

BME-16 Artificial Intelligence and Medical Decision Support Systems

Informed and Uninformed Intelligent Search

Michail G. Lagoudakis

Intelligent Systems Laboratory

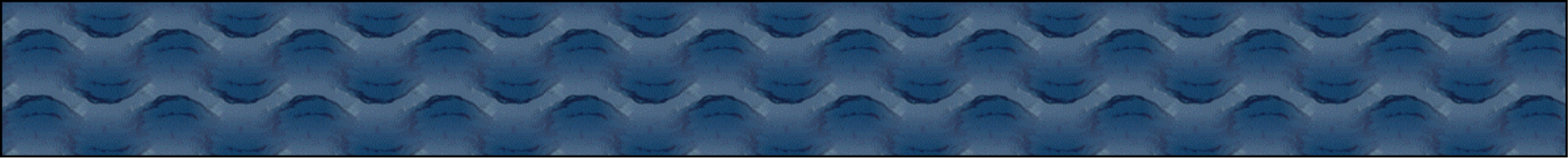
School of Electrical and Computer Engineering

Technical University of Crete

Chania, Crete, Greece

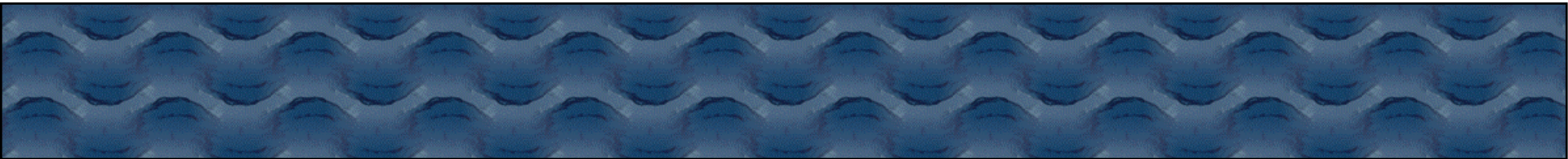


intelligence
INTELLIGENT SYSTEMS LABORATORY

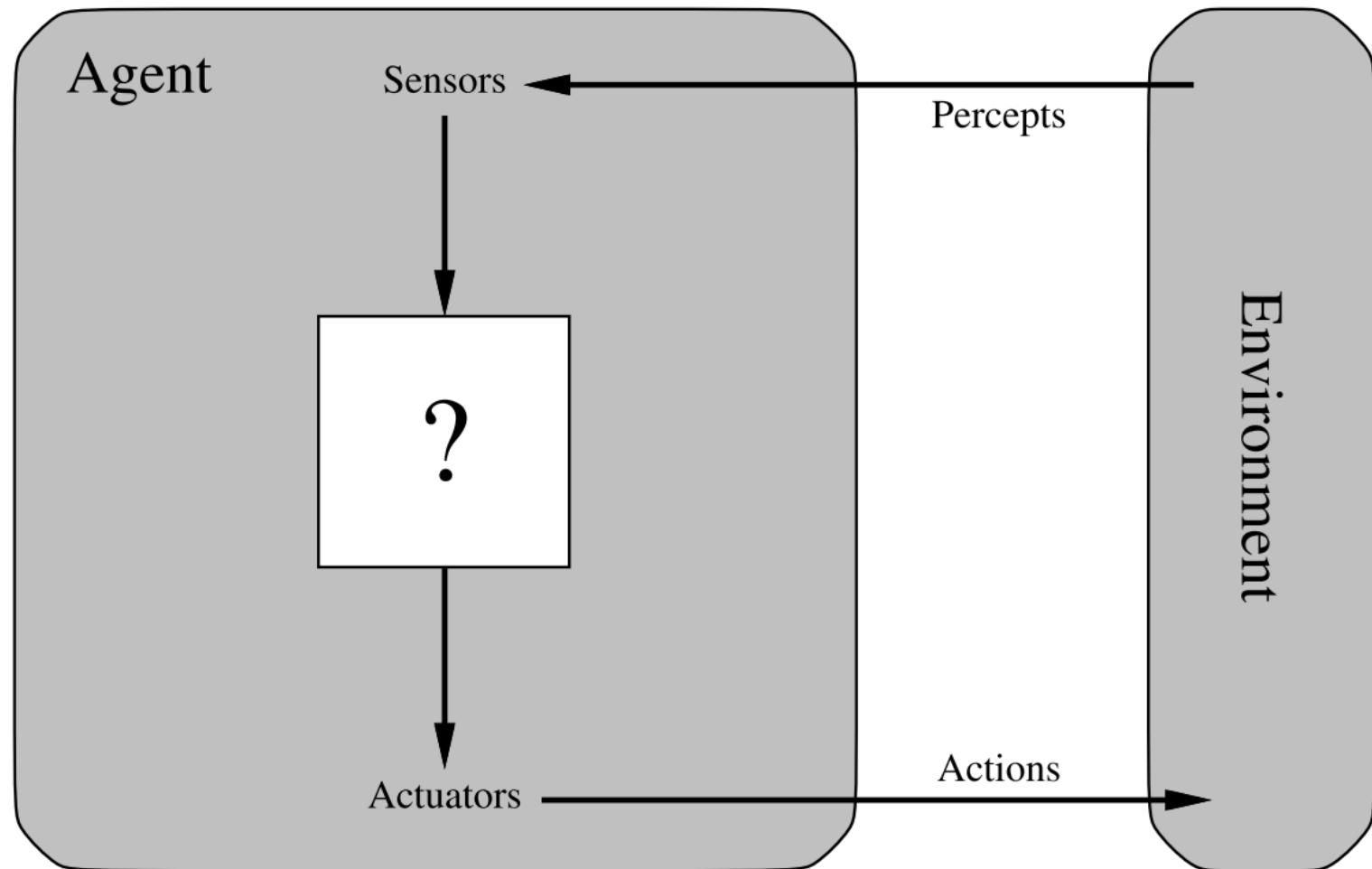


Agents and Environments

a unified view



Agent and Environment



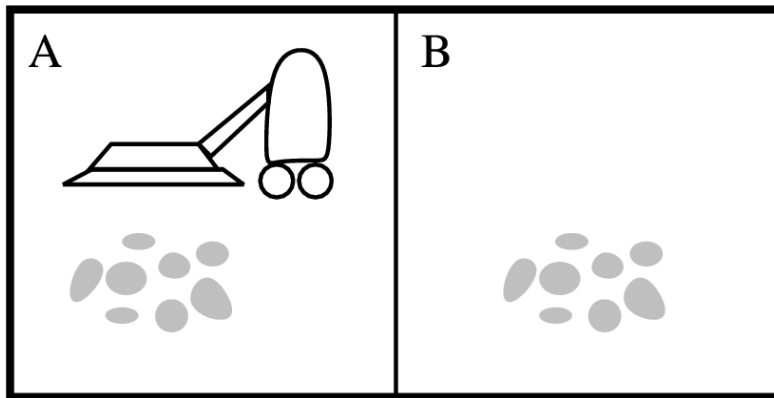
Agents

- **Agent**
 - from the Latin *agere* = act
 - entity with **sensors** and **actuators**
 - **perceives** the environment and **affects** the environment
- **Human agents**
 - sensors: eyes, ears, nose, skin, ...
 - actuators: arms, hands, legs, mouth, ...
- **Robotic agents**
 - sensors: cameras, laser, infrared, encoders, ...
 - actuators: motors, joints, wheels, tracks, ...
- **Software agents**
 - sensors: keyboard, mouse, files, network packages, ...
 - actuators: print on screen, write on disk, ...

Agent Structure

- **Percepts**
 - the information from the sensors at any time point
- **Percept sequence**
 - the full history of percepts temporally ordered
- **Actions**
 - the effects of the actuators at any time point
- **Agent Function**
 - mapping from percept sequences to actions
 - definition: a huge look-up table
 - implementation: a computer program

A Simple Agent



Percepts

Room (A/B), Status (Dusty/Clean)

Actions

Left, Right, Suck

Percept Sequence	Action
[A, Clean]	Right
[A, Dusty]	Suck
[B, Clean]	Left
[B, Dusty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [B, Dusty]	Suck
...	...

Program Vacuum_Cleaner

...

if (Status==Dusty) **then** Suck

if (Room==A) **then** Right

if (Room==B) **then** Left

...

Rational Agents

- **Performance measure**
 - objective definition of successful agent behavior
 - design principle: **what** we want, not **how** it can be done
- **Rational agent**
 - chooses actions that maximize the performance measure
 - uses the percept sequence and the embedded knowledge
- **Rationality**
 - does not mean omniscience or perfection
 - maximization of the *expected* performance

Task Environment

- **Task environment**
 - environment = problem, agent = solution
 - PEAS description
 - Performance measure, Environment, Actuators, Sensors
- **Example: Agent for Internet Shopping**

Measure	Environment	Actuators	Sensors
price quality fitness efficiency	www e-shops comparison stores couriers	searching following links filling forms printing receipts	HTML pages product codes price offers payment methods

Environment Properties

- **Observability**
 - fully observable
 - partially observable
- **Predictiveness**
 - deterministic
 - stochastic
 - strategic
- **Dependency**
 - episodic
 - sequential

Environment Properties

- **Variability**
 - static
 - dynamic
 - semi-dynamic
- **Description**
 - discrete
 - continuous
- **Multitude**
 - single-agent
 - multi-agent
 - competitive or cooperative

Range of Environments

□ **Hard environments**

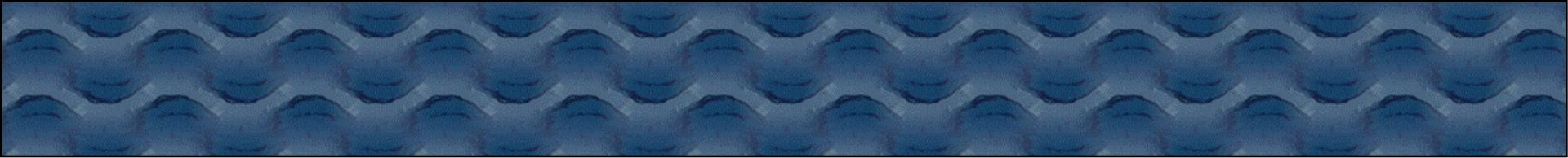
- partially observable
- stochastic
- sequential
- dynamic
- continuous
- multi-agent

□ **Easy environments**

- fully observable
- deterministic
- episodic
- static
- discrete
- single-agent

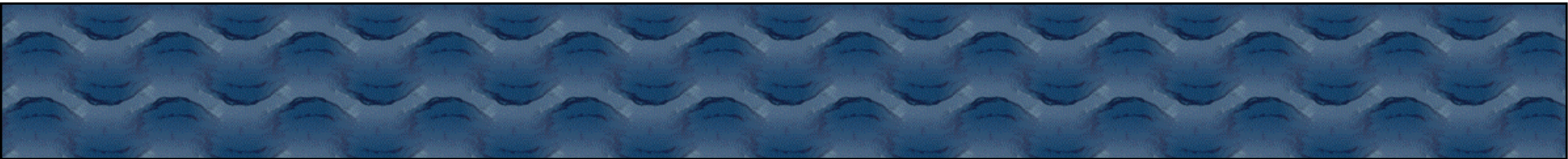
Environment Examples

Crossword Puzzles	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a timer	Fully	Strategic	Sequential	Semi Dynamic	Discrete	Multi
Poker	Partially	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi Driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical Diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image Analysis	Fully	Deterministic	Episodic	Semi Dynamic	Continuous	Single
Robotic Part Picker	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Plant Controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English Tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi



Problem-Solving Agents

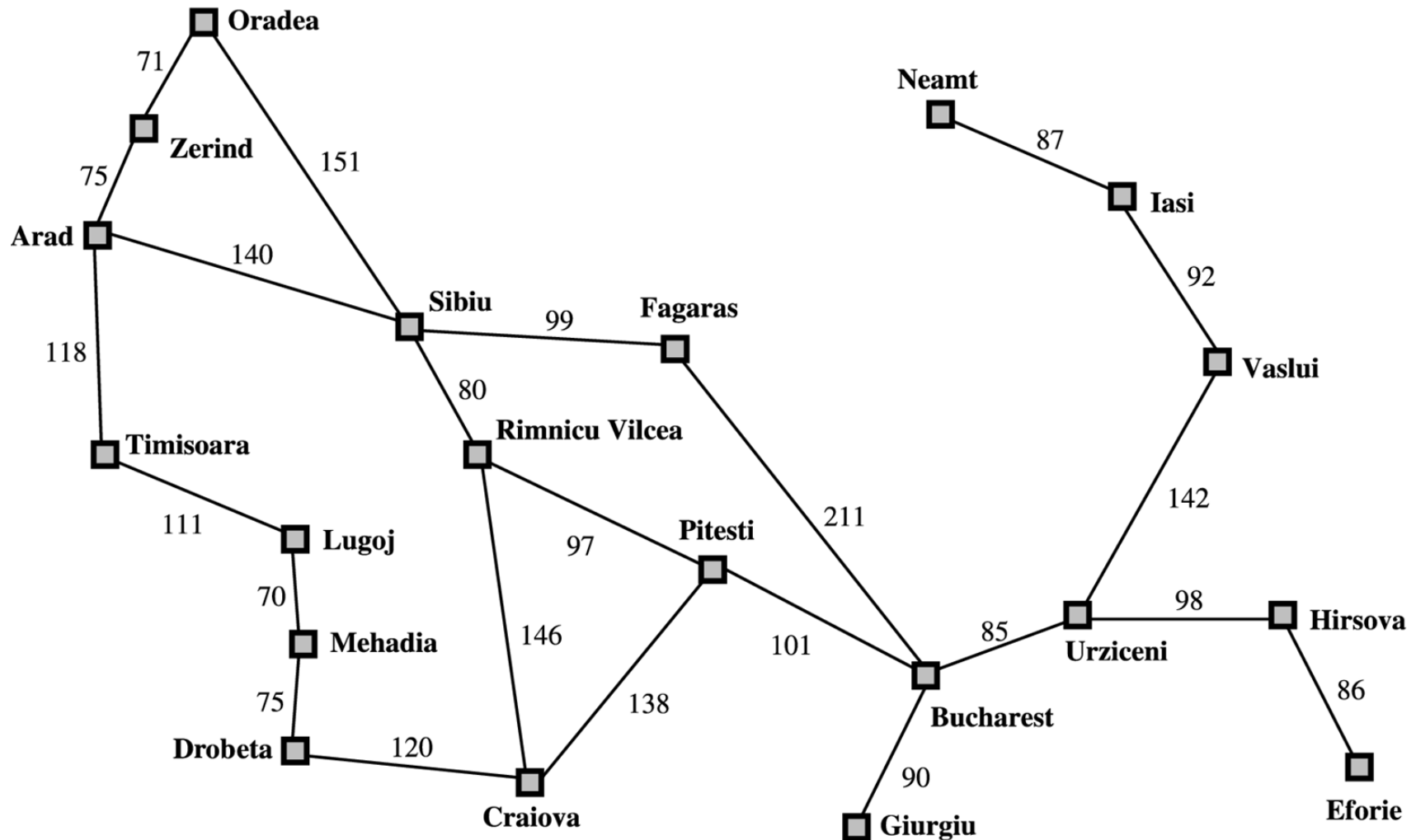
useful agents



Problem-Solving Agents

- **Problem solving**
 - the process of finding a solution for some problem
- **Problem formulation**
 - definition of states and actions
- **Goal formulation**
 - definition of a subset of states considered as solutions
- **Action selection**
 - action sequences (as opposed to instant actions)
- **Search**
 - the process of finding an appropriate action sequence that leads to a solution

Example: Routes in Romania



Problem-Solving Agent Code

function Simple-Problem-Solving-Agent(*percept*) **returns** *action*

inputs: *percept*, some percept

static: *sequence*, an action sequence, initially empty
state, description of the current state of the world
goal, some goal, initially empty
problem, a formulation of some problem

state $\leftarrow$ Update-State(*state*, *percept*)

if *sequence* is empty **then do**

goal $\leftarrow$ Formulate-Goal(*state*)

problem $\leftarrow$ Formulate-Problem(*state*, *goal*)

sequence $\leftarrow$ Search(*problem*)

action $\leftarrow$ First(*sequence*)

sequence $\leftarrow$ Rest(*sequence*)

return *action*

Conventions

- **Environment**

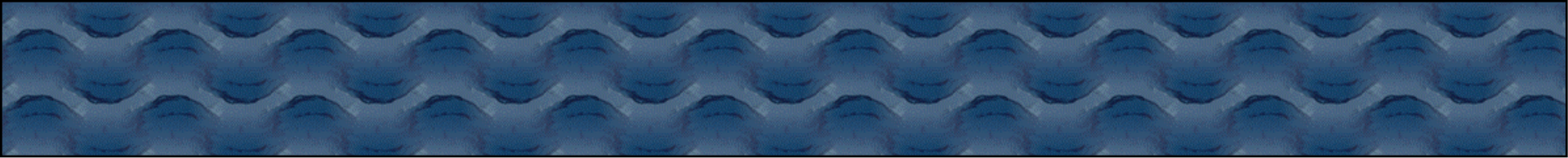
- static (no changes during the search)
- observable (known initial state)
- discrete (finite number of actions)
- deterministic (predictable transitions)

- **Agent**

- open-loop control
- action execution without perception (sensor-less)

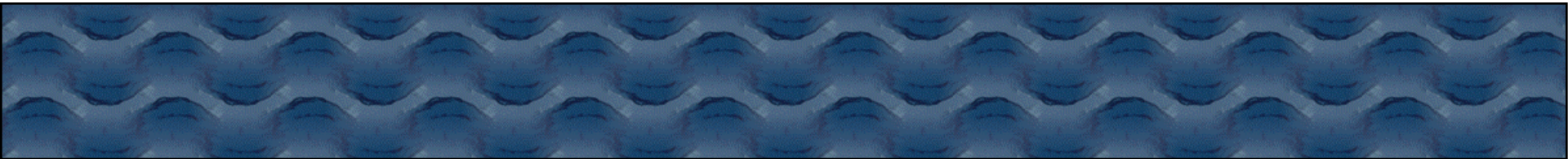
- **Abstraction**

- elimination of unnecessary details from the representation



Search

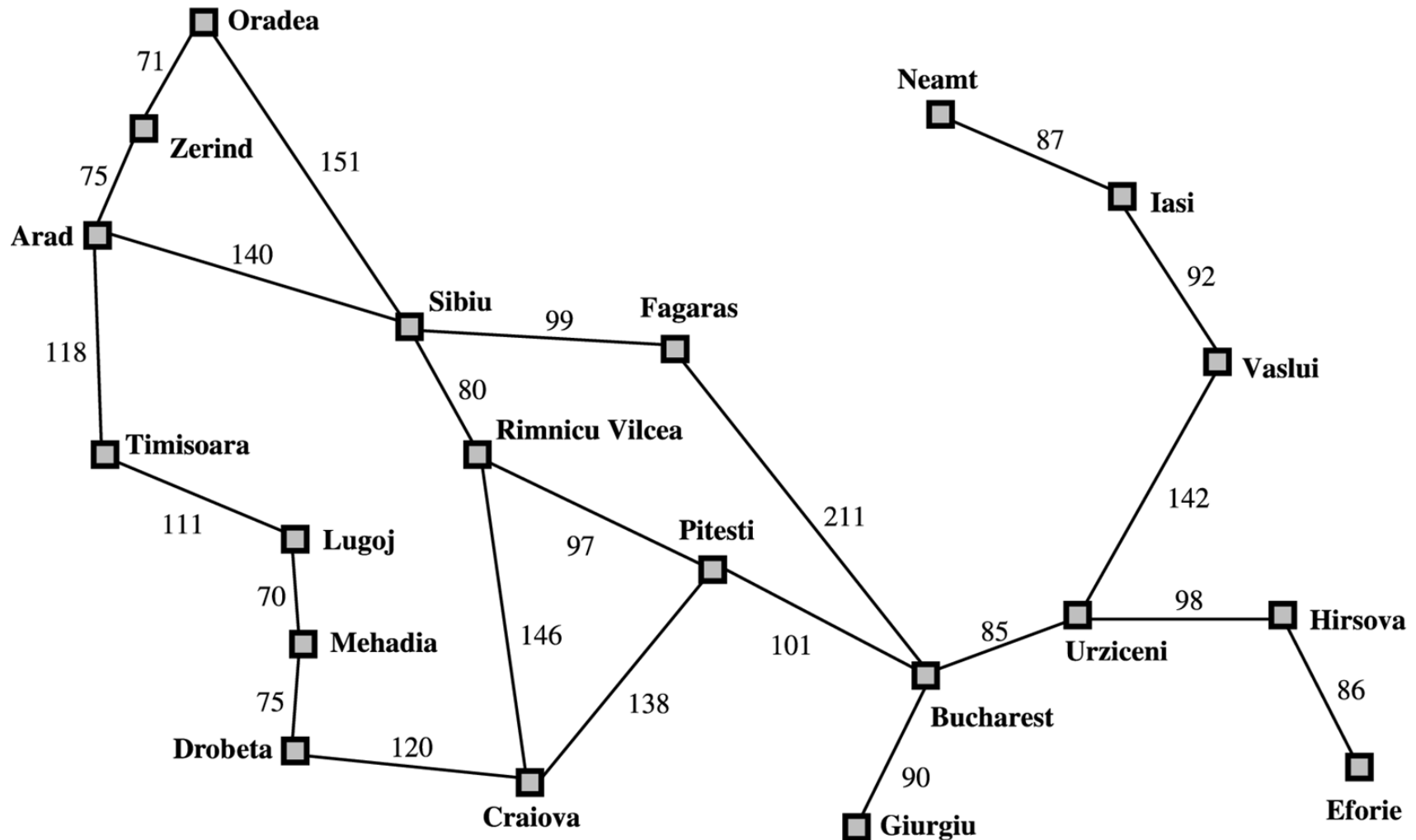
looking for something



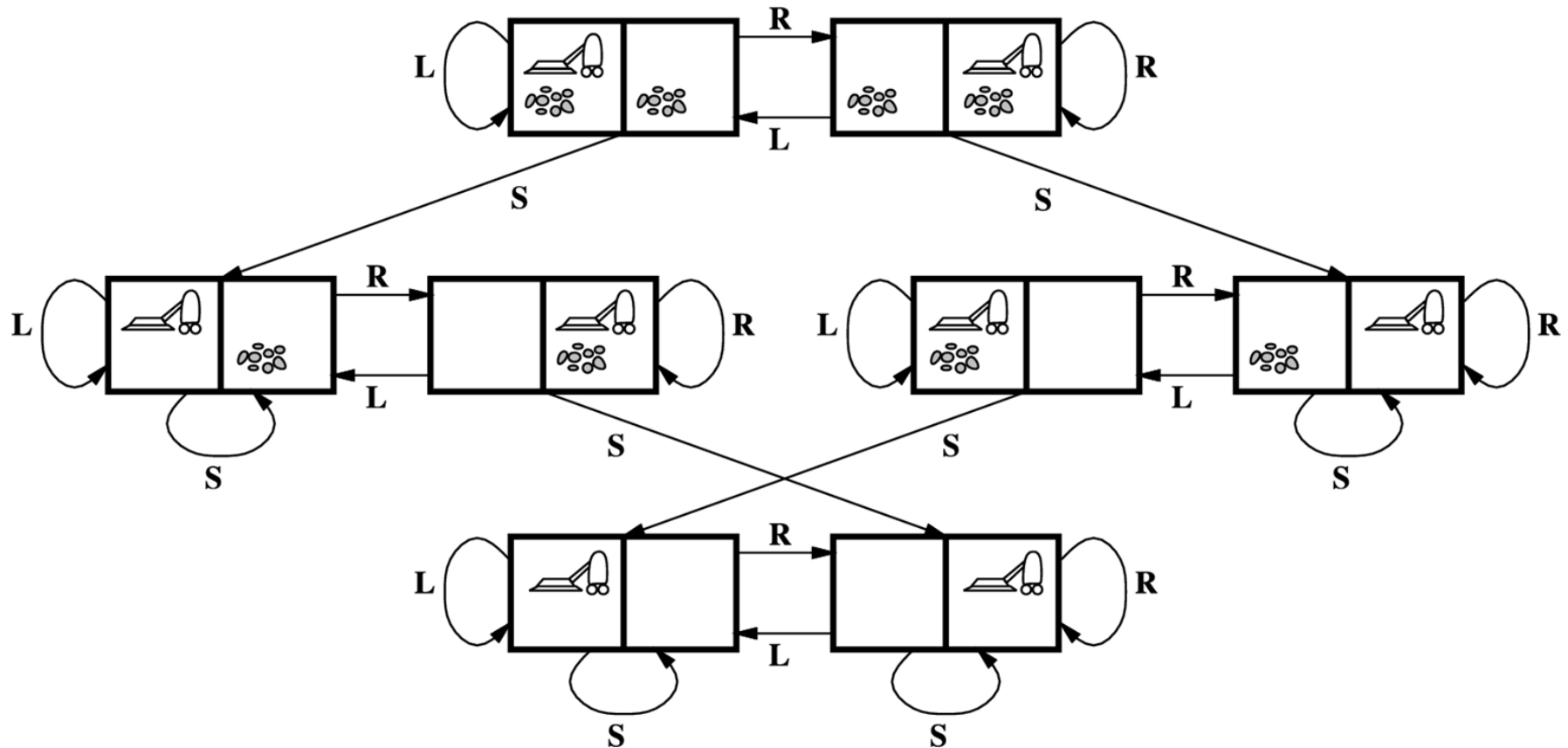
Problem Formulation

- **Initial state**
 - current state of environment and agent
- **State space**
 - all states accessible from the initial one
- **Successor function**
 - for each state, gives the valid actions and successor states
- **Goal test**
 - decides whether the goal is achieved in a state
- **Cost function**
 - determines the cost for each step of a sequence of actions

Example: Routes in Romania



The Vacuum Cleaner Micro-World



3x3 Tile Puzzle

7	2	4
5		6
8	3	1

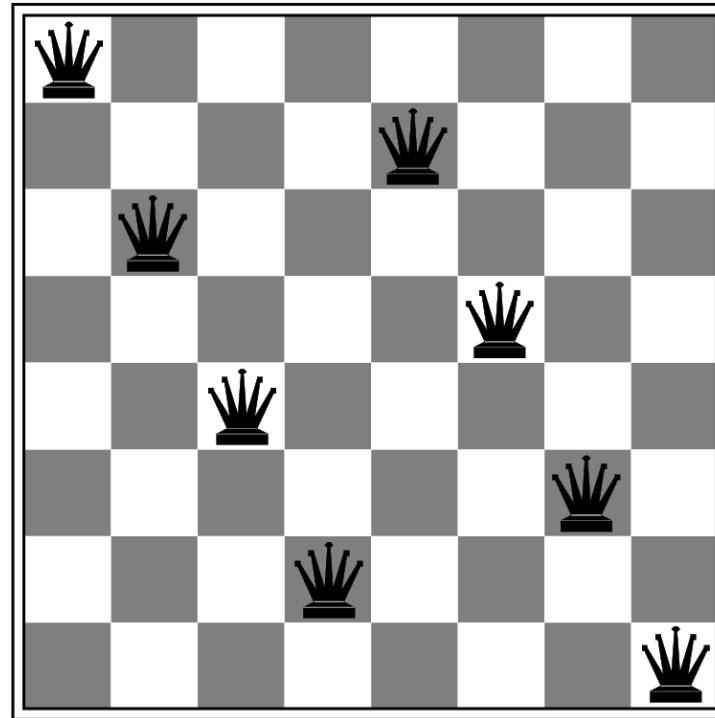
Start State

	1	2
3	4	5
6	7	8

Goal State

- generic class of *NP*-hard problems
- 4×4 puzzle: $16!/2 = 1,300,000,000,000$ states (easy)
- 5×5 puzzle: $25!/2 = 10^{25}$ states (hard)

The 8-Queen Problem



- incremental formulation (placing one queen at a time)
- full-state formulation (all queens placed on board)
- generalized problem: n queens on $n \times n$ board

Real-World Search Problems

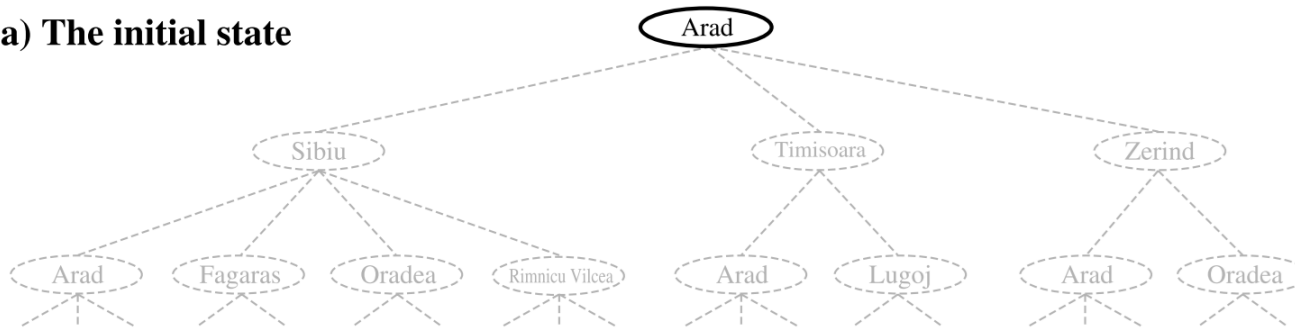
- **Routing problem**
- **Touring problem**
- **Traveling salesman problem**
- **VLSI layout**
- **Robot navigation**
- **Automatic assembly**
- **Protein synthesis**
- **Internet search**
- **...**

Search Tree

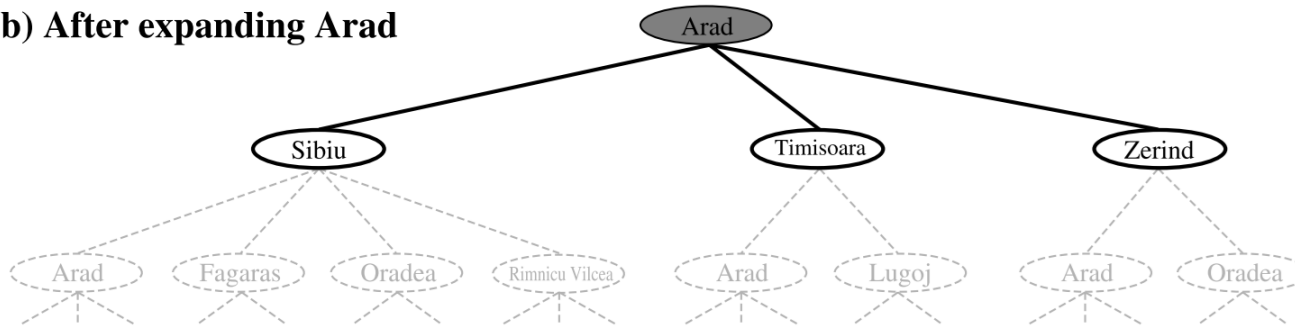
- **Search tree**
 - systematic generation of paths within the state space
 - search node: corresponds to a single state
 - root node: the node of the initial state
 - node expansion: generation of the successors of a node
 - generation method: successor function
 - search strategy: choosing where to expand the tree
- **Notes**
 - a node is different from a state
 - the search tree is different from the state space

Search Tree

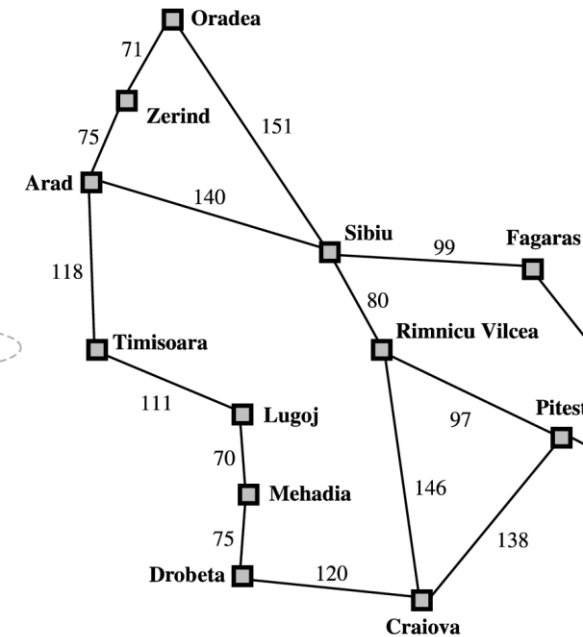
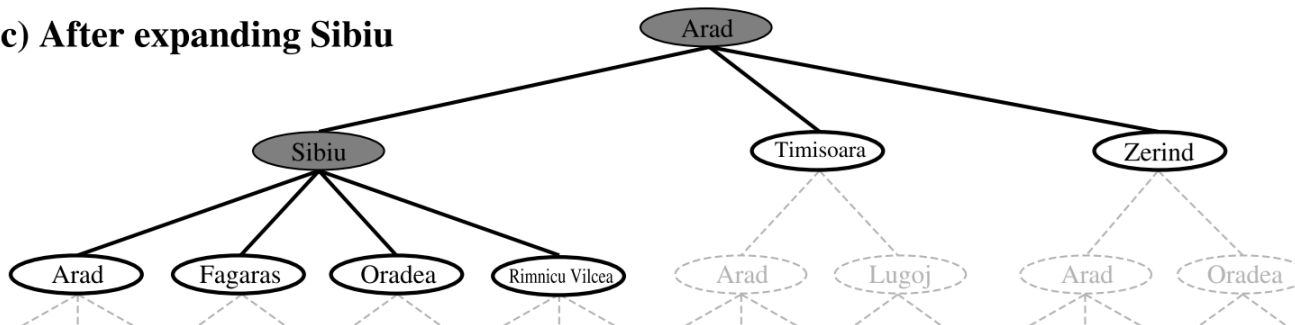
(a) The initial state



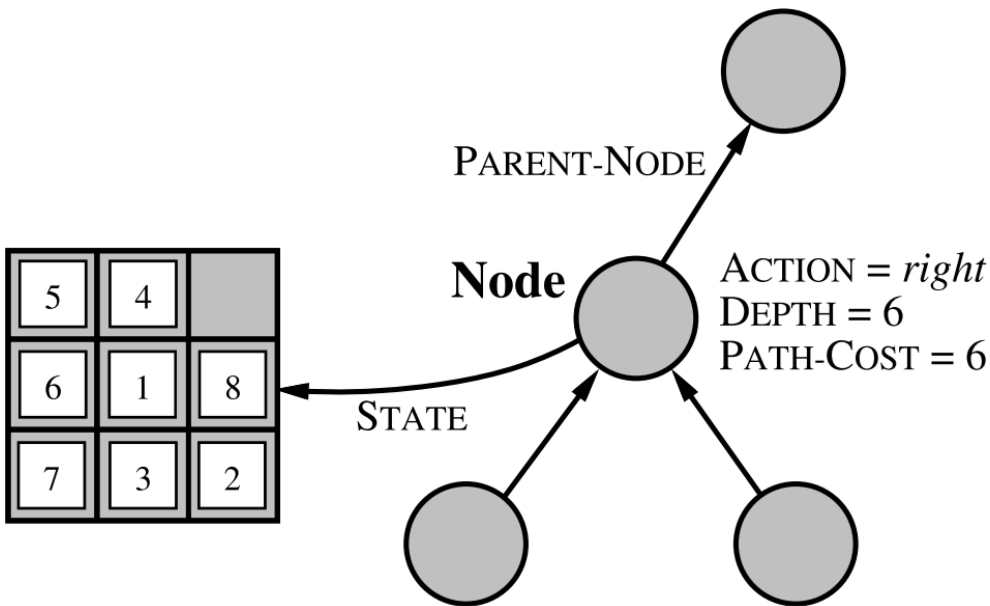
(b) After expanding Arad



(c) After expanding Sibiu



Search Node



■ Node: data structure

- state
- parent node
- action
- depth
- path cost

■ Fringe of nodes

- set of pending nodes
- waiting for expansion
- the leaves of the search
- implementation using some kind of queue

Informal Search Algorithm

function Tree-Search(*problem*, *strategy*) **returns** a solution or failure

initialization of the search tree using the initial state of the *problem*

loop do

if there are no candidate nodes for expansion

then return failure

choose a node from the fringe for expansion, according to the *strategy*

if the node contains a goal state

then return the corresponding solution

else expand the node and add the successor nodes in the search tree

Formal Search Algorithm (I)

function Tree-Search(*problem*, *fringe*) **returns** a solution or failure

fringe $\leftarrow$ InsertNode(MakeNode(InitialState[*problem*]), *fringe*)

loop do

if EmptyQueue(*fringe*) **then return** failure

node $\leftarrow$ RemoveFirstNode(*fringe*)

if GoalTest[*problem*] applied on State[*node*] succeeds

then return Solution(*node*)

fringe $\leftarrow$ InsertNodes(Expand(*node*, *problem*), *fringe*)

Formal Search Algorithm (II)

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

successors $\leftarrow$ empty set

for each $\langle action, effect \rangle$ in Successor-Function[*problem*](State[*node*]) **do**

s $\leftarrow$ new Node

State[*s*] $\leftarrow$ *effect*

Parent-Node[*s*] $\leftarrow$ *node*

Action[*s*] $\leftarrow$ *action*

Path-Cost[*s*] $\leftarrow$ Path-Cost[*node*] + Step-Cost(*node*, *action*, *s*)

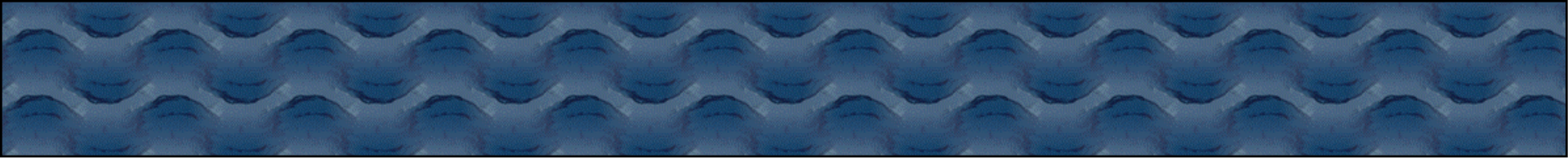
Depth[*s*] $\leftarrow$ Depth[*node*] + 1

addition of *s* to the set *successors*

return *successors*

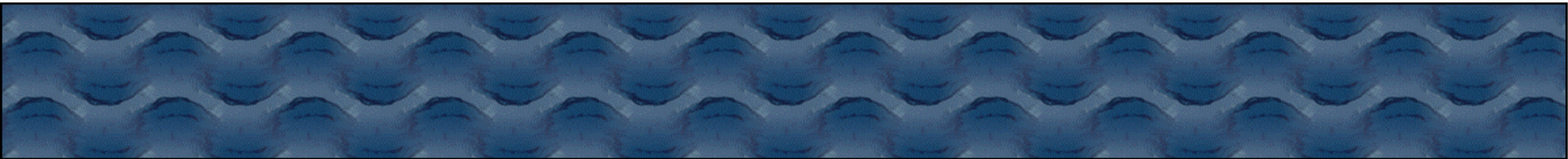
Measuring Search Performance

- **Completeness**
 - does it guarantee to find a solution, if one exists?
- **Optimality**
 - if it finds a solutions, does it guarantee to be an optimal one?
- **Time complexity**
 - how much time is required to find a solution?
- **Space complexity**
 - how much space (memory) is required to find a solution?
- **Problem size**
 - branching factor b (maximum number of children of a node)
 - depth of the closest solution d
 - maximum length of any path m

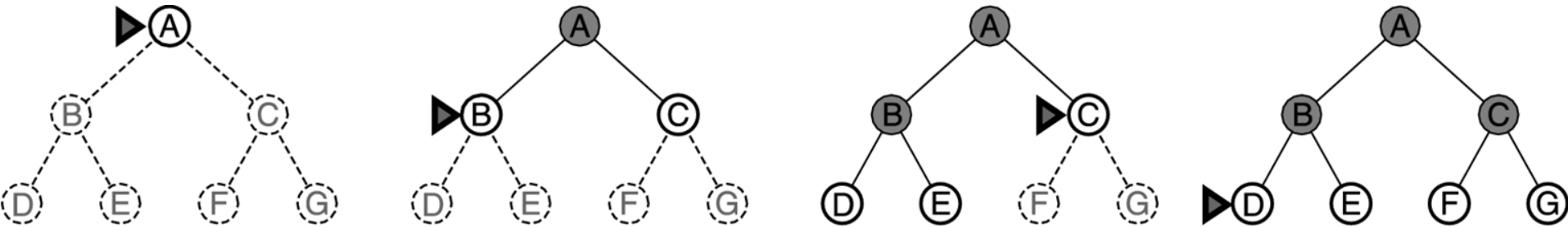


Uninformed or Blind Search

search with no guidance



Breadth-First Search



- expand the node with the least depth in the current fringe
- implementation with a FIFO queue (First In, First Out)
- complete, optimal (under conditions), exponential complexity $O(b^{d+1})$

Depth (d)	Nodes ($b=10$)	Time (10000 nodes/second)	Memory (1000 bytes/node)
2	1100	0.11 seconds	1 megabyte
4	111100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3523 years	1 exabyte

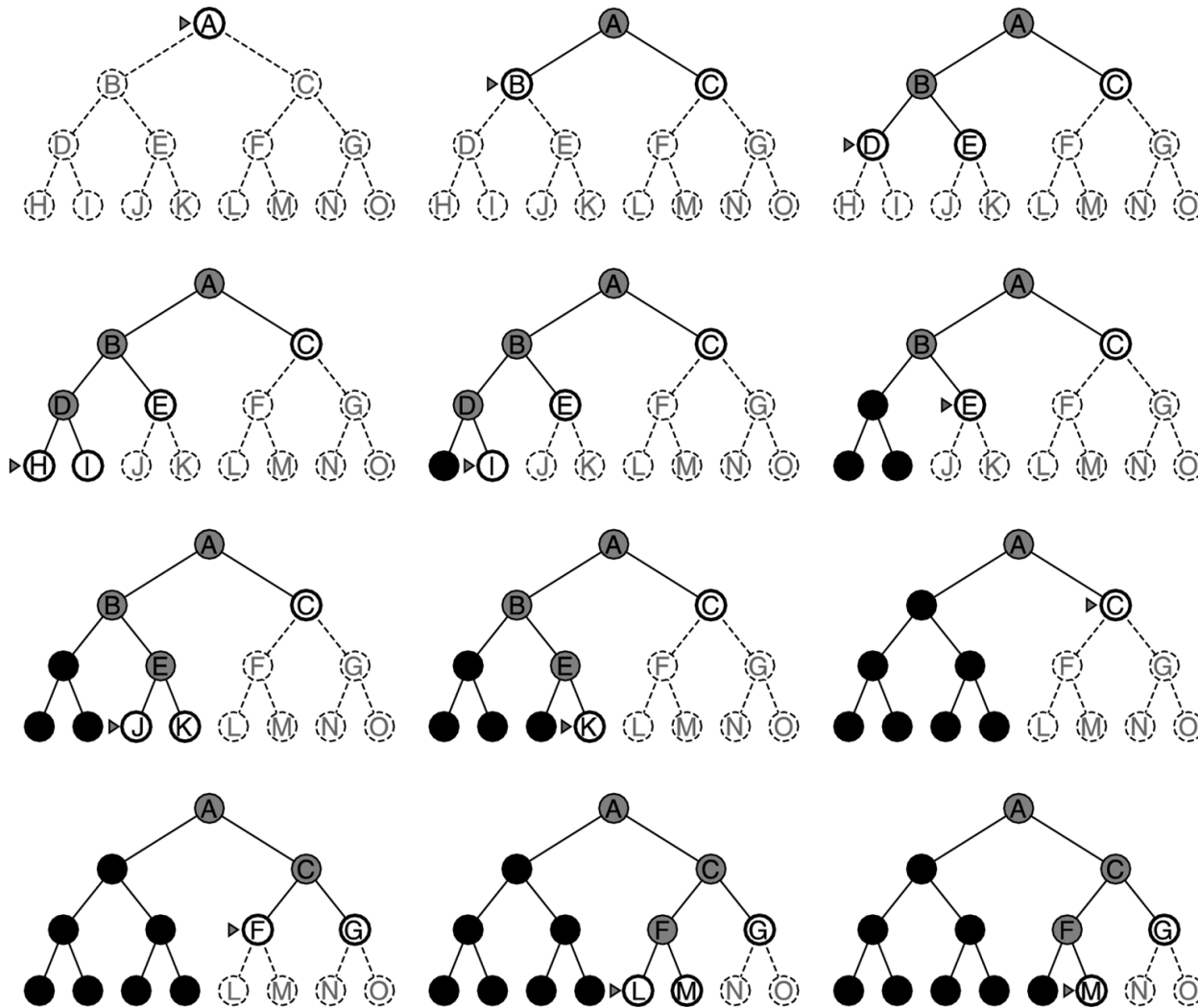
Uniform Cost Search

- variation of breadth-first search
- expand the node with the least path cost
- fringe: nodes with (almost) uniform cost
- complete, if the cost of each step is always positive
- optimal, if the cost of each step is always positive
- exponential complexity
 - depends on the optimal cost
 - depends on the minimum cost of each step
- possibly better, possibly worse performance

Depth-First Search

- expand the node with the maximum depth in the current fringe
- backtrack to the just previous node
- implementation with a LIFO queue (Last In, First Out) or a stack
- it is not complete (infinite-length paths)
- it is not optimal (proceeds to depth independently of cost)
- linear space complexity
 - improve with backtracking
 - improve with state change (requires the ability of undo)
- exponential time complexity

Depth-First Search



Depth-Limited Search

- variation of depth-first search
- cut-off expansion after a pre-determined depth
- heuristic determination of depth limit (graph diameter)
- advantage: elimination of infinite-length paths
- disadvantage: incompleteness
 - the solution may be located a little deeper
- disadvantage: suboptimality
 - an optimal solution may be located a little deeper

Iterative-Deepening Search

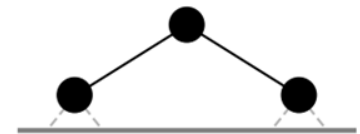
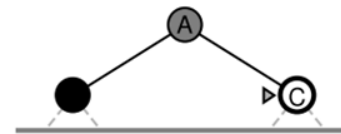
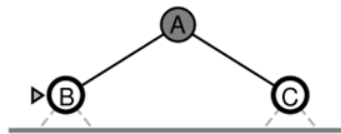
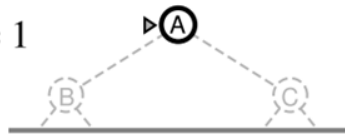
- smart management of depth-limited search
- incremental increase of depth limit ($L = 0, 1, 2, 3, \dots$)
- combines the advantages of depth-first and breadth-first search
- complete
- optimal (under conditions)
- space complexity $O(bd)$
 - linear, way less than the space complexity of breadth-first search!
- time complexity
 - exponential, but not worse $(d)b + (d-1)b^2 + \dots + (1)b^d = O(b^d)$!!!!
- variation: iterative-lengthening search

Iterative Deepening

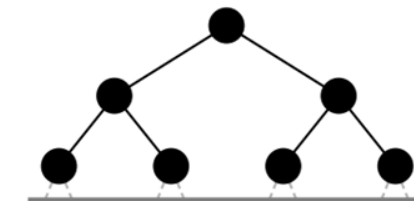
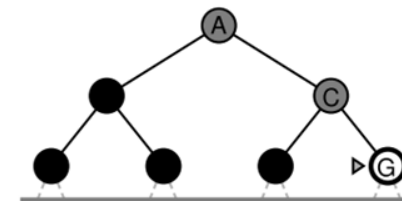
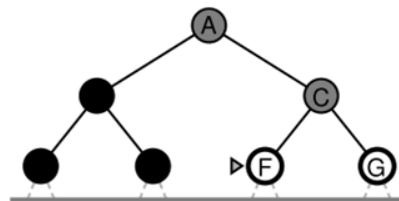
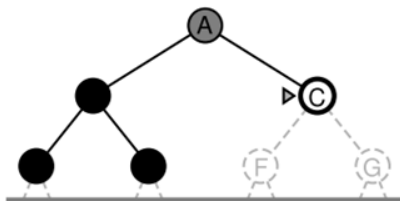
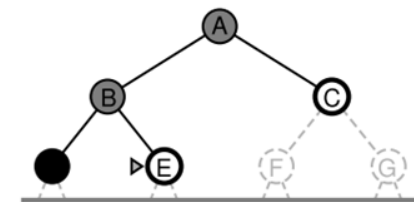
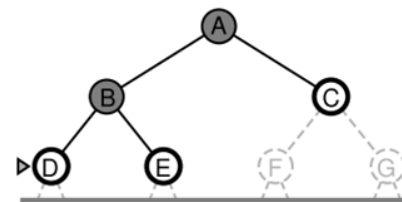
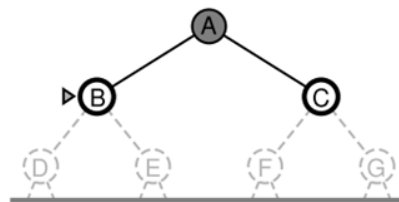
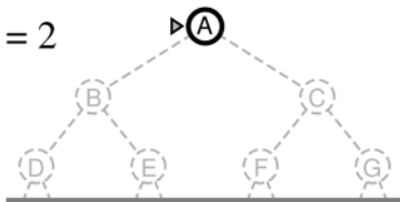
Limit = 0



Limit = 1

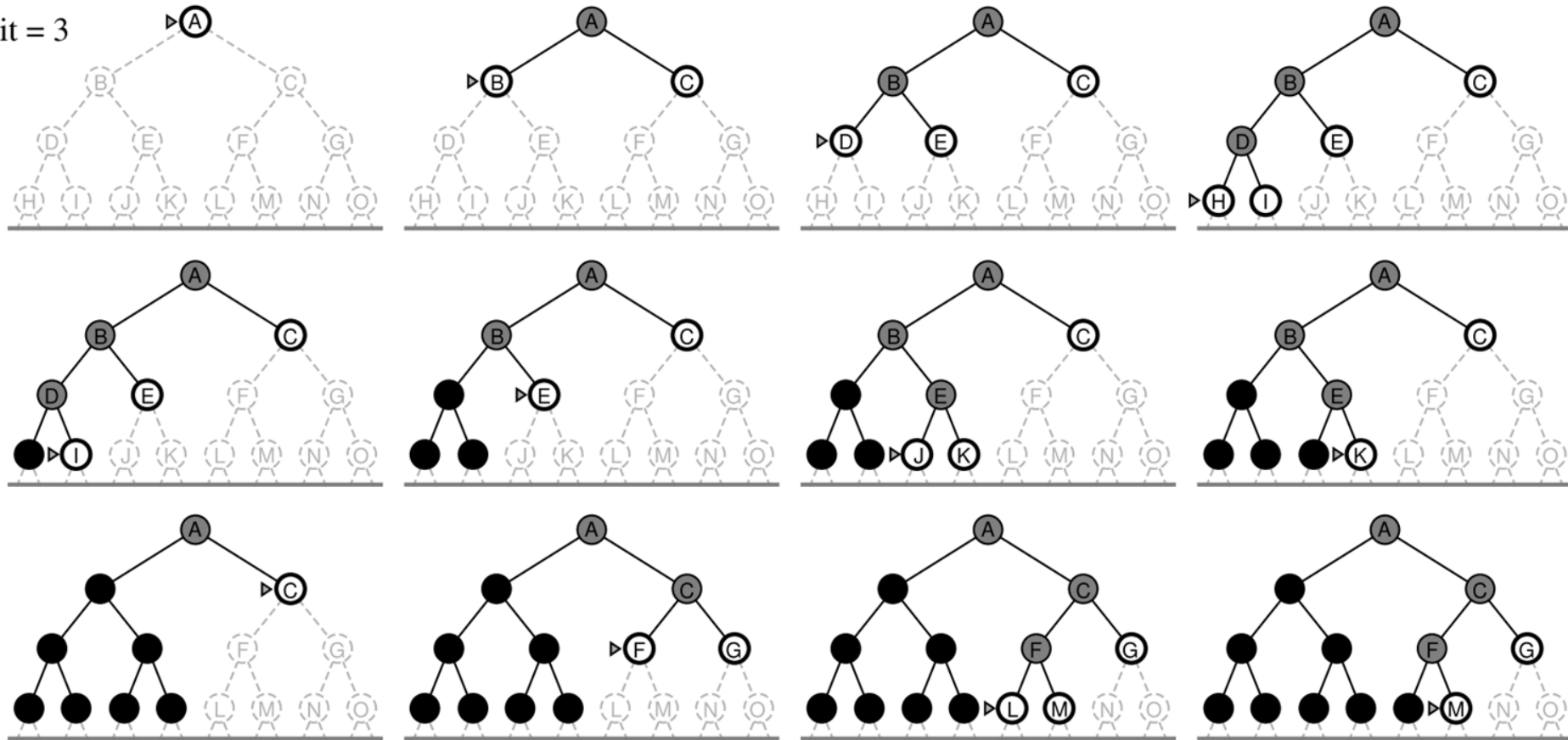


Limit = 2

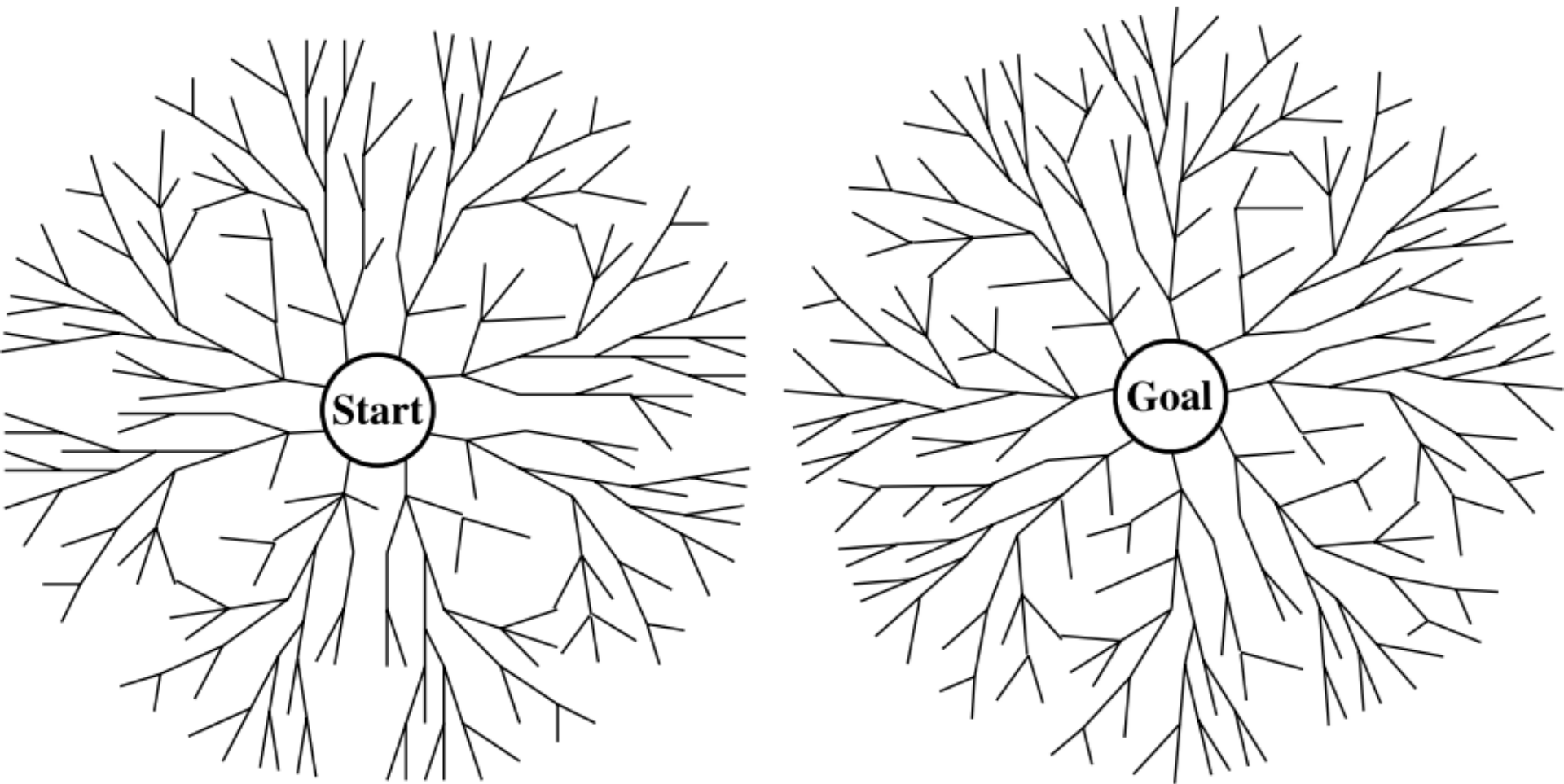


Iterative Deepening

Limit = 3

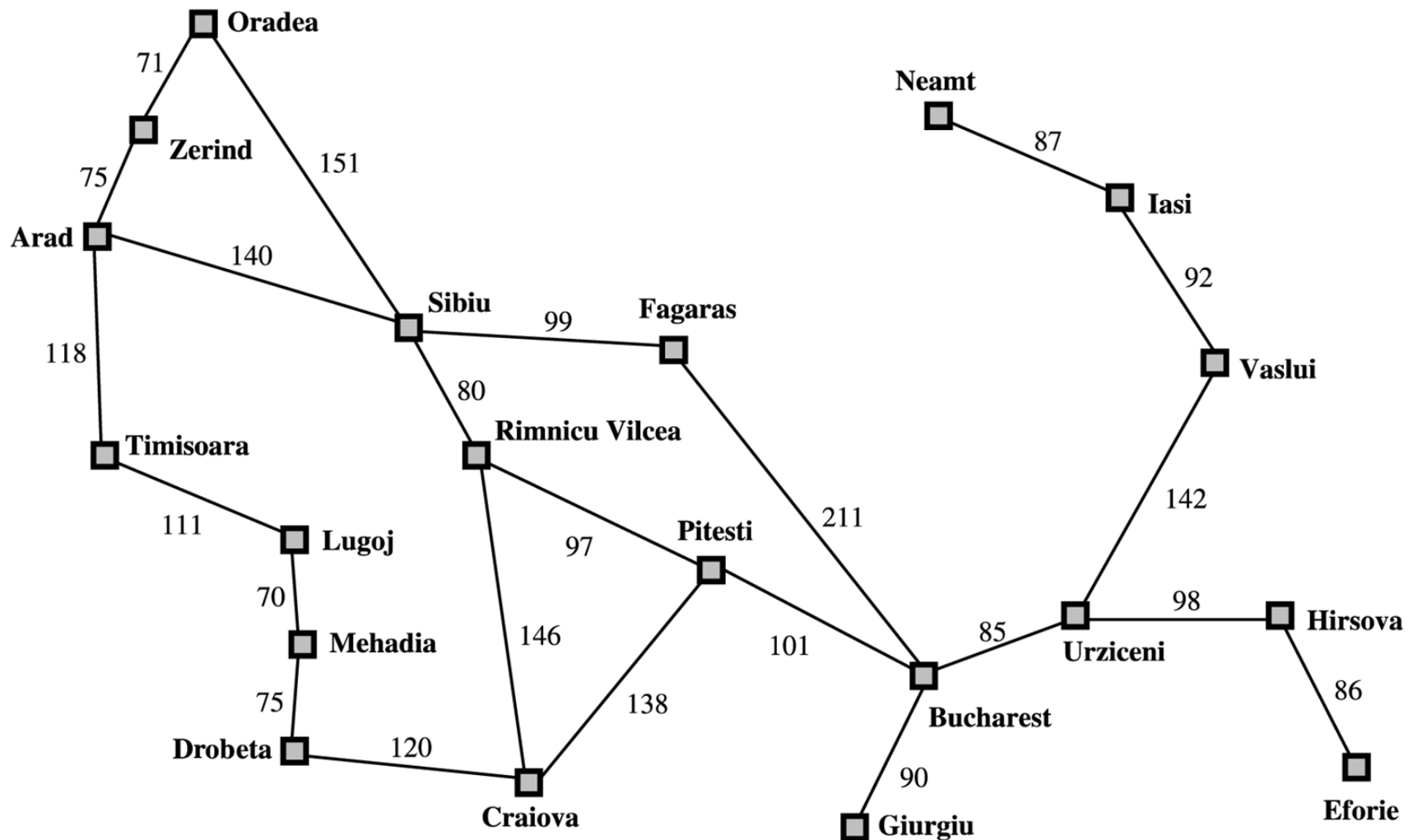


Bidirectional Search



$$b^{d/2} + b^{d/2} \ll b^d$$

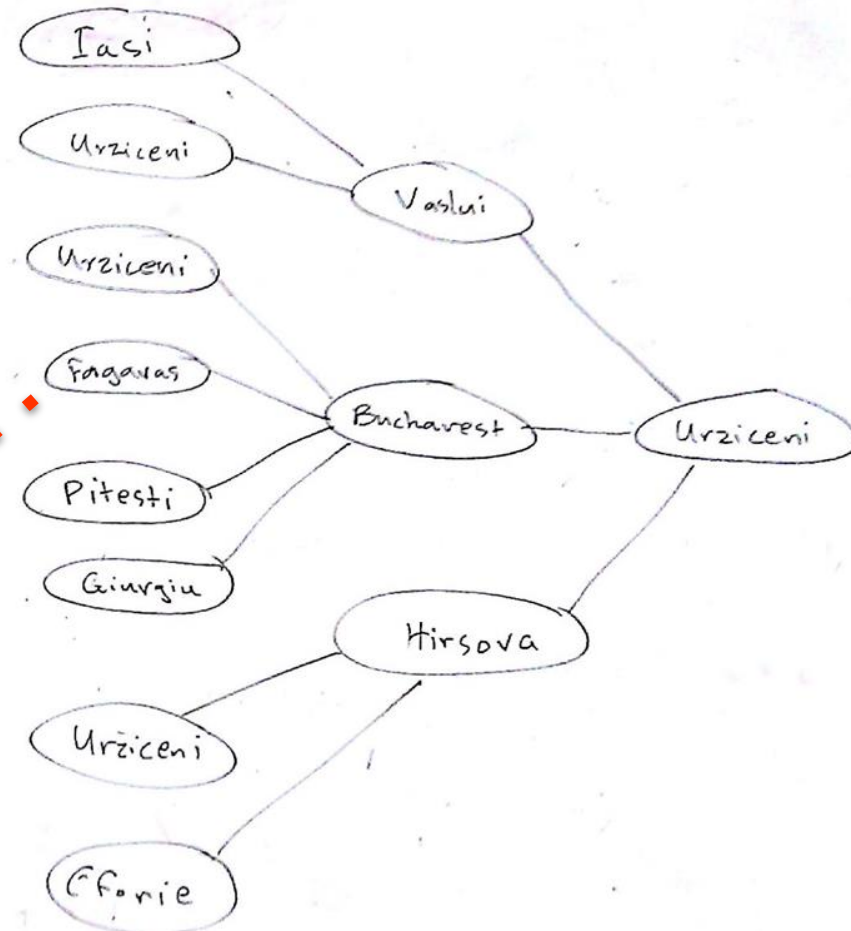
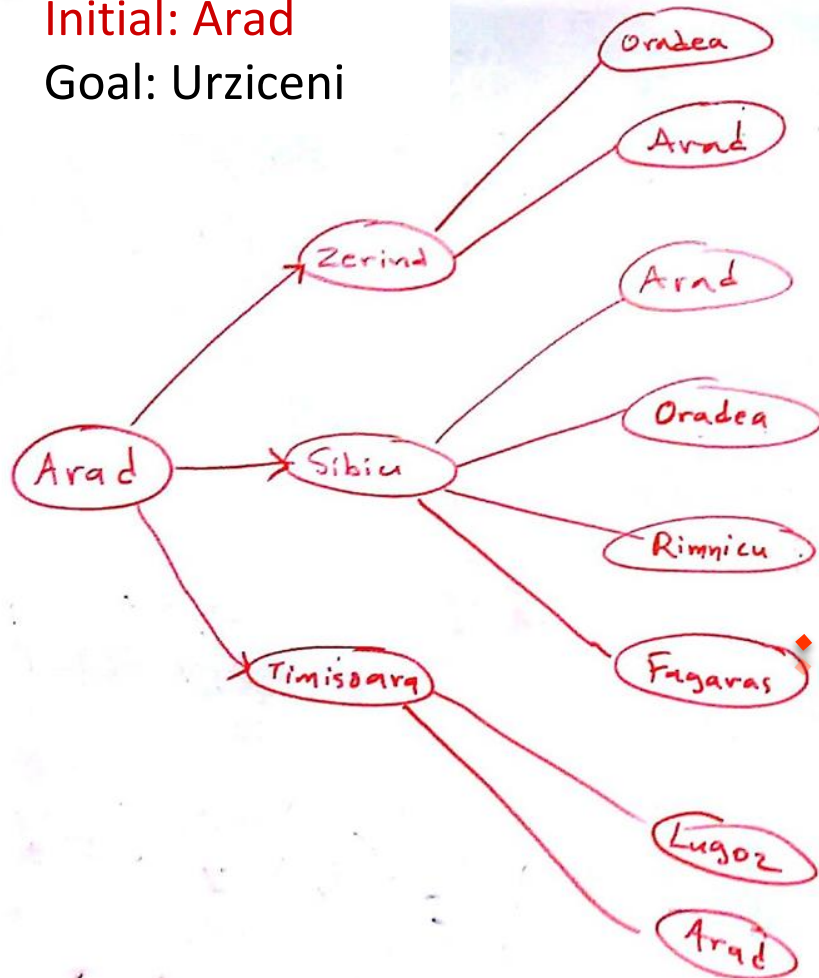
An Example: Routes in Romania



Bidirectional Search Example

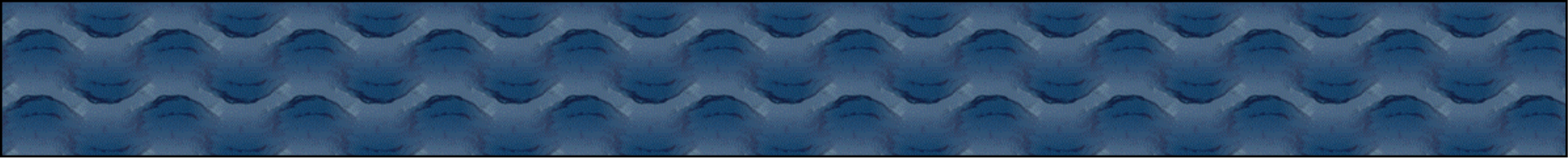
Initial: Arad

Goal: Urziceni



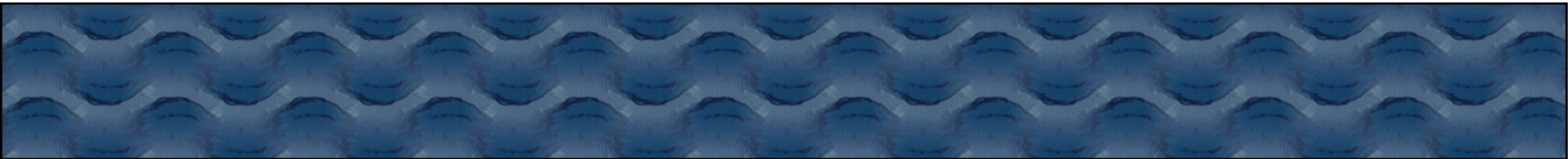
Comparing Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

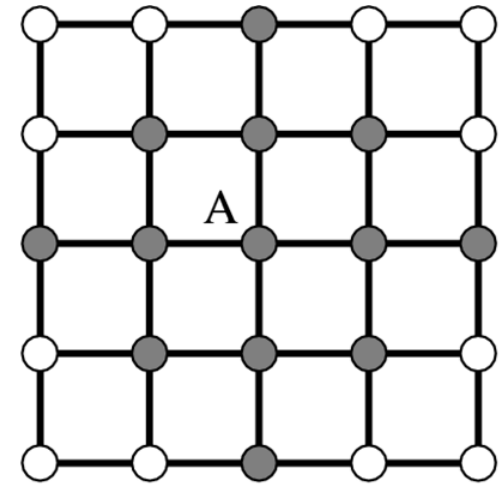
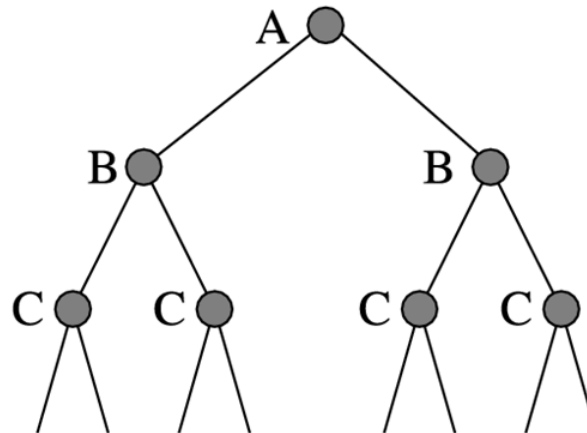
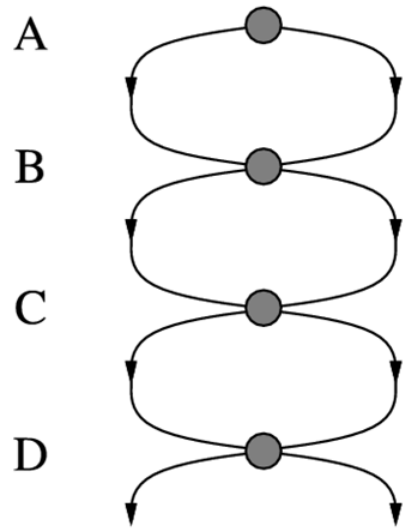


Repeated States

avoiding loops



Repeated States



- problem: a linear space starts looking exponential
- usually in problems with reversible actions
- simple comparison with nodes in the fringe is not enough

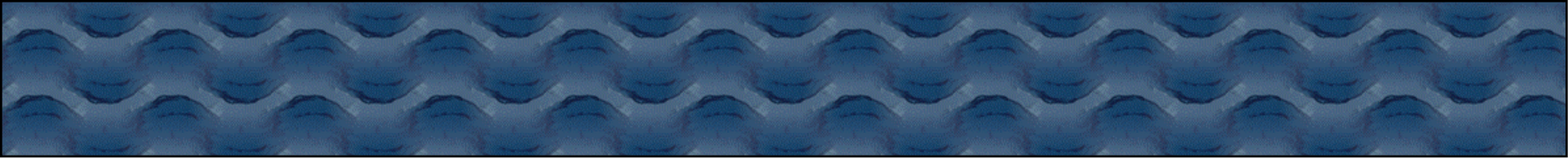
*algorithms that forget their history
are doomed to repeat it!*

Avoiding Repeated States

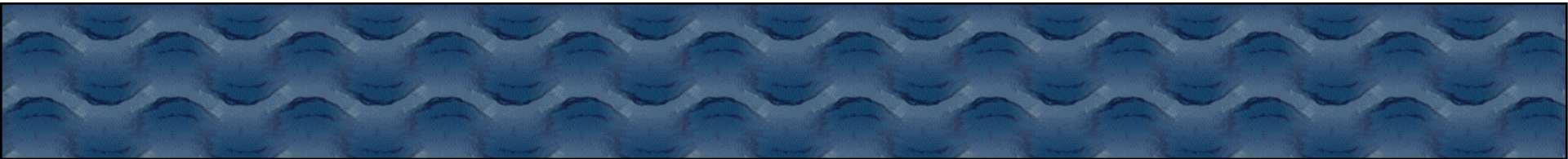
- **Memorization**
 - closed list: expanded nodes
 - open list: nodes to be expanded – fringe
- **Advantages**
 - space and time requirements proportional to state space
 - all search algorithms become complete for finite spaces
 - quick check in the closed list using a hash table
- **Disadvantages**
 - no algorithm has linear time requirements anymore
 - attention must be paid to alternative paths

Avoiding Repeated States: Algorithm

```
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
```



Search with Partial Information



Problems with Partial Information

- **Sensorless problems**
 - the agent cannot perceive the current state
- **Contingency problems**
 - partially-observable environments or uncertain actions
 - percepts provide some information and restrict outcomes
- **Adversarial problems**
 - uncertainty comes from other agents
- **Exploration problems**
 - unknown state space and unknown actions
 - extreme case

Sensorless Problems

- **Features**

- known state space and known action effects
- unknown current state (initial, intermediate, final)

- **Confrontation**

- reasoning with sets of states, not just states
- belief space: powerset of state space
 - belief: the subset of currently possible states
 - actions: transitions between beliefs (union of simple transitions)
- initialization: belief with all possible states
- solution: search in the belief space
- goal: a belief where all states are goal states

Actions without Sensors

□ Coercion

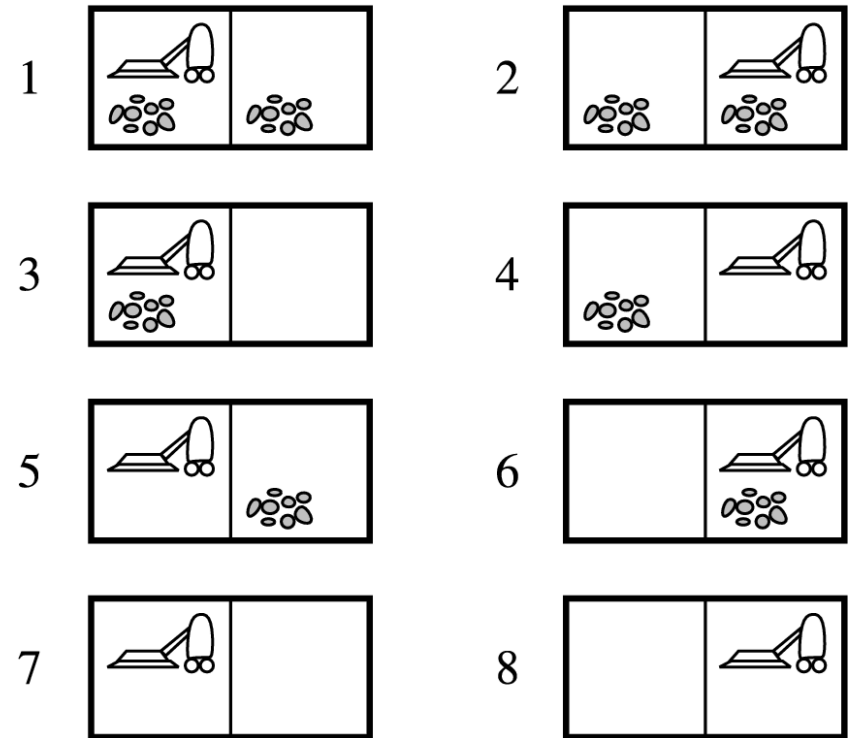
- choice of an action to exclude certain states

□ Vacuum Cleaner Example

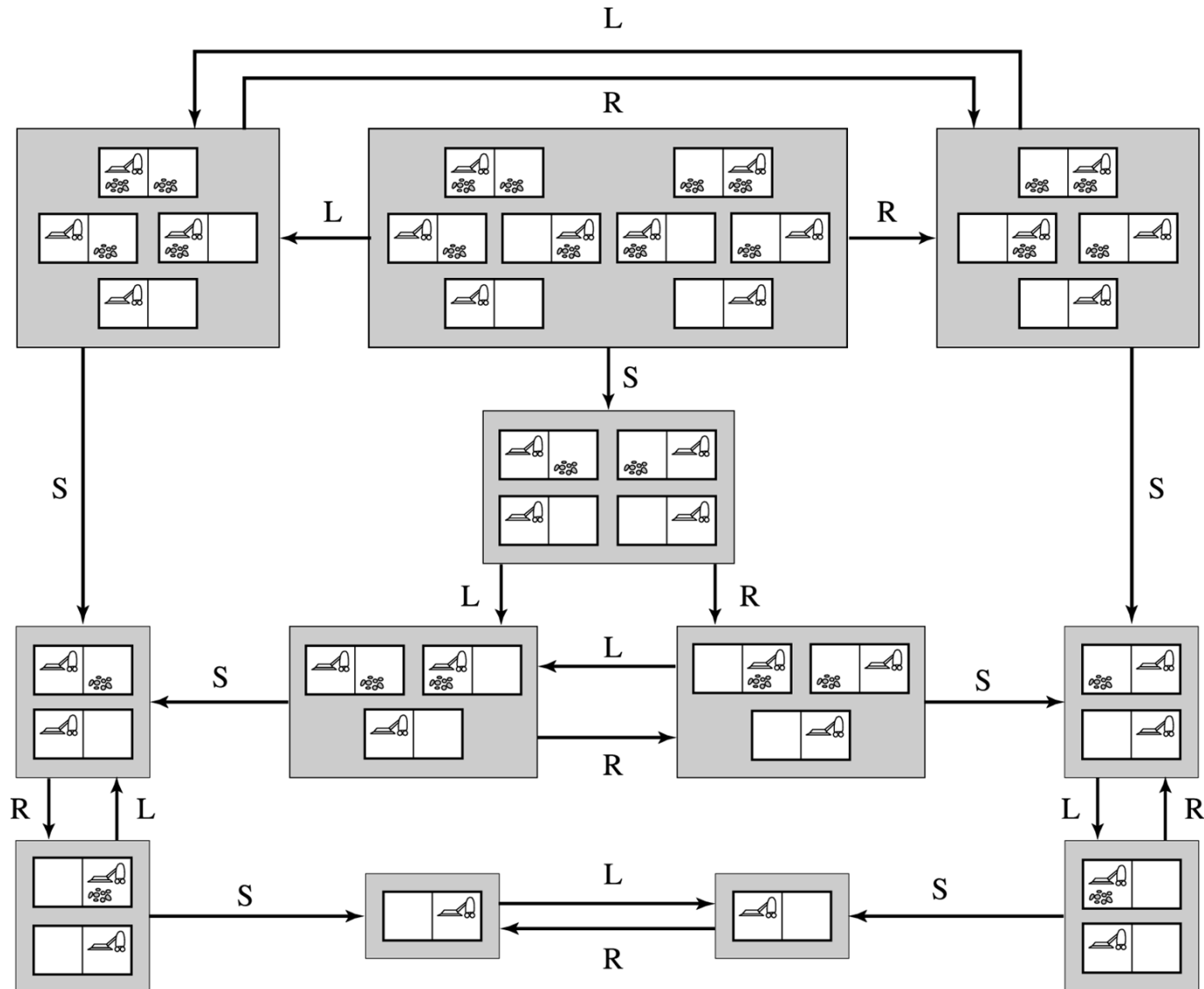
- initial belief {1,2,3,4,5,6,7,8}
- action: *Right*
- next belief: {2,4,6,8}
- action: *Suck*
- next belief: {4,8}
- ...

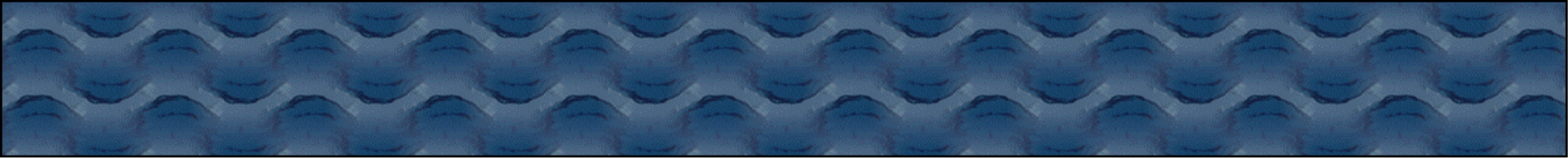
□ Generalization

- non-deterministic actions with uncertain effects



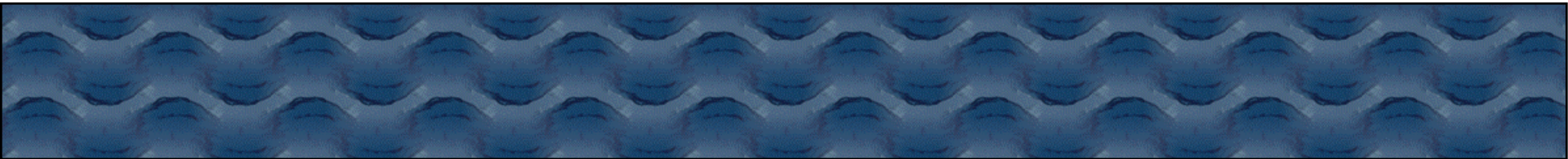
Vacuum Cleaner without Sensors





Informed Search

searching with guidance



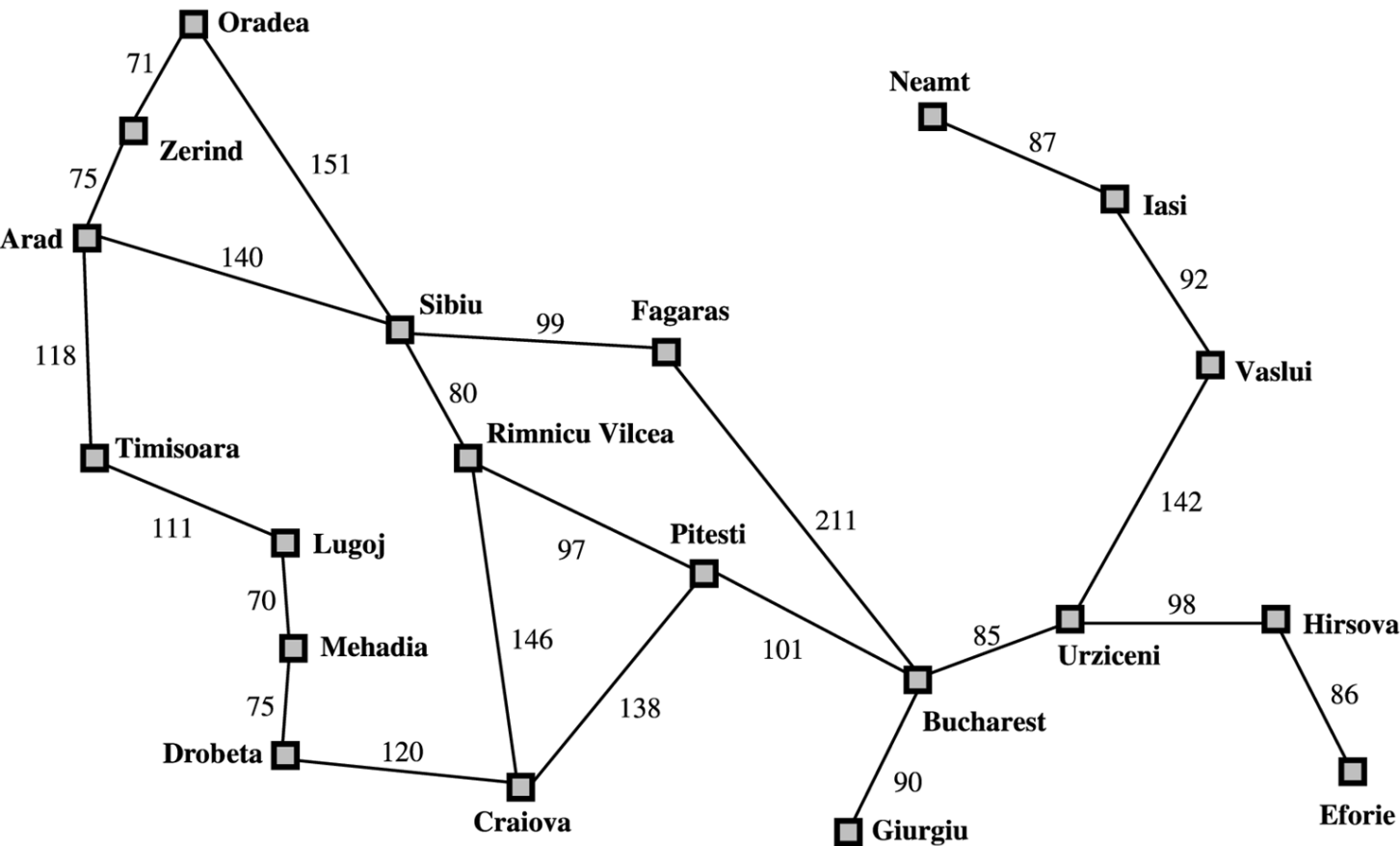
Best-First Search

- **Informed search**
 - use of domain knowledge about the problem, not just definition
 - uninformed search: depth, cost, repeat, etc.
 - informed search: extraction of information from the state, exploitation of domain knowledge
- **Best-first search**
 - evaluation function for the “best”
 - evaluation of each node n with the evaluation function $f(n)$
 - expand the node with the minimum value
 - implementation using a priority queue for the nodes in fringe

Greedy Best-First Search

- **Evaluation function**
 - heuristic function $h(n)$, $h(G)=0$
 - estimated cost of cheapest path from node n to the goal
 - the estimate depends only on node n and the goal
 - evaluation: $f(n) = h(n)$
- **Example: Routes in Romania**
 - $h_{\text{SLD}}(n)$: Straight-Line Distance (SLD) heuristic
 - e.g. SLD of city n from Bucharest (goal) in Routes in Romania
 - it cannot be computed easily from the search problem definition
 - it can be computed easily from the city coordinates on a map

Routes in Romania: SLD from Bucharest



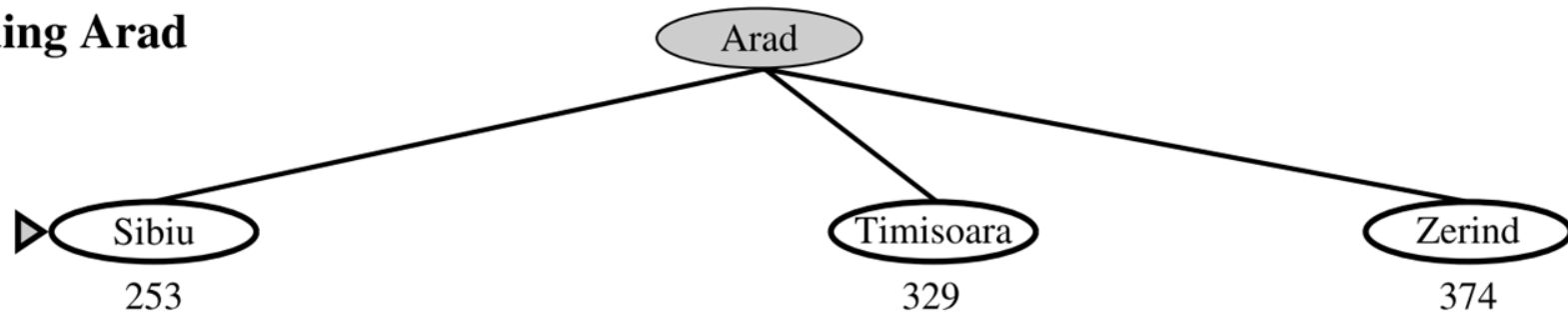
City	h_{SLD}	City	h_{SLD}
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Routes in Romania (GBFS)

(a) The initial state

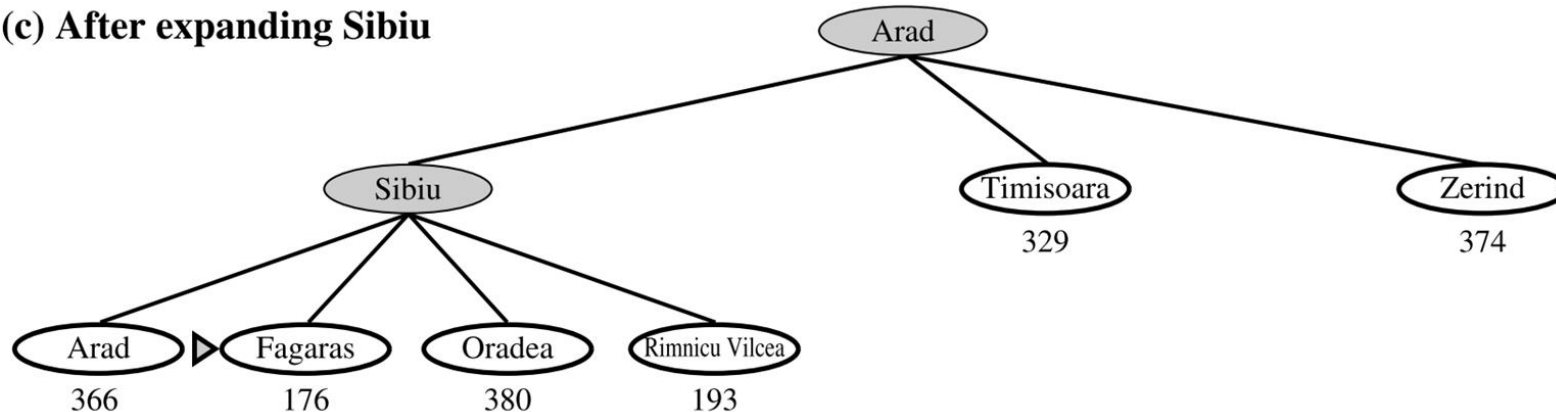


(b) After expanding Arad

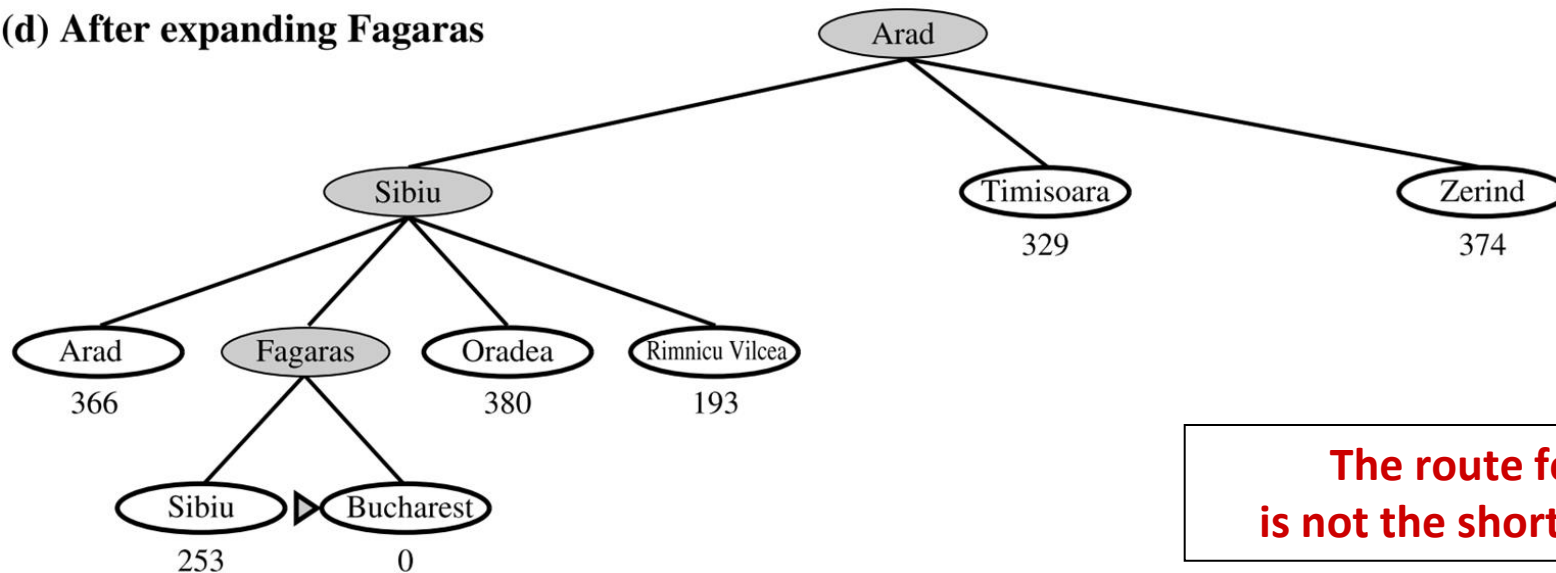


Routes in Romania (GBFS)

(c) After expanding Sibiu



(d) After expanding Fagaras



**The route found
is not the shortest one!**

Greedy Best-First Search

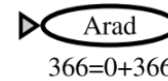
- **Completeness**
 - no in general, can get stuck in infinite loops (Iasi → Fagaras)
 - complete in finite spaces, if repeated nodes are avoided
- **Optimality**
 - no, subject to erroneous estimates (Sibiu → Bucharest)
- **Time and Space Complexity**
 - exponential $O(b^m)$, where m is the maximum depth of the space
- **Overall**
 - looks like depth-first search

A* Search

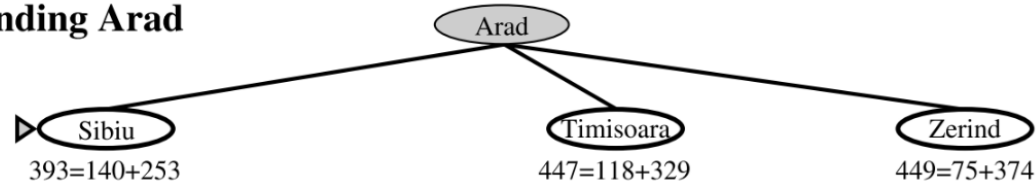
- **Evaluation function**
 - estimated cost $h(n)$ of cheapest path from node n to the goal
 - real cost $g(n)$ of current path from initial state to node n
 - evaluation: $f(n) = g(n) + h(n)$ – total path cost through node n
- **Admissible heuristic**
 - always makes optimistic estimates
 - does not overestimate costs
- **Theorem**
 - A* search without pruning of repeated states is optimal, if the heuristic function $h(n)$ is admissible.

Routes in Romania (A*)

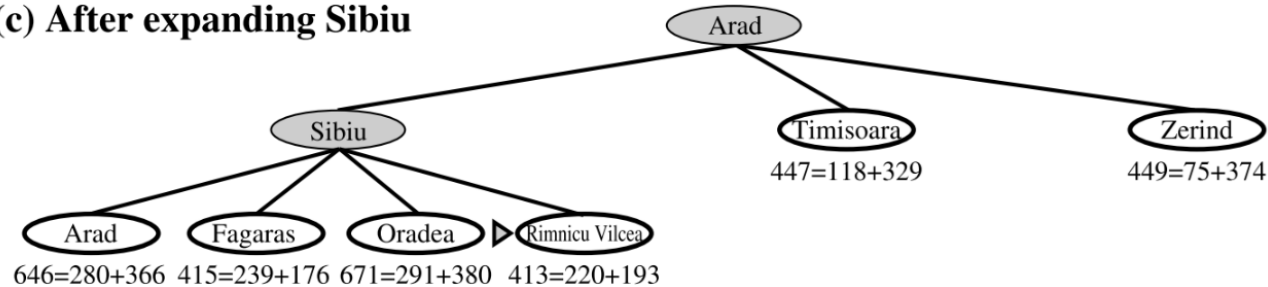
(a) The initial state



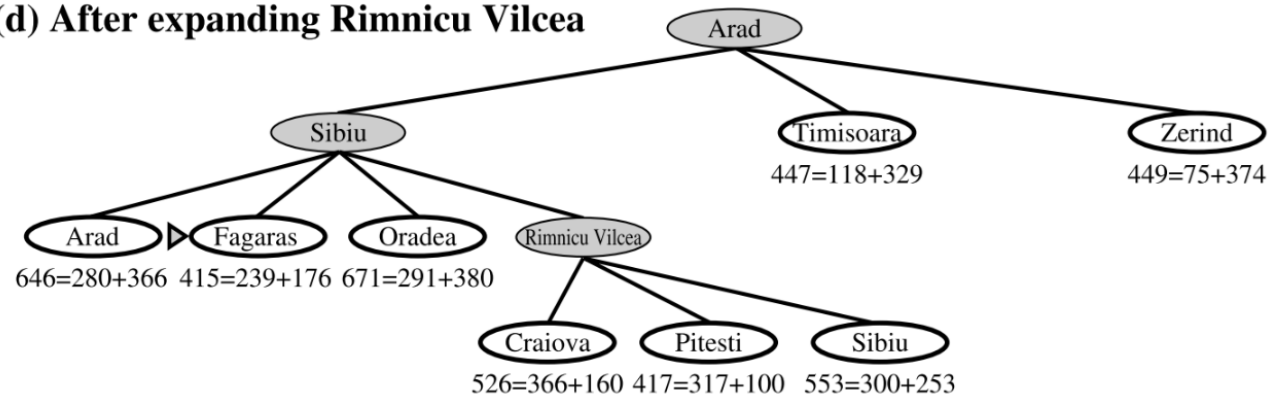
(b) After expanding Arad



(c) After expanding Sibiu

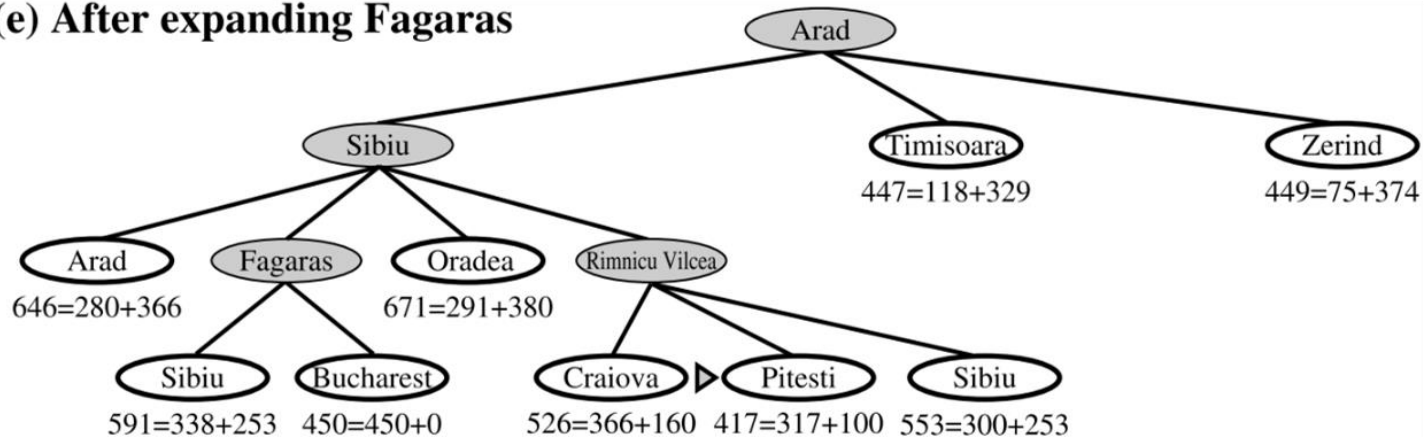


(d) After expanding Rimnicu Vilcea

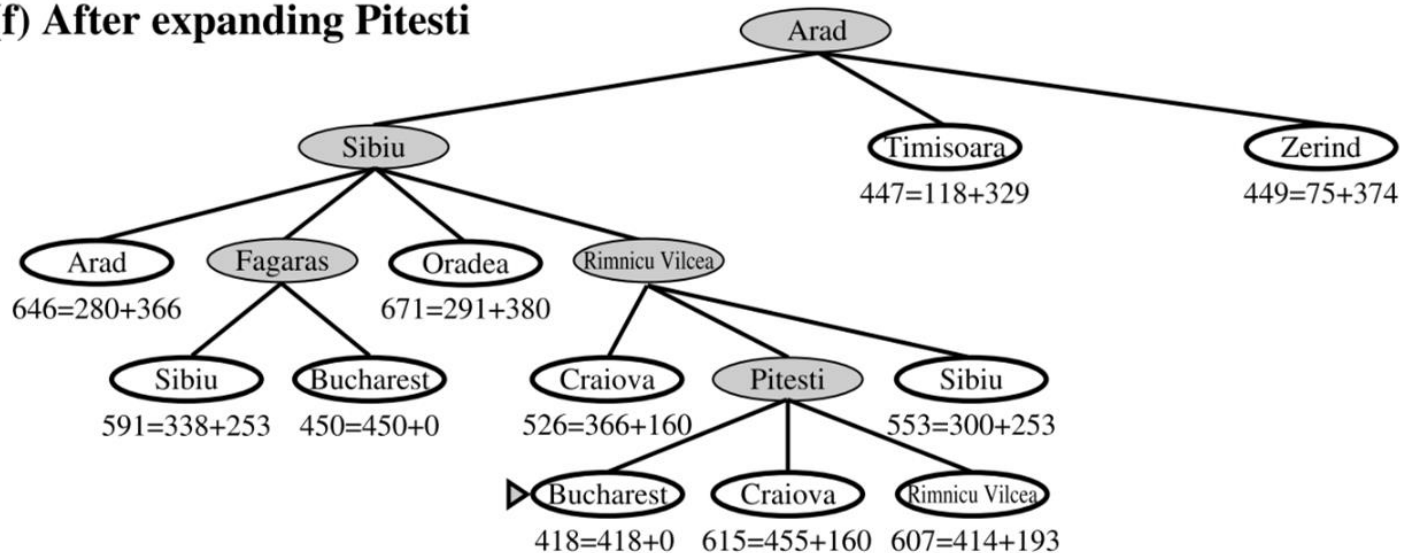


Routes in Romania (A*)

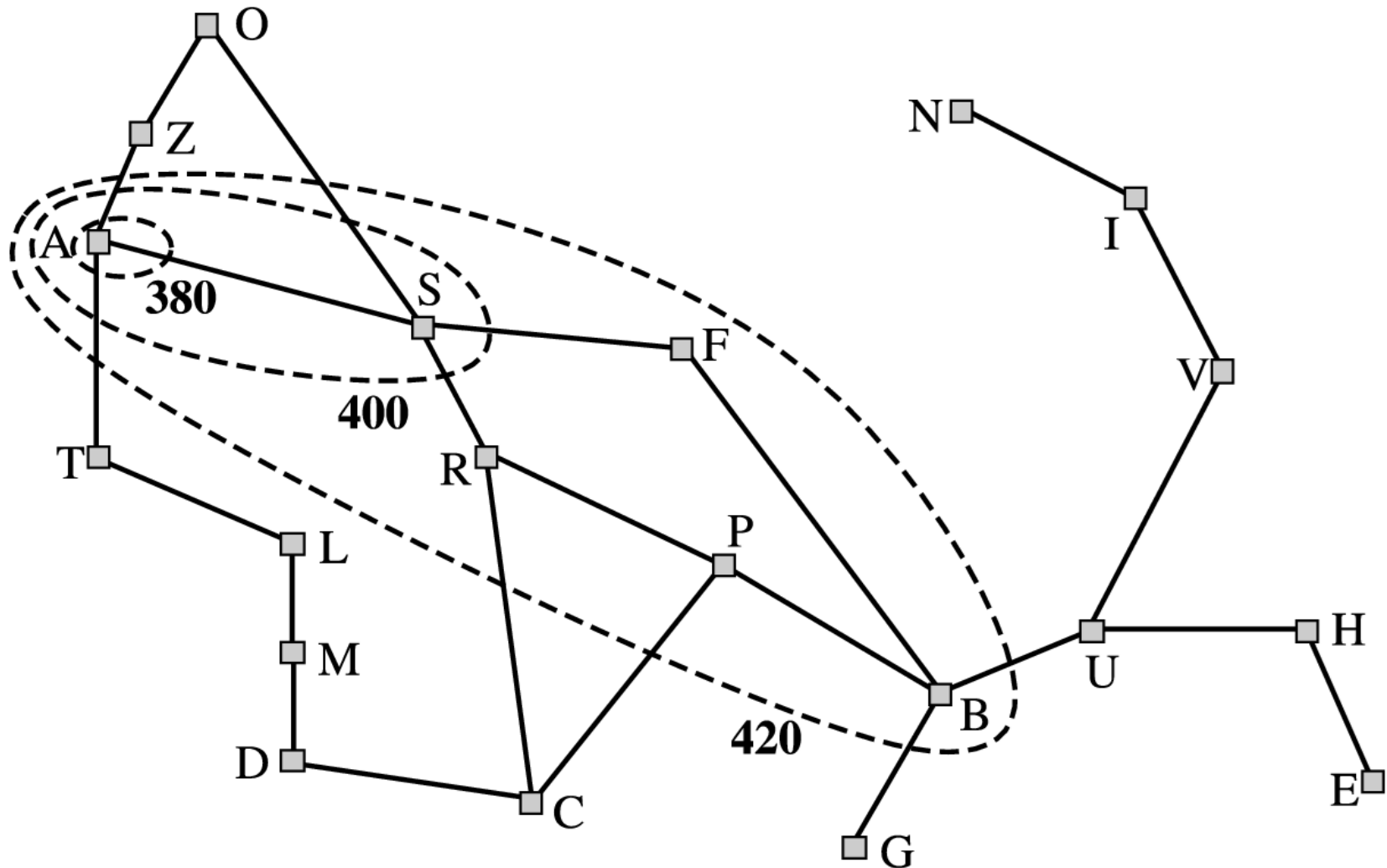
(e) After expanding Fagaras



(f) After expanding Pitesti

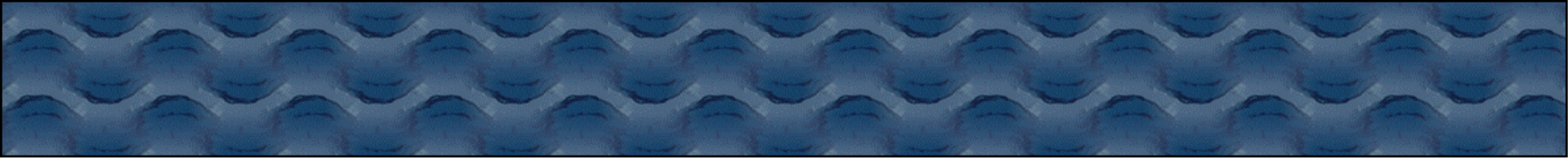


Cost Contour Lines



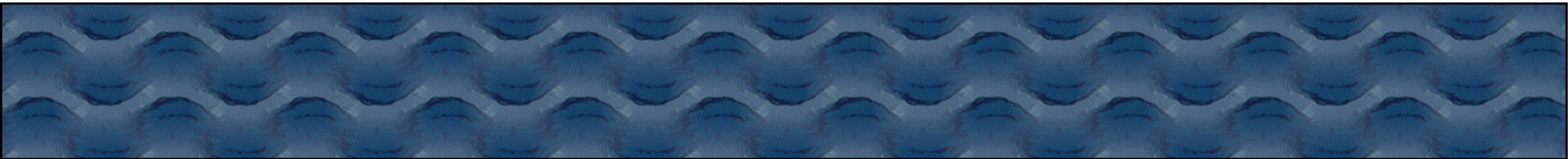
A* Performance

- **Completeness**
 - yes, unless there are infinite nodes n with $f(n) \leq f(G) = C^*$
- **Optimality**
 - yes, with an admissible heuristic
 - expansion: all with $f(n) < C^*$, some with $f(n) = C^*$, none with $f(n) > C^*$
- **Complexity**
 - exponential requirements in time and (mostly) in memory
- **Optimally efficient**
 - for the same problem, A* explores the fewest states of all the optimal and complete search algorithms



Informed Search with Bounded Memory

can we save on memory?



Iterative-Deepening A* - IDA*

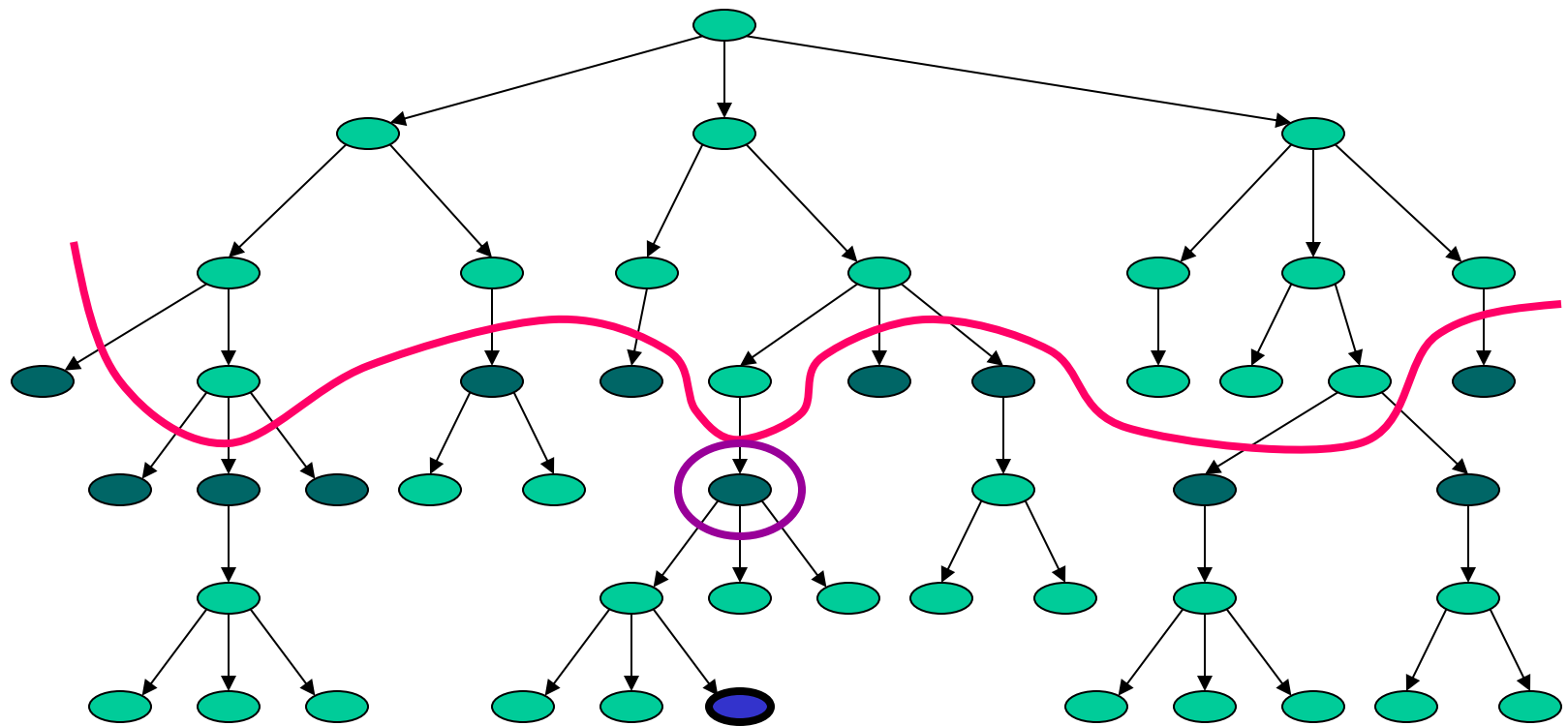
▪ Methodology

- analogous to iterative lengthening search
- search up to a depth with limited cost $f(n) = g(n) + h(n)$
- gradual increase of the cut-off limit $0, f_1, f_2, f_3, \dots$
- new limit f_i : the least $f(n)$ over the limit f_{i-1} at stage $i-1$
- complete and optimal with admissible h
- linear space complexity $O(bd)$
- exponential time complexity $O(b^d)$

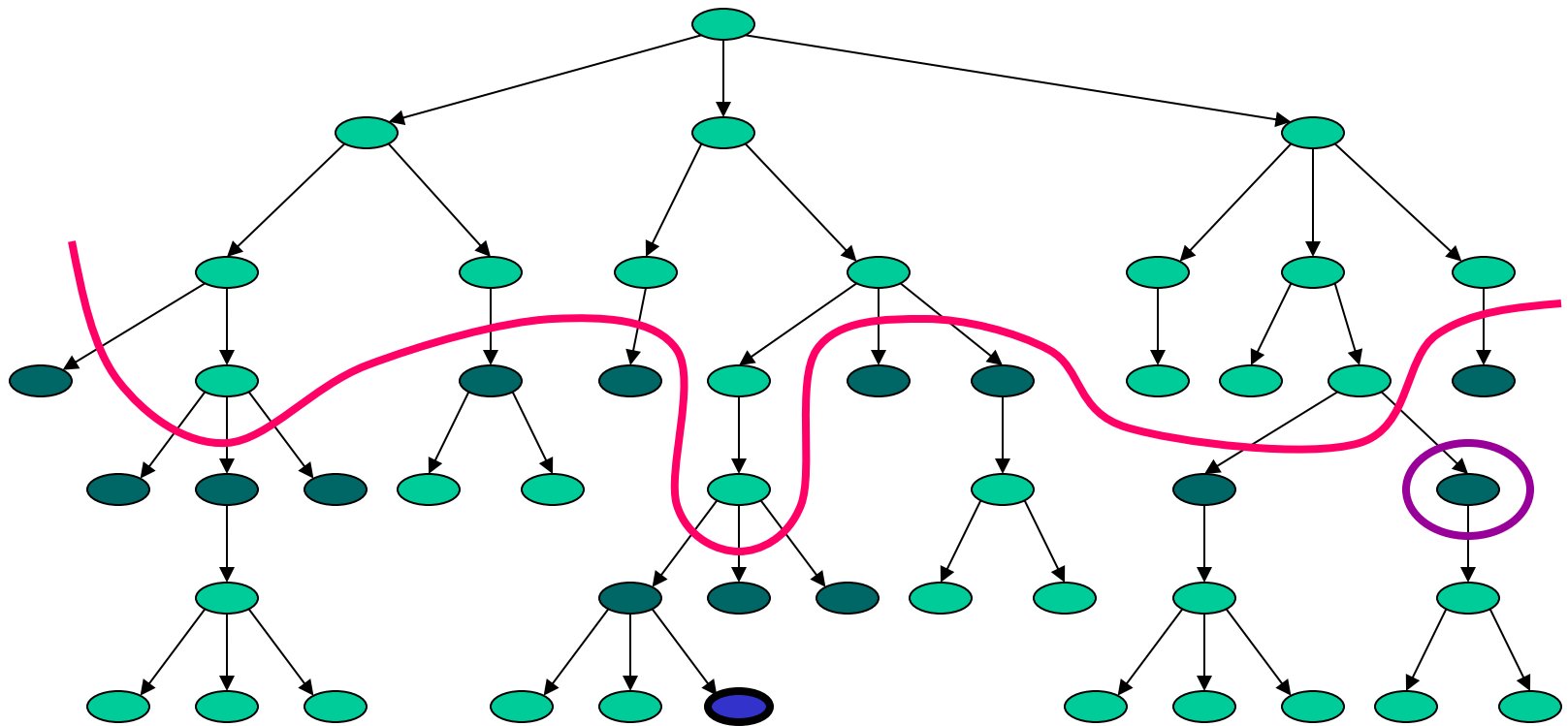
▪ Disadvantages

- possible, slow increase of the cut-off limit $\Rightarrow$ slow progress
- does not detect repeated states

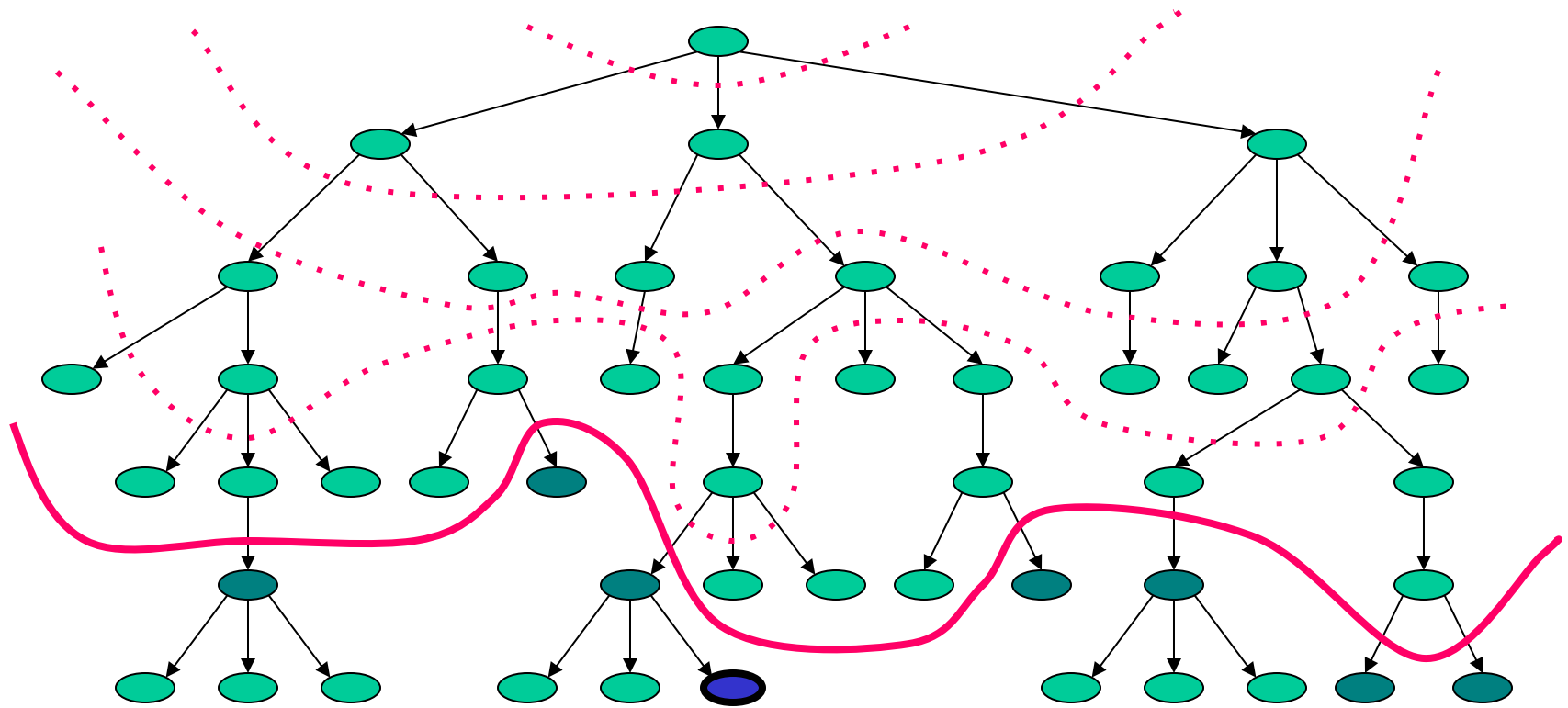
IDA* - Searching up to a Cut-Off Limit

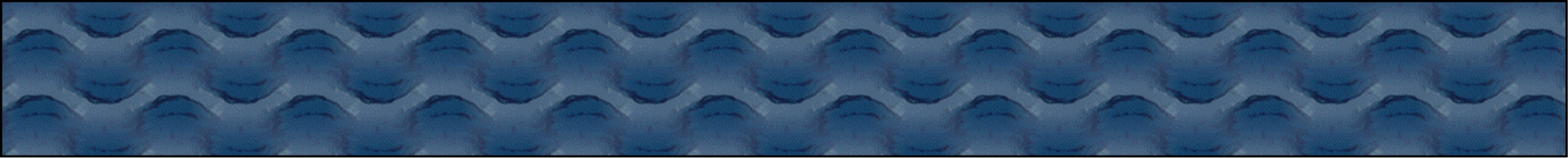


IDA* - Increasing the Cut-Off Limit



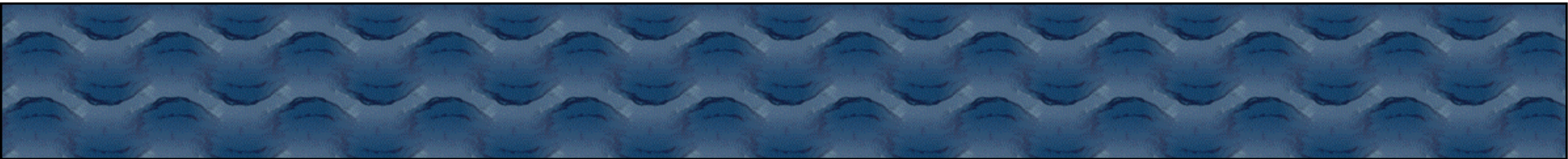
IDA* Search Stages





Heuristic Functions

providing guidance



8 Tile Puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$$h_1 = 8$$

$$h_2 = 18$$

$$h^* = 26$$

Heuristic functions

- h_1 = the number of tiles not in the correct place
- h_2 = the sum of distances of the tiles from their destination
 - Manhattan distance (distance of building blocks)
- admissible heuristic functions (does not overestimate)
- $h_1 \leq h_2 \Rightarrow h_2$ dominates over $h_1 \Rightarrow h_2$ expands less nodes

Comparing Heuristic Functions

– uniform tree: $N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$

d	Search Cost N			Active Branching Factor b^*		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	—	539	113	—	1,44	1,23
16	—	1301	211	—	1,45	1,25
18	—	3056	363	—	1,46	1,26
20	—	7276	676	—	1,47	1,27
22	—	18094	1219	—	1,48	1,28
24	—	39135	1641	—	1,48	1,26

Inventing Admissible Heuristics

- **Relaxed problems**
 - simplification or relaxation of the rules of the problem
 - in general, the relaxed problem can be easily solved optimally
 - $f(\text{optimal solution of relaxed}) \leq f(\text{optimal solution of original})$
 - admissible h : cost of an optimal solution for the relaxed problem
- **Relaxation example**
 - 8 tile puzzle original rule: *a tile can be moved from A to B, if A borders horizontally or vertically with B **and** B is empty*
- **Collection of heuristic functions**
 - admissible heuristic functions $h_1, h_2, \dots, h_m$
 - $h = \max\{h_1, h_2, \dots, h_m\}$ is admissible and dominates over $h_1, \dots, h_m$

Learning Heuristic Functions

- **Experience**

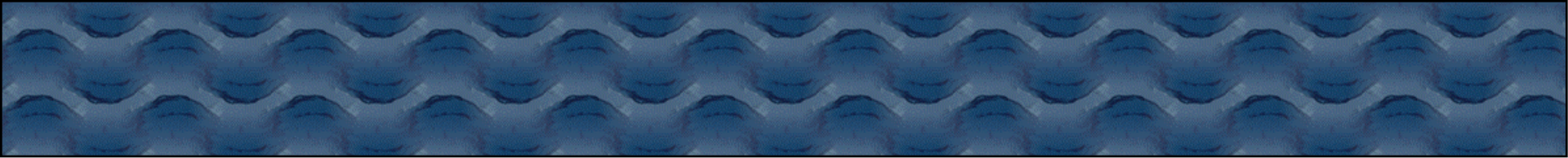
- solve specific problem instances, compute optimal cost
- form a training set $\{ \text{state } n, \text{ cost } h^*(n) \}$

- **Features**

- enhance state description
- e.g. tiles not in place, sum of distances, ...
- enhanced training set $\{ [\text{state } n, \text{ features of } n], \text{ cost } h^*(n) \}$

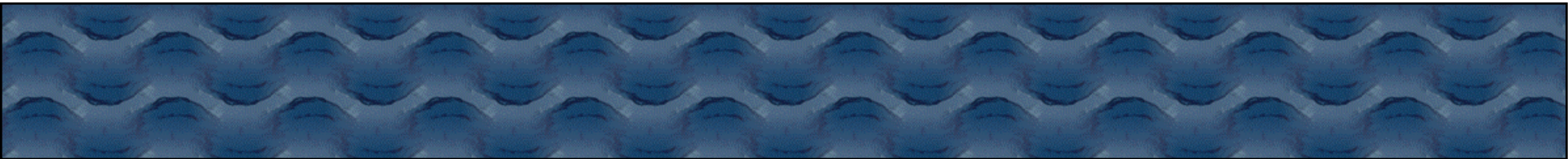
- **Learning**

- supervised learning for generalization over the entire space
- decision trees, SVMs, RVMS, deep neural networks, ...



Local Search

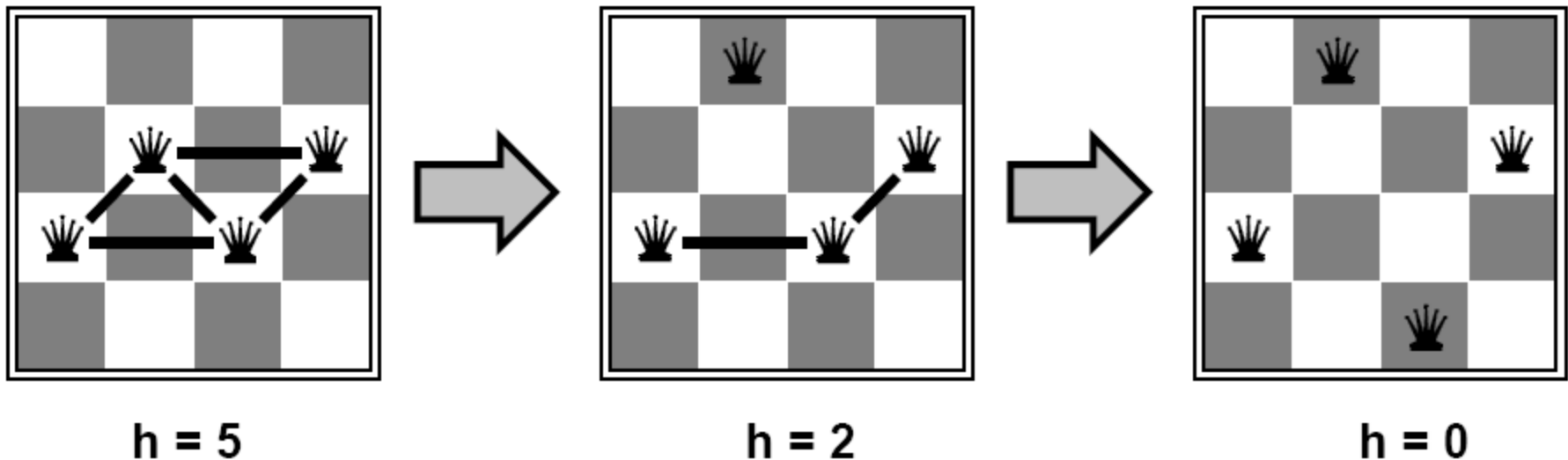
**looking for the goal itself!
not for the path**



Local Search

- **Goal-oriented problems**
 - we care to find the goal, the path to the goal is not important
 - e.g., 8 queens problem, various scheduling tasks, ...
- **Local search algorithms**
 - keep track of the current state only
 - (mostly) move to “neighbouring” states only
 - local search algorithms are not *complete*
 - no guarantees that a solution will be returned, even if it exists
- **Advantages**
 - space-efficient: they use constant amount of memory
 - very good for search in very large or infinite state spaces

An Example: The n-Queens Problem



h = total number of threats (pairs of queens threatening each other)

Optimization

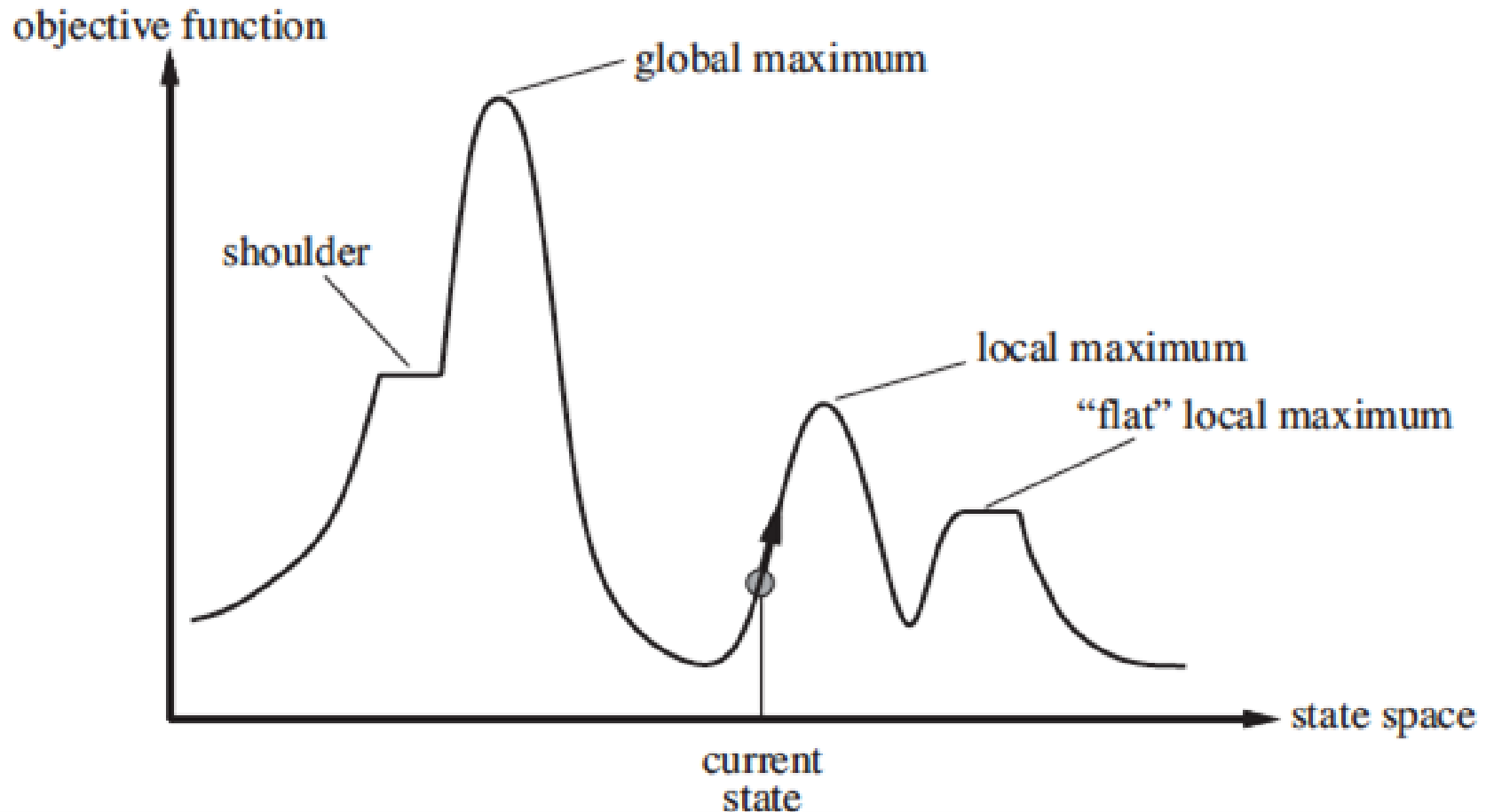
▪ Optimization Problems

- use of an *objective function*
- objective function: maps a state to a number
- goal state: optimizes the objective function
 - optimization: *maximization* or *minimization*
 - just change the sign to switch!

▪ State Space landscape

- position = state ; altitude = objective function value
- extrema: global maximum / minimum
- local extrema: local maximum / minimum
- optimization: find a global maximum / minimum

State Space Landscape



Hill Climbing

- **Algorithm**
 - move to the neighbouring state with the “best value”
 - stop, when no neighbour can improve on current value

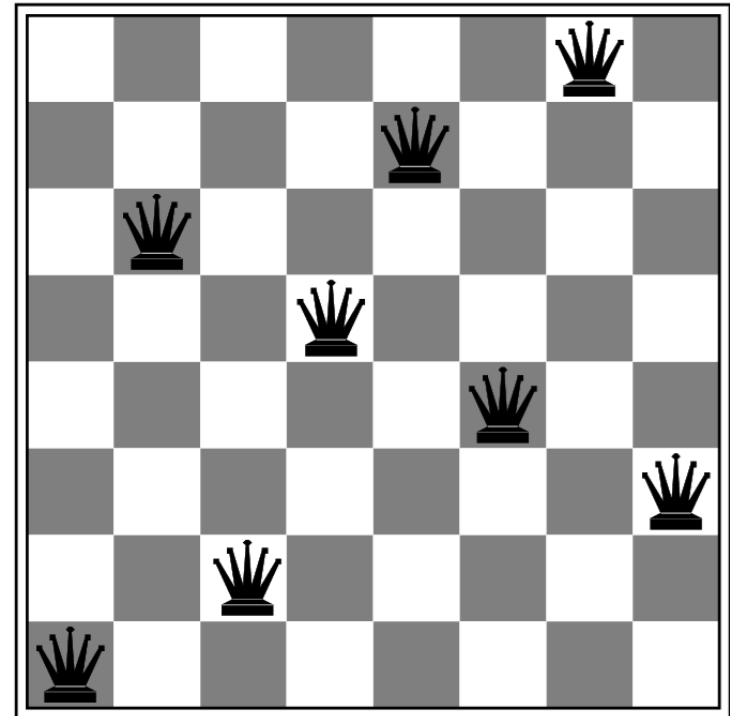
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
  end
```

Hill Climbing – An Example

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

current value $h=17$

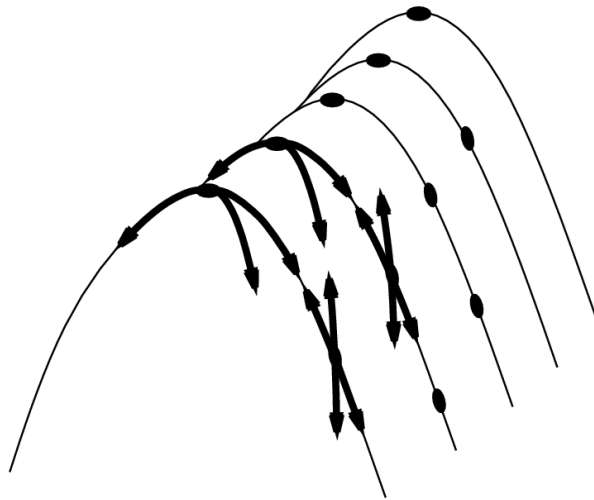


a local minimum with $h=1$

Hill Climbing Problems

□ Challenges

- local maxima
- plateaus
- ridges



□ Treatment

- side moves (up to some limit)
- stochastic search
 - choose among “promising” moves according to some probability distribution (probability of choice : depends on the move’s/state’s value)
 - (when many neighbours) create moves randomly, pick first “good” move
- random restarts
 - a complete algorithm
 - p : probability a search succeeds
 - $1/p$: expected number of restarts

8 queens: 94% success instead of 14% (1.06 restarts instead of 7.14) with side moves!

Simulated Annealing

- **History**

- invented by Nicholas Metropolis in 1953
- based on physics/metallurgy (solid state transition models)
- a variant of stochastic hill climbing

- **Algorithm**

- pre-selects an action randomly (irrespective of value)
- if value better than current, it is accepted as the next move
- if not, it is accepted with probability $p=e^{\Delta E/T}$
 - $\Delta E < 0$ the distance to current value; $T > 0$ a “temperature” parameter
- temperature is decreased progressively with time
- many “sub-optimal” actions initially
- focus progressively moves to “expected-to-be-best” moves
- the decreasing schedule of T is important for optimality

Simulated Annealing

```

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to “temperature”
  local variables: current, a node
                    next, a node
                    T, a “temperature” controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
     $\Delta E \leftarrow \text{VALUE}[\textit{next}] - \text{VALUE}[\textit{current}]$ 
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 

```

Local Beam Search

- **Algorithm**
 - climb hills from different points in-parallel
 - k random initial states are selected
 - k local searches are performed in-parallel
 - at each step: select the k best among all previous step's children
- **Features**
 - funnel resources to most promising areas
 - problem: searches are concentrated in the same sub-area
 - treatment: choose the k children nodes *stochastically*
 - selection probability proportional to the child's value

Genetic Algorithms (GAs)

▪ Algorithm and Glossary

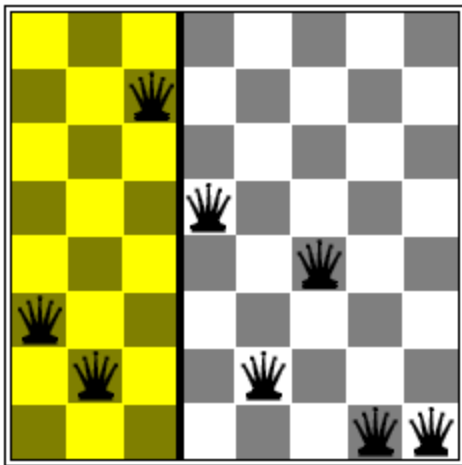
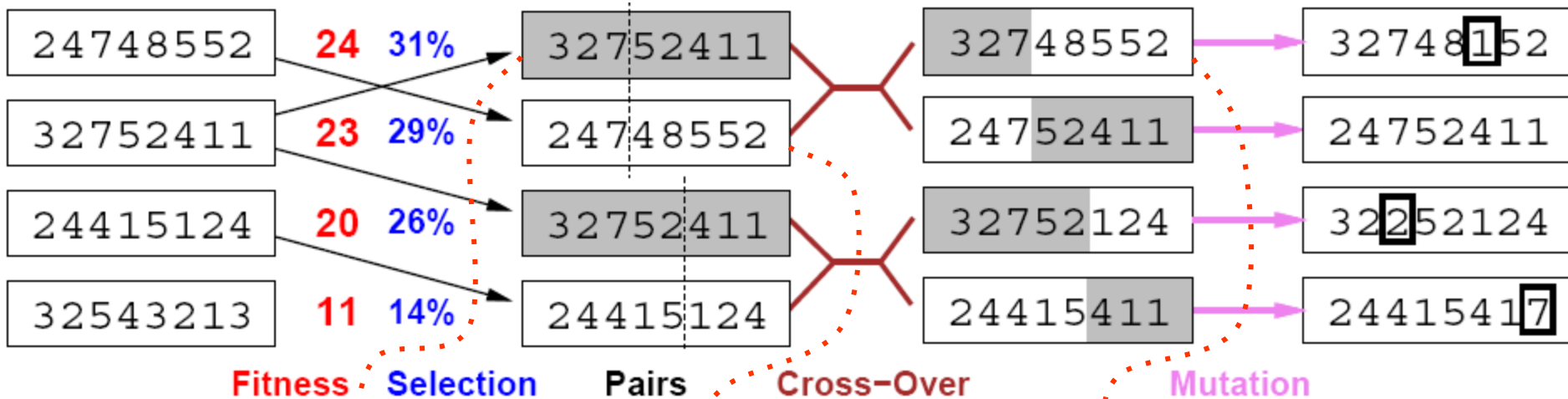
- stochastic beam search + genetic reproduction of successors
- **population**: k **individuals** or states (initially random)
- **chromosomes**: a representation of each state (strings)
- **fitness**: a heuristic function to evaluate individuals
- **selection**: the k “best” individuals survive
- **crossover**: “mate” two strong individuals to give birth to a new one
- **mutation**: a random change in an individual’s chromosome

▪ Features

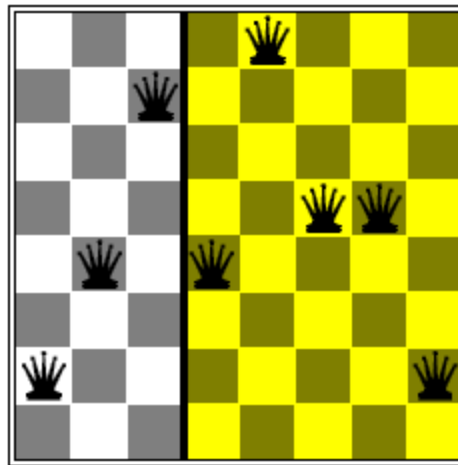
- careful engineering of the representation and other parameters
- 8 queens: position (1-8) of each queen at its column (8 symbols)

GA for the 8-Queens Problem

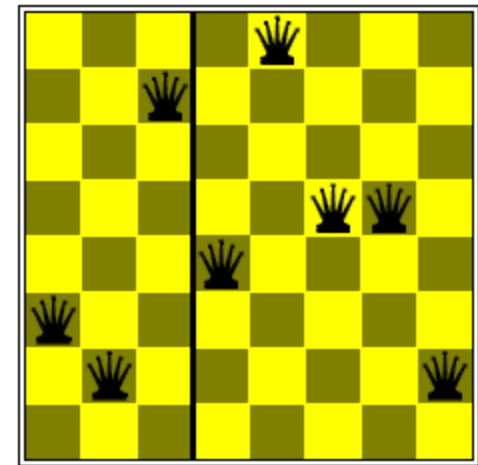
fitness = pairs of queens **not** threatening each other (max value = 28)



+



=



Related Bibliography

- **Artificial Intelligence: A Modern Approach,**
Stuart Russell and Peter Norvig,
4th edition, Prentice Hall, 2020
 - Sections 2.1 – 2.4
 - Sections 3.1 – 3.3

