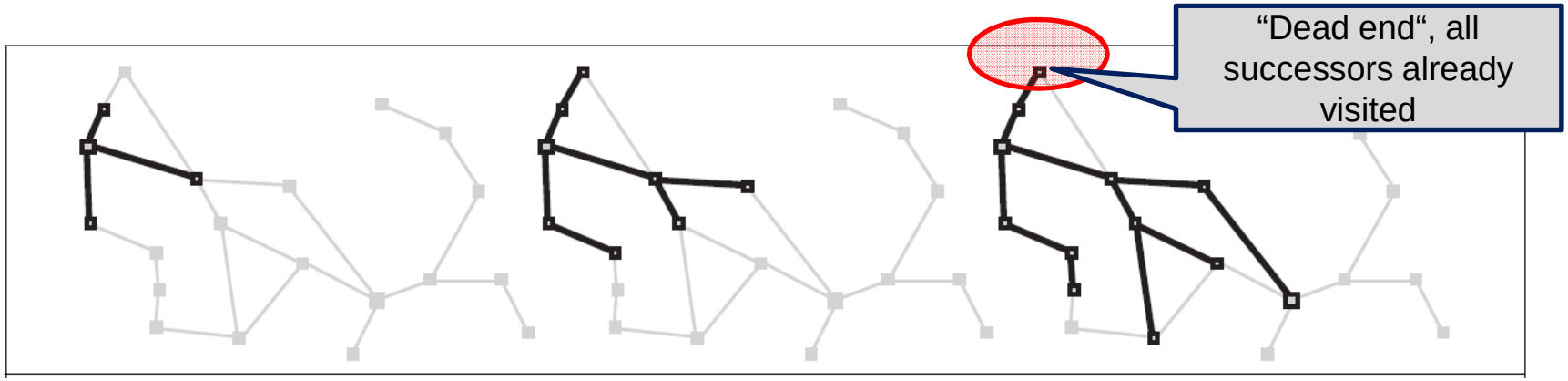
static: *open*, the set of frontier nodes, initialized with root node
closed, the nodes already visited, initally empty set

forever
if *open* is empty **then return** error
take a node out of *open*
add this node to *closed*
if this node contains a goal state **then return** solution
expand this node (i.e. take all successors not in *closed*)
add resulting nodes (successors) to *open*

Graph search

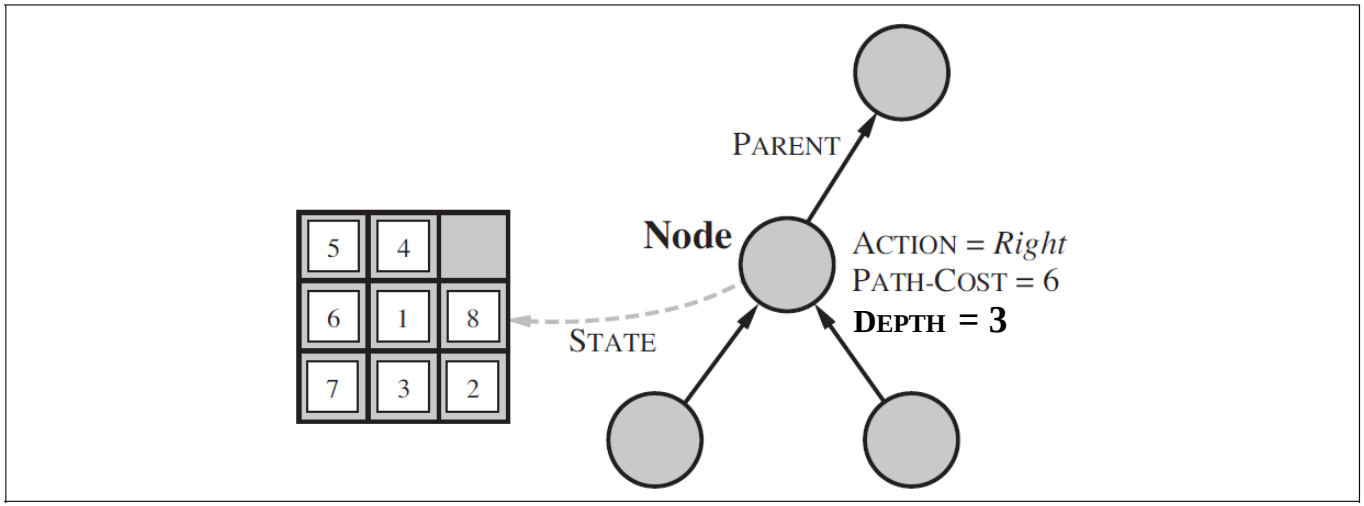
- Each path from initial state to an unexplored state has to cross the border described by *open*. *Open* set forms a separator





Graph search: data structures

- Nodes described by attributes:
  - State
  - Parent
  - Action
  - Path-cost
  - Depth



- Nodes can be stored in queues (FIFO, LIFO, prioritized)
  Operators:
  - EMPTY?(*queue*)
  - POP(*queue*)
  - INSERT(*element*, *queue*)

Criteria for choosing an algorithm

- Completeness: Does the algorithms find a solution ifo one exists?
- Optimality: Does the algorithm find the optimal solution (lowest path cost)?
- Time complexity: How long does it take to find a solution?
- Space complexity: How much memory is needed?

- Complexity in theoretical CS described for |V|+|E|
  - Number of edges plus number of vertices
- Complexity of infinite search spaces?
  - Often the case in AI

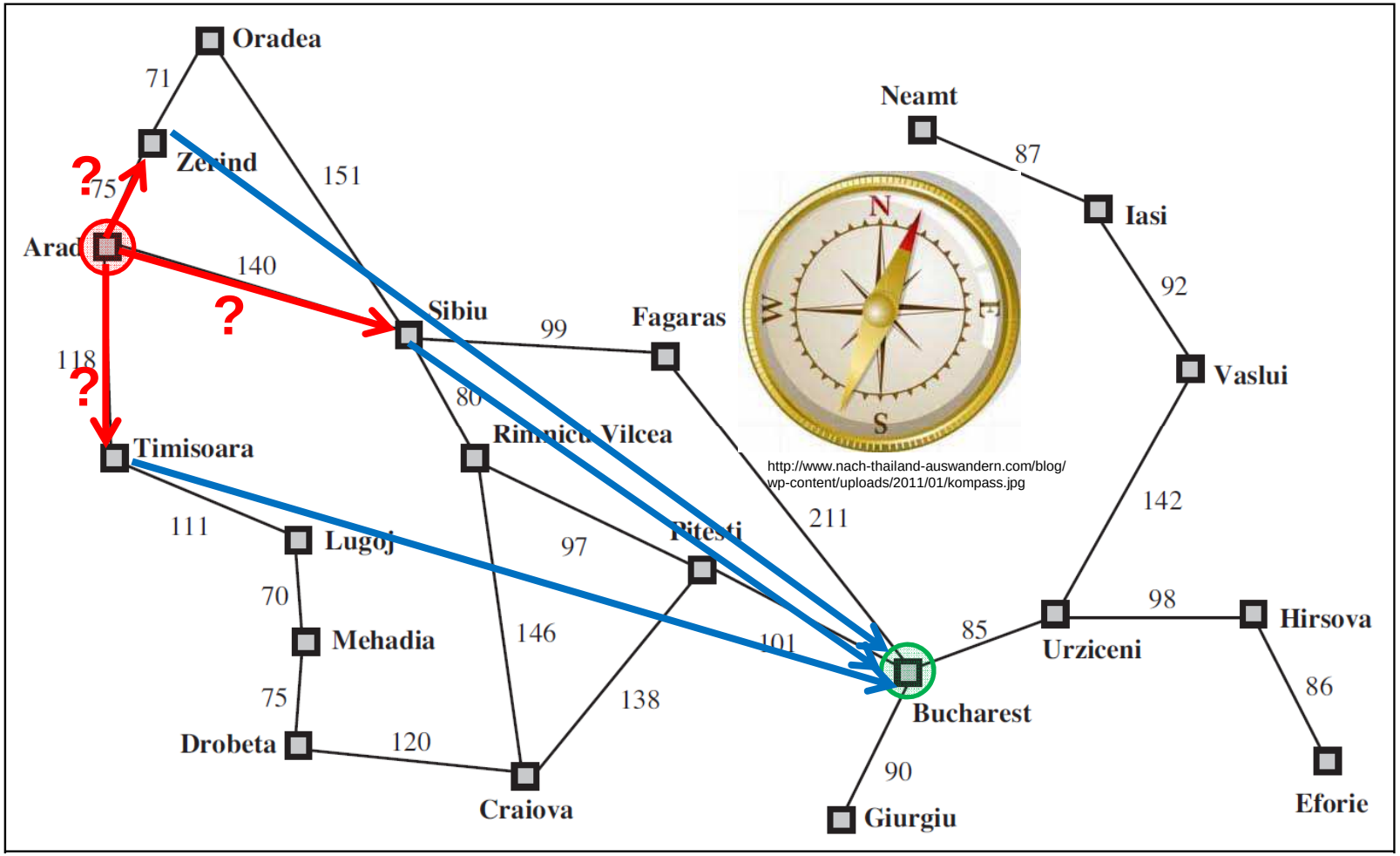
Criteria for choosing an algorithm

- Complexity of AI problems often described by 3 numbers
  - b: branching factor, i.e. max. or avg. number of successors
  - d: depth, i.e. number of steps from the root to the “lowest“ leaf
  - m: maximum length of any path in the search space

- Consideration of seach costs or the total with action costs

Search strategies

- Uninformed vs. informed search



Uninformed or blind search

- Another look at the graph-search algorithm

function GENERAL-GRAPH-SEARCH(*problem*)
returns a solution or an error

static: *open*, the initial state (set of nodes)
closed, the nodes already visited, initally empty set

forever
if *open* is empty **then return** error
take a node out of *open*
add this node to *closed*
if this node contains a goal state **then return** solution
expand this node (i.e. take all successors not in *closed*)
add resulting nodes (successors) to *open*

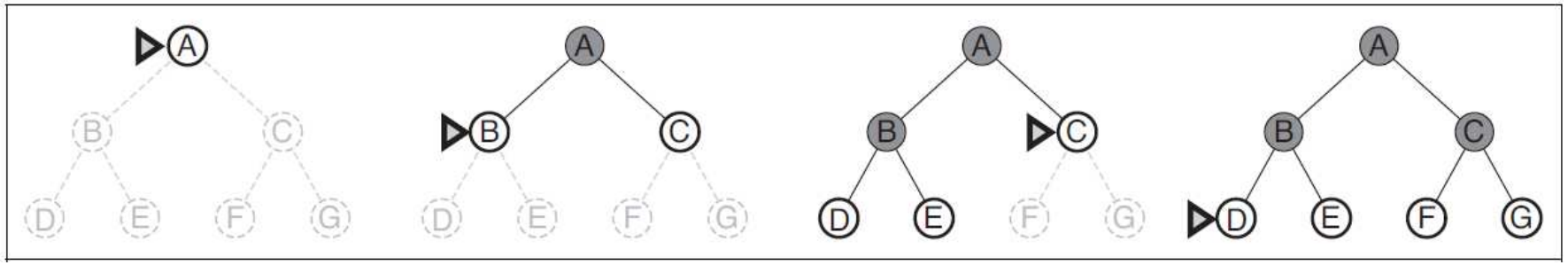
- Degree of freedom: way of adding successor nodes

Uninformed or blind search

- No information about length or costs of a path
  - Breadth-first
  - Depth-first
  - Uniform-cost
  - Depth-limited
  - Iterative deepening
  - Bi-directional

Breadth-first search

- Add nodes at the end of the *open* queue
- Search Pattern: “spread before dive”



open

(A) *(B,C)* *(C,D,E)* *(D,E,F,G)*

initial state

Optimization: test for goal state already when node created

Breadth-first search

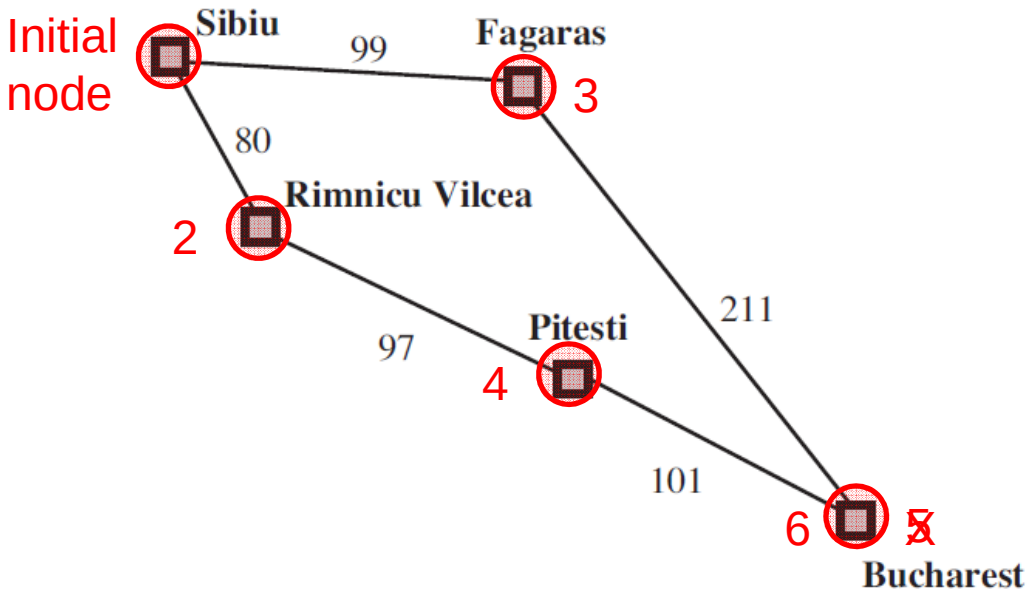
- Completeness?
  - Yes, if deepest node at finite depth d
- Optimal?
  - Yes, if uniform step costs
- Time complexity:
  - $b+b^2+b^3+ \ldots+b^d = O(b^d)$
- Space complexity
  - $O(b^d)$

- What if no uniform step costs?

Uniform-cost search

- Idea: store new nodes in priority queue

(Sibiu-0)
(Riminiu Vilcea-80, Fagaras-99
(Fagaras-99, Pitesti-177)
(Pitesti-177, Bucharest-310)
(Bucharest-278)

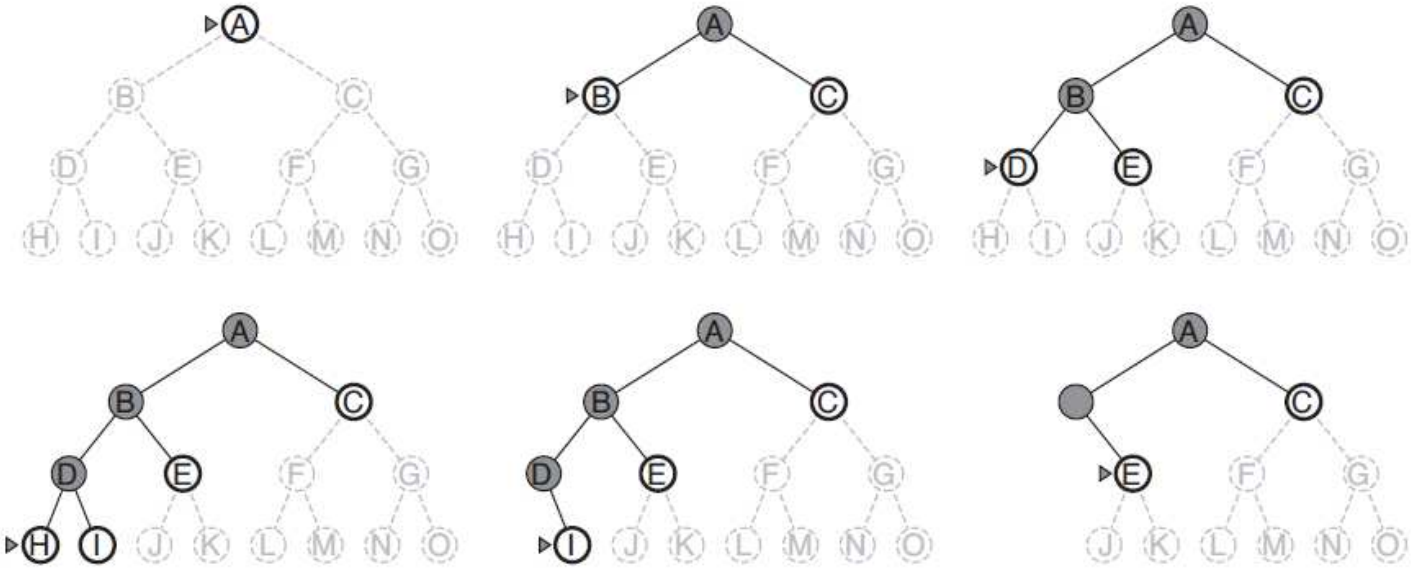


Uniform-cost search

- Completeness?
  - Yes, if b finite and step costs $c >= \varepsilon > 0$ for all actions
- Optimal?
  - Yes
- Time complexity:
  - $O(b^{1+C^*/\varepsilon})$, with C^*: cost of the optimal solution,
    $\varepsilon > 0$: min. action cost
- Space complexity
  - $O(b^{1+C^*/\varepsilon})$

Depth-first search

- Add nodes at the front of the *open* queue
- Search Pattern: “dive before spread”



(A)

(H,I,E,C)

(B,C)

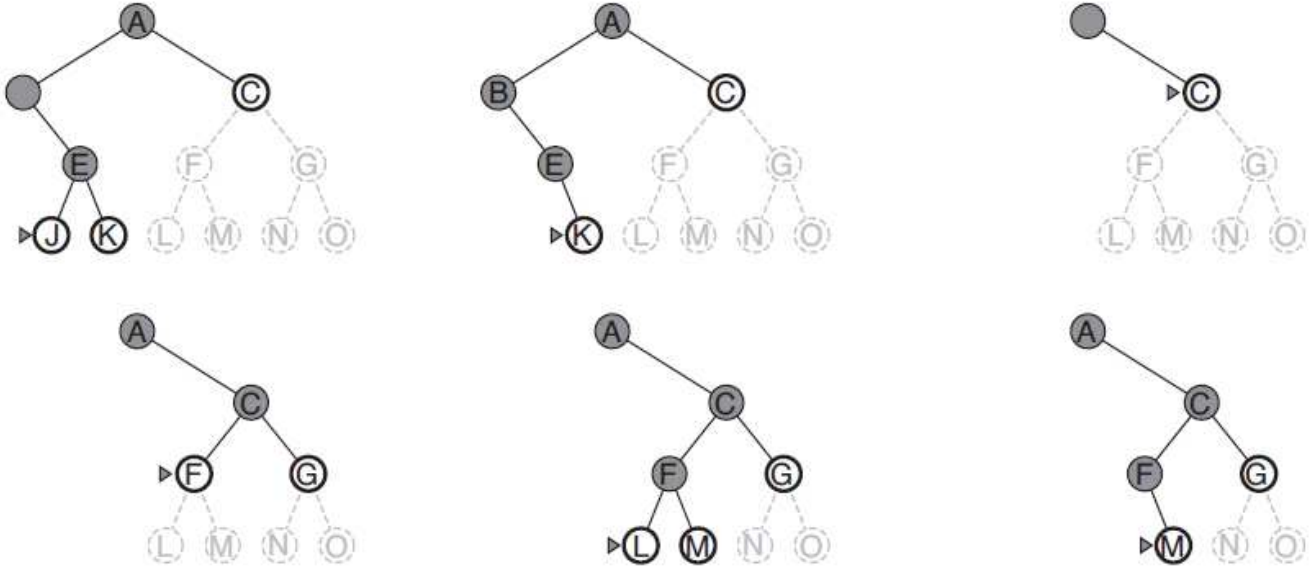
(I,E,C)

(D,E,C)

(E,C)

Depth-first search

- Add nodes at the front of the *open* queue



open

(J,K,C)	*(K,C)*	*(C)*
(F,G)	*(L,M,G)*	*(M,G)*

Depth-first search

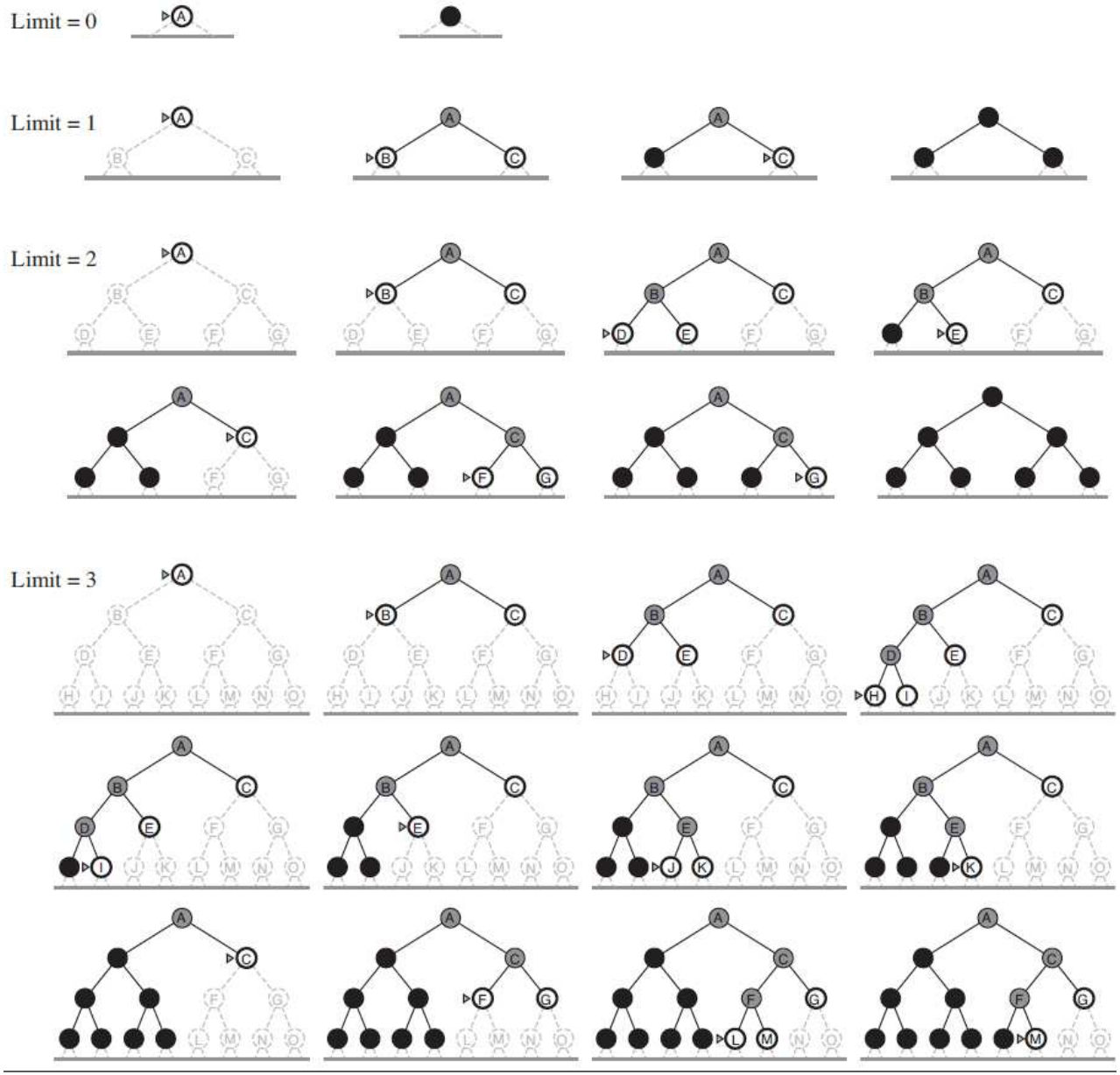
- Completeness?
  - No
- Optimal?
  - No
- Time complexity:
  - $O(b^m)$
- Space complexity
  - O(bm)

Depth-limited search

- Treat nodes with depth >= d_{max} as if no sucessors
- $d_{max} \rightarrow \infty$ leads to dept-first search

- Completeness?
  - No
- Optimal?
  - No
- Time complexity:
  - $O(b^{d_{max}})$
- Space complexity
  - $O(b\ d_{max})$

Iterative deepening depth-frist search

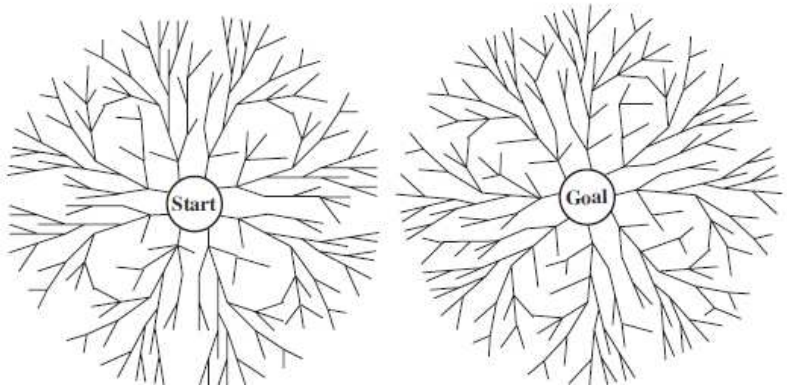


Iterative deepening depth-frist search

- Repetition of search for the upper levels, can be ignored
- Prefereed choice for large search spaces and unknown depth

- Completeness?
  - Yes, if b finite
- Optimal?
  - Yes, if uniform step costs
- Time complexity:
  - $O(b^d)$
- Space complexity
  - O(b d)

Bi-directional search



How to check for solution?
Test for non-empty intersection set of *open*

- Completeness?
  - Yes, if b finite and breadth-first in both directions
- Optimal?
  - Yes, if uniform step costs and breadth-first in both directions
- Time complexity:
  - $O(b^{d/2})$, much better than breadth-first because $2x\ b^{d/2} << b^d$
- Space complexity
  - $O(b^{d/2})$

Bi-directional search

Problems:

- Operators are not always or only very diffcultly invertible (computation of parent nodes)
- In come cases there exist many goal states, which are described only partially. Example: predecessor state of “checkmate".
- One needs effcient procedures in order to test whether the search procedures have met
- Which search method should one use for each direction?

Summary

- In order to search for a solution, an agent has to define its goal and based on this the agent has to defineits problem
- A problem consists of 5 parts: state space, initial state, operators, goal test and path costs. A path from the initial state to a goal state is a solution.
- There exists a general search algorithm that can be used to find solutions. Special variants of the algorithm make use of different search strategies.

Summary

- Search algorithms are evaluated on the basis of the following criteria:
- completeness, optimality, time- and space complexity.

Criterion	Breadth-First	Uniform Cost	Depth-First	Depth-Limited	Iterative Deep.	Bidirec.
Complete	Yes	Yes	No	No	Yes	Yes
Time	$O(b^d)$	$O(b^{1+C^*/\varepsilon})$	$O(b^m)$	$O(b^d{}_{max})$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+C^*/\varepsilon})$	$O(bm)$	$O(b\ d_{max})$	$O(bd)$	$O(b^{d/2})$
Optimal	Yes	Yes	No	No	Yes	Yes