MC906/MO416 Introduction to AI

Lecture 5

UNICAMP

Introduction to AI

Lecture 5 - Informed and Local Search

Profa. Dra. Esther Luna Colombini
esther@ic.unicamp.br

Prof. Dr. Alexandre Simões
alexandre.simoes@unesp.br

LaRoCS – Laboratory of Robotics and Cognitive Systems

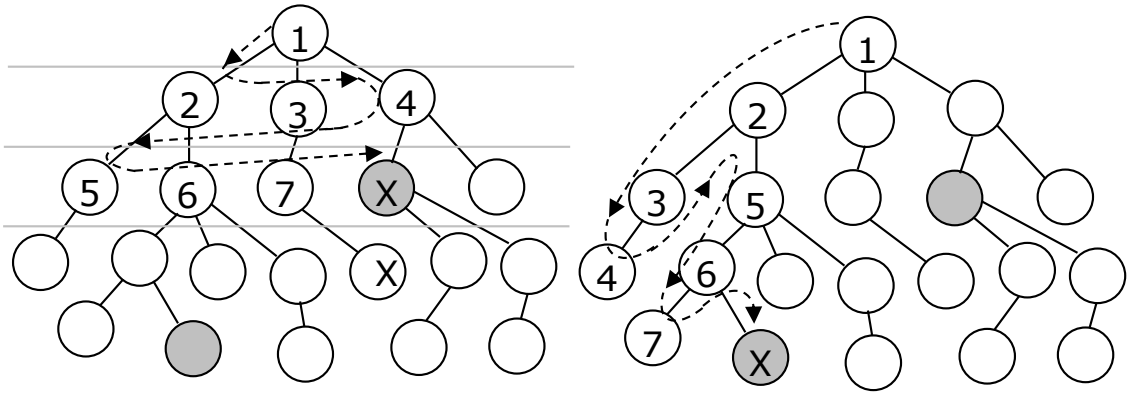
UNICAMP

Advanced Institute for Artificial Intelligence

Summary

- Heuristic
  - Informed search methods
  - Best First Search
  - A*
- Local Search Algorithms
  - Hill climbing search
  - Variations
    - Simulated annealing search
    - Local beam search
- Exercise

- Main search strategies without information: breadth and depth search



- Derived strategies:
  - Search with uniform cost
  - Limited depth search
  - Iterative deepening search
  - Bidirectional search

Information-based search

- How can an agent use knowledge of a specific problem in addition to defining the problem itself to find more efficient solutions?

General strategy

- **Best-First Search**
  - a node is selected for expansion based on an evaluation function f(n) that measures the distance to the objective
  - identical to searching for uniform cost, except it uses **f(x)** instead of **g(x)** to sort the priority queue
- Uses a heuristic function

Heuristics

- Word originated from the Greek "heuriskein" [1] (to discover)
- Most common way to apply additional knowledge of the problem to the search algorithm

- [1] Archimedes is attributed with the passage of shouting "heureka" ("I discovered it!") when he discovered the principle of fluctuation

Heuristics function

- Function that describes how good a state is
- Guides algorithms based on searching for the best choice in the most promising direction, suggesting which way to go
- Mathematically:
  - h(n) = estimated cost of the most economical path from node **n** to an objective node
  - If **n** is objective, then h(n) = 0
- Problem:
  - It is chosen case by case, according to the nature of the problem

Algorithms

- Information search strategies: systematically explore the search space
  - Greedy search for the best choice
  - A*
- Local search algorithms: operate using a single current state
  - Hill climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

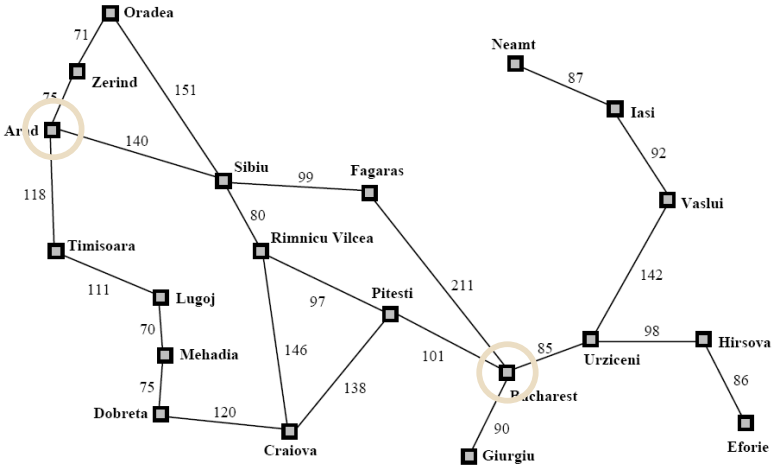
Best First Search (BFS) - Greedy search

- Try to expand the node closest to the goal, assuming that this will lead to a faster solution
- The evaluation of the nodes is done using the function:
  - f(n) = h(n)
  - Example:
    - **Problem**: getting from Arad to Bucharest
    - **Heuristic**: distance in a straight line. h(n) is the straight line distance to Bucharest from each of the evaluated cities.

Greedy search: example

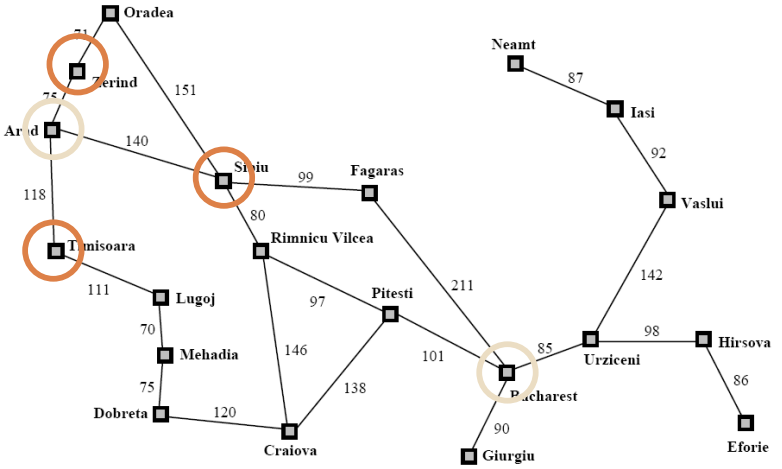
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

(a) The initial state

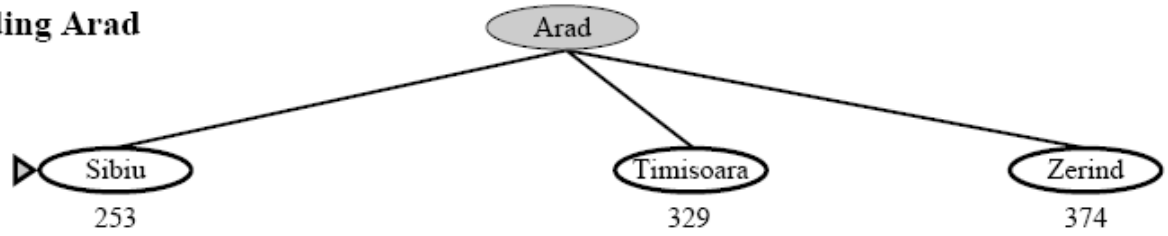


Greedy search: example

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

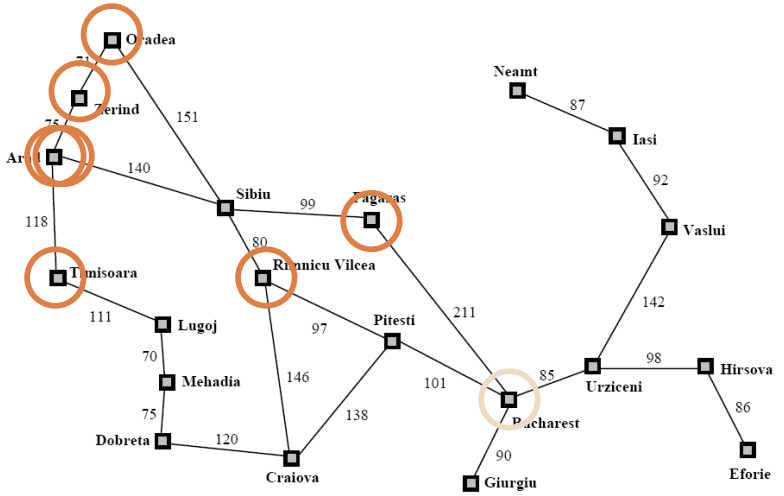


(b) After expanding Arad

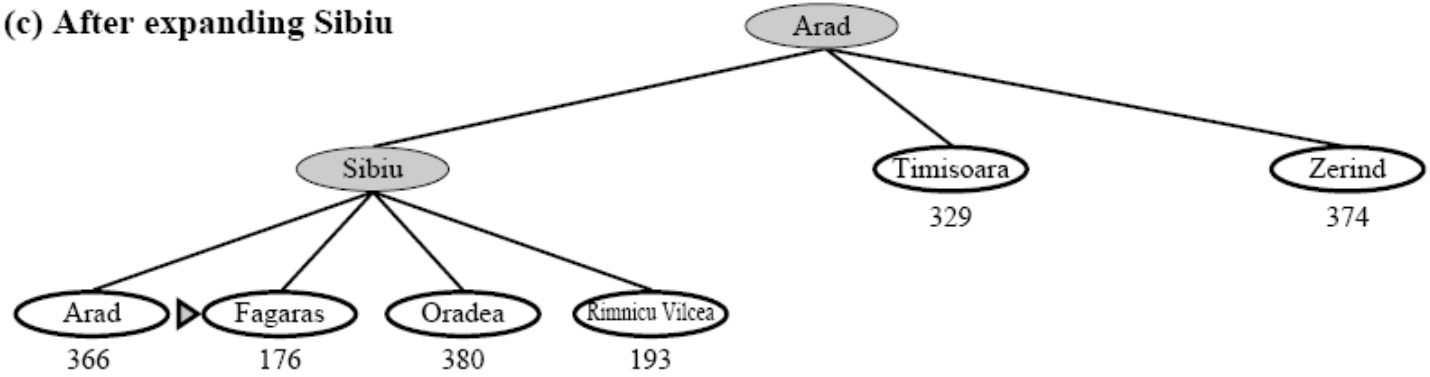


Greedy search: example

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



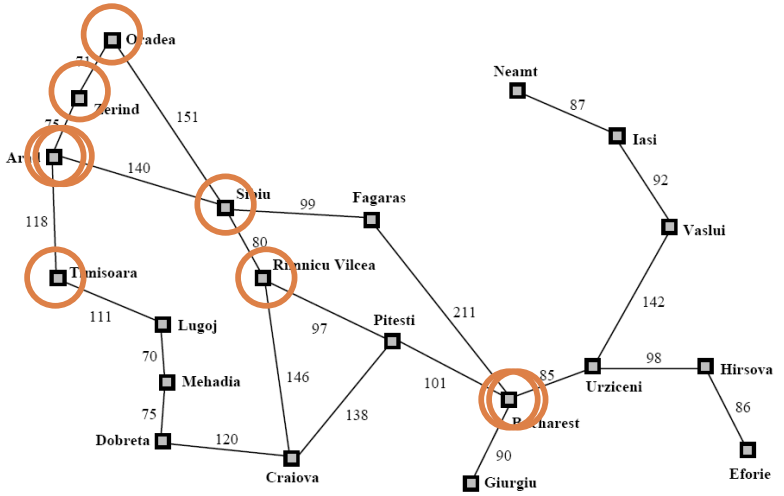
(c) After expanding Sibiu



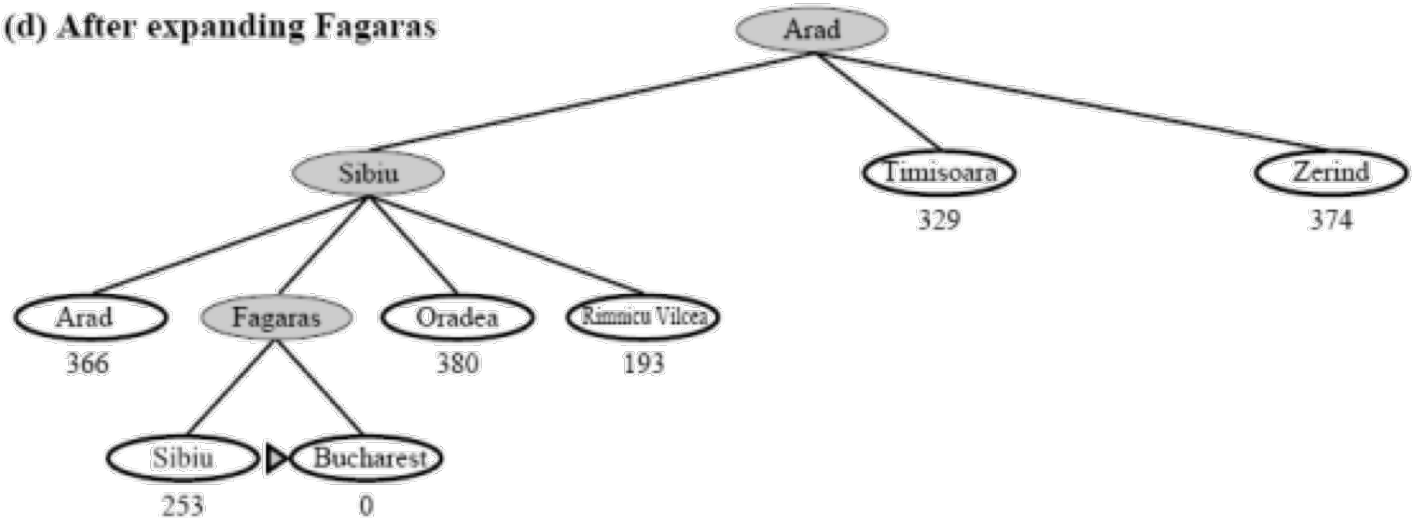
Greedy search: example

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

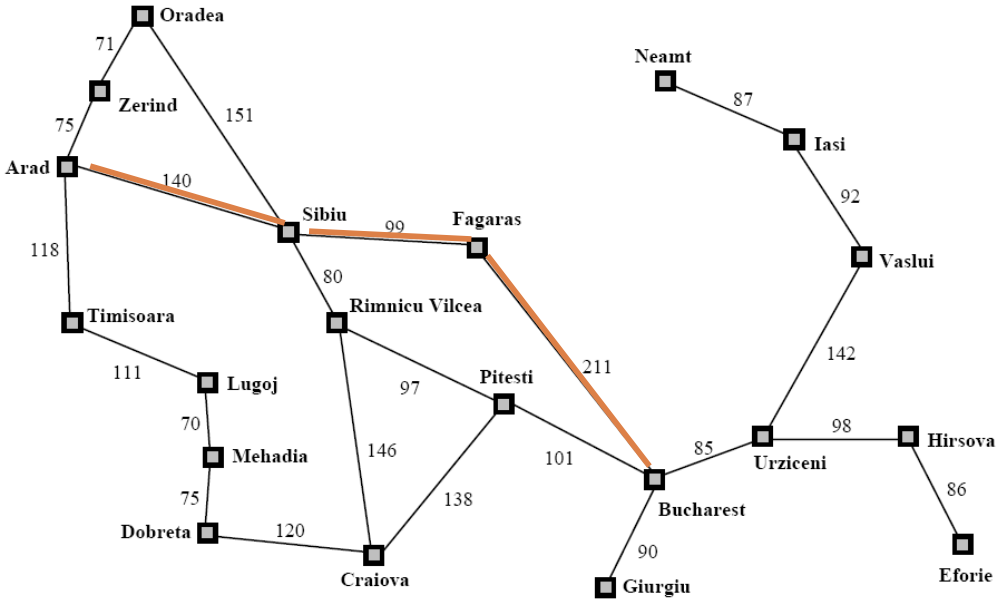


(d) After expanding Fagaras



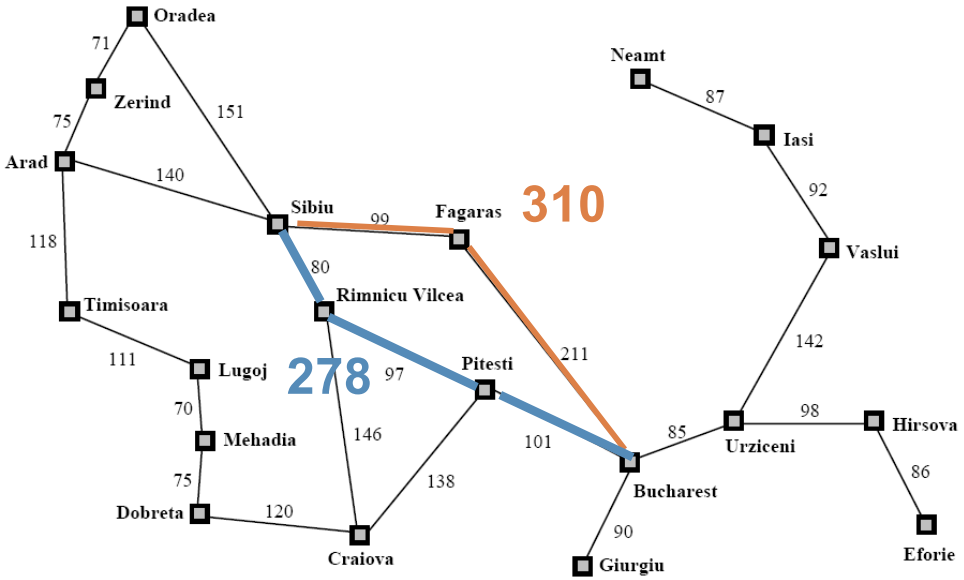
Greedy search: evaluation

- Complexity:
  - It didn't expand any node out of the way of the solution: low search cost!
- Optimization:
  - The path through Sibiu and Fagaras is 32 kilometers longer than through Rimnicu Vilcea and Pitesti.



Greedy search: evaluation

- Complexity:
  - It didn't expand any node out of the way of the solution: low search cost!
- Optimization:
  - The path through Sibiu and Fagaras is 32 kilometers longer than through Rimnicu Vilcea and Pitesti. It's not optimal.



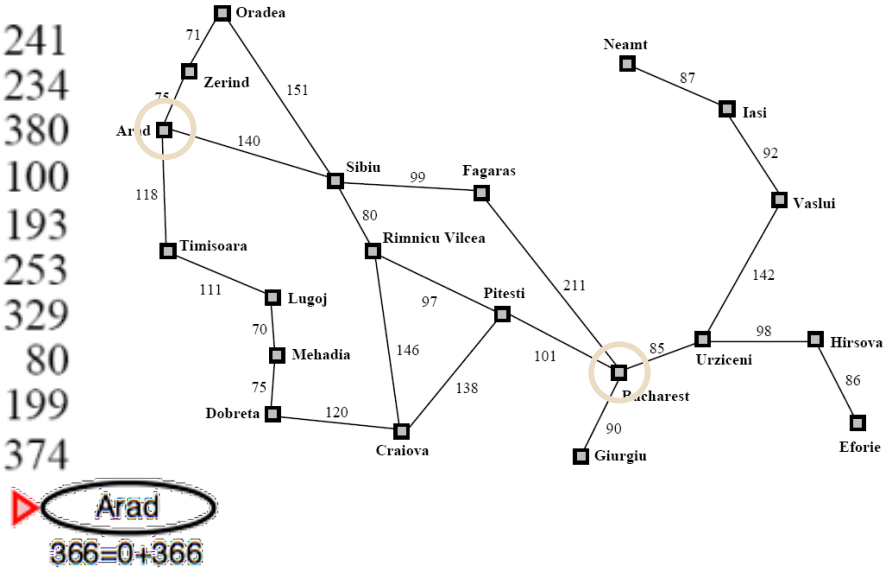
Greedy search: evaluation

- Performance is very dependent on choosing a good heuristic
  - Is it complete?
    - Yes, as long as, like DFS, the algorithm controls access to repeated states (nodes). Otherwise, you can loop.
- Similar to depth-first search:
  - It prefers to follow a single path to the goal, but will return to find a dead end
  - It is not optimal and is incomplete (it can enter an infinite path if you are not careful to detect repeated states)
  - Time and space complexity in the worst case:
    - O(bm). However, a good heuristic function can dramatically reduce this complexity

- Most widely known form of searching for the best choice
- Evaluates the nodes by doing:
  - f(n) = g(n) + h(n)
- **g(n):** cost to reach each node
- **h(n):** cost to get from the node to the goal

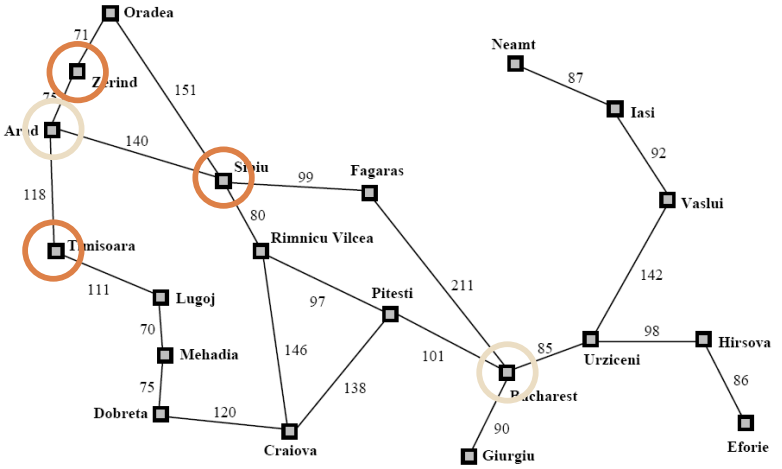
A*: example

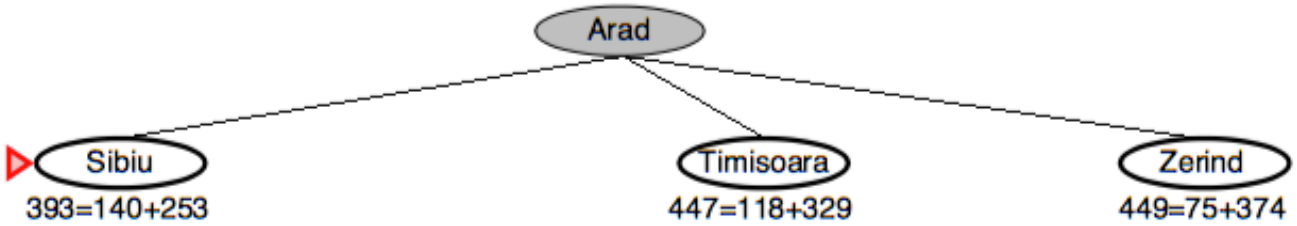
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



A*: example

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

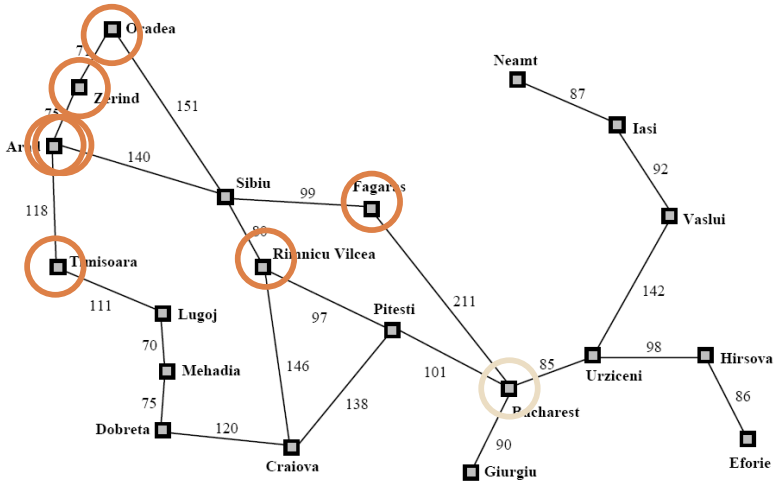


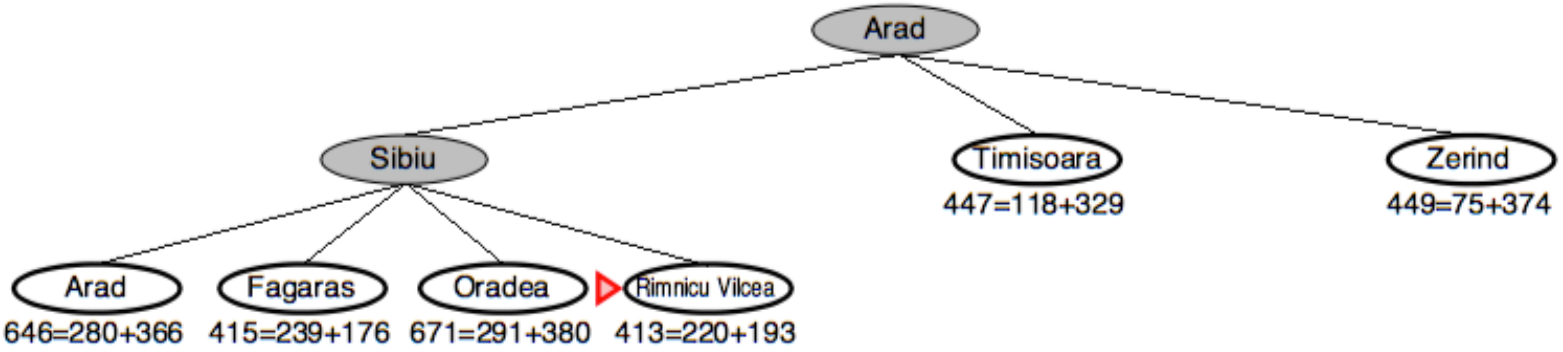


A*: example

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

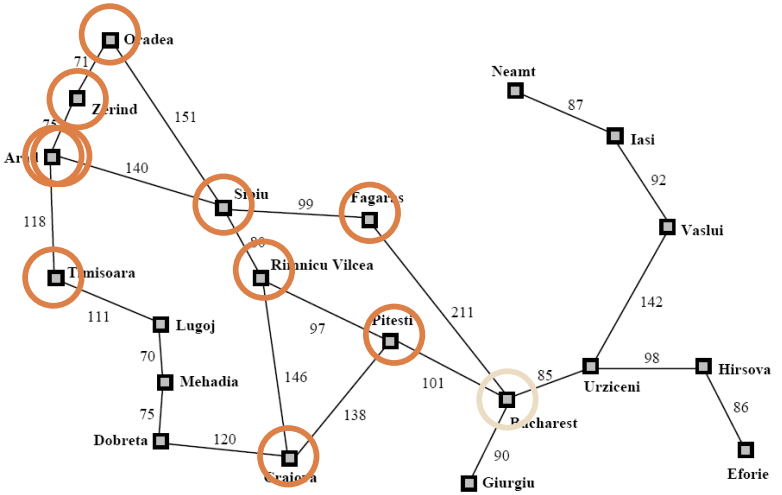


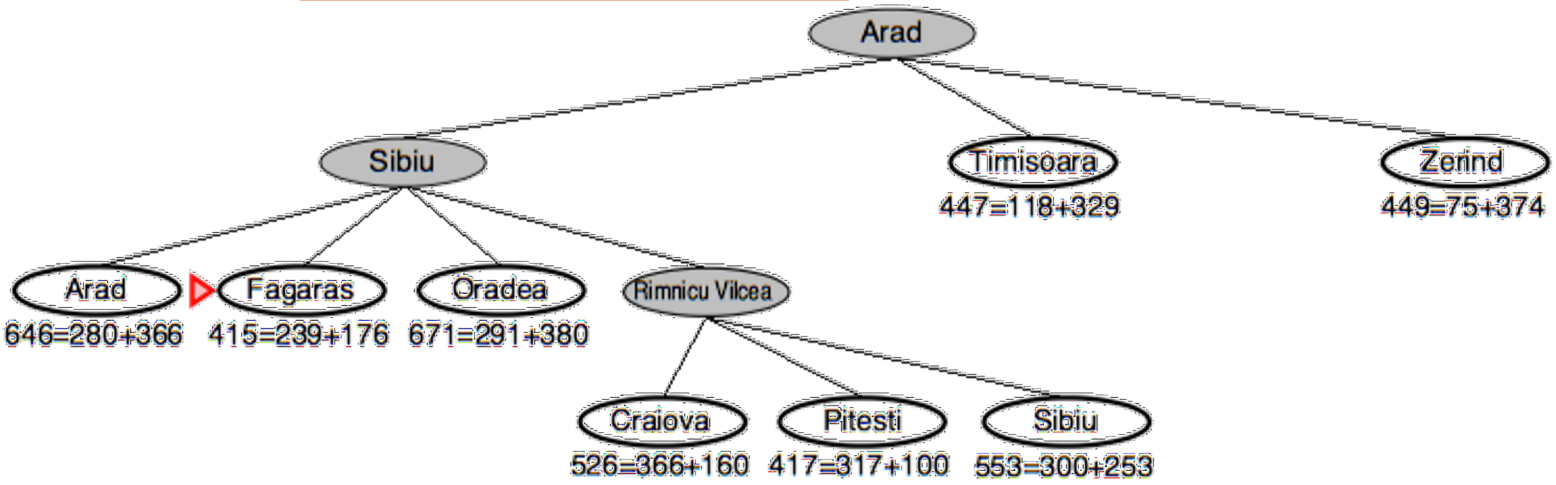


A*: example

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

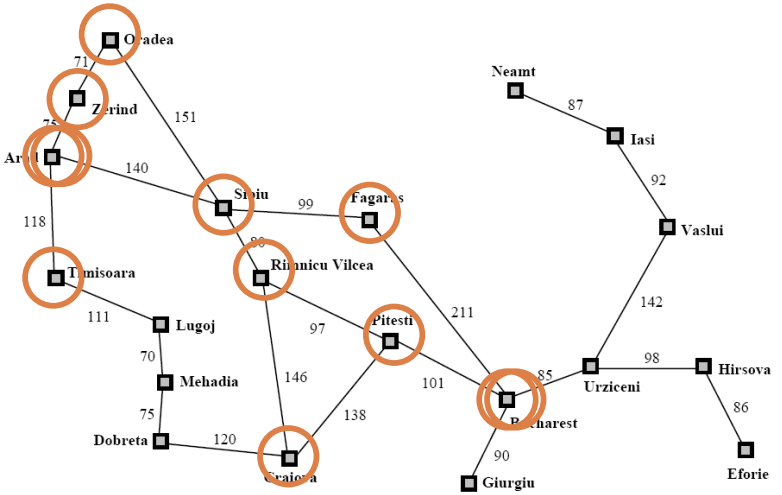
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

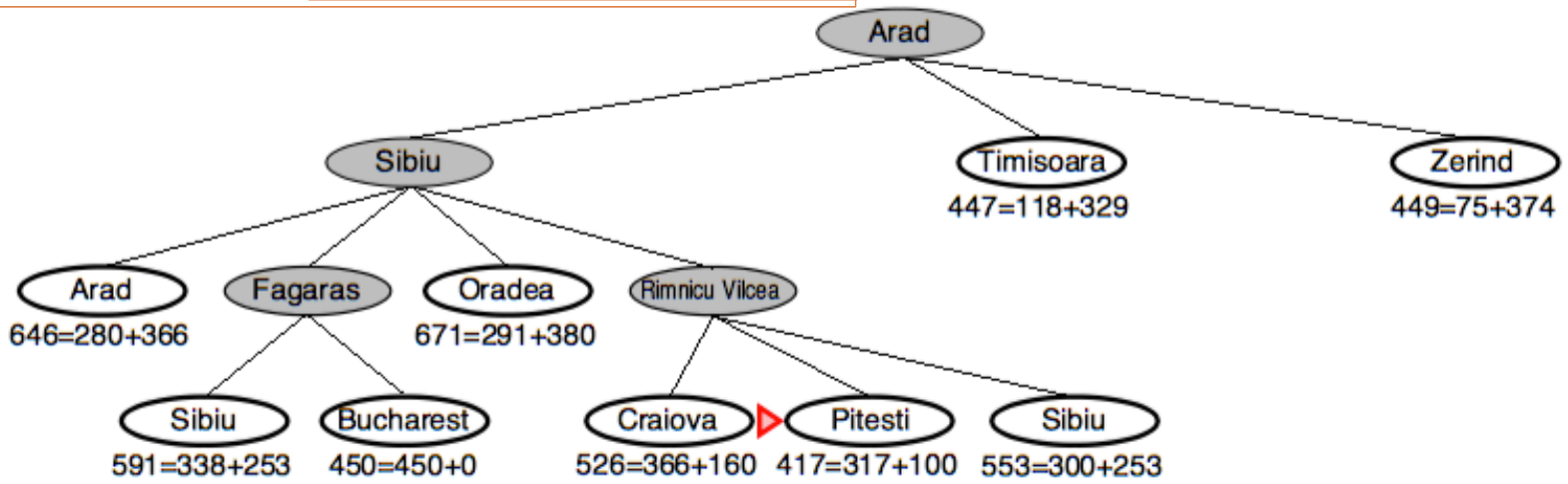




A*: example

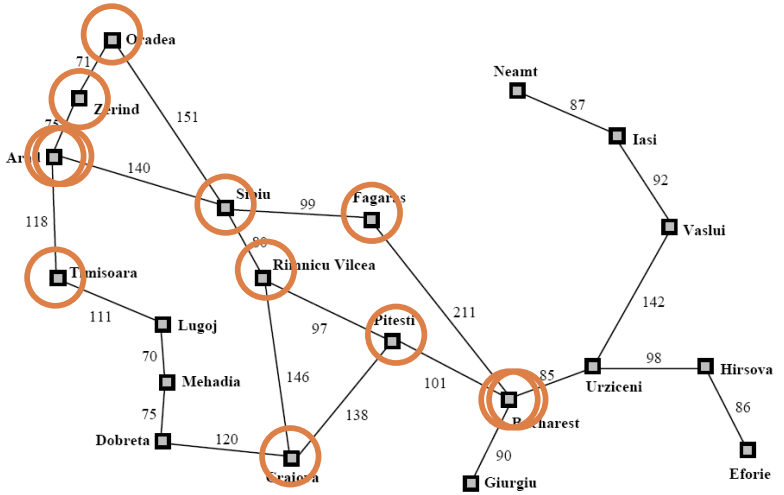
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

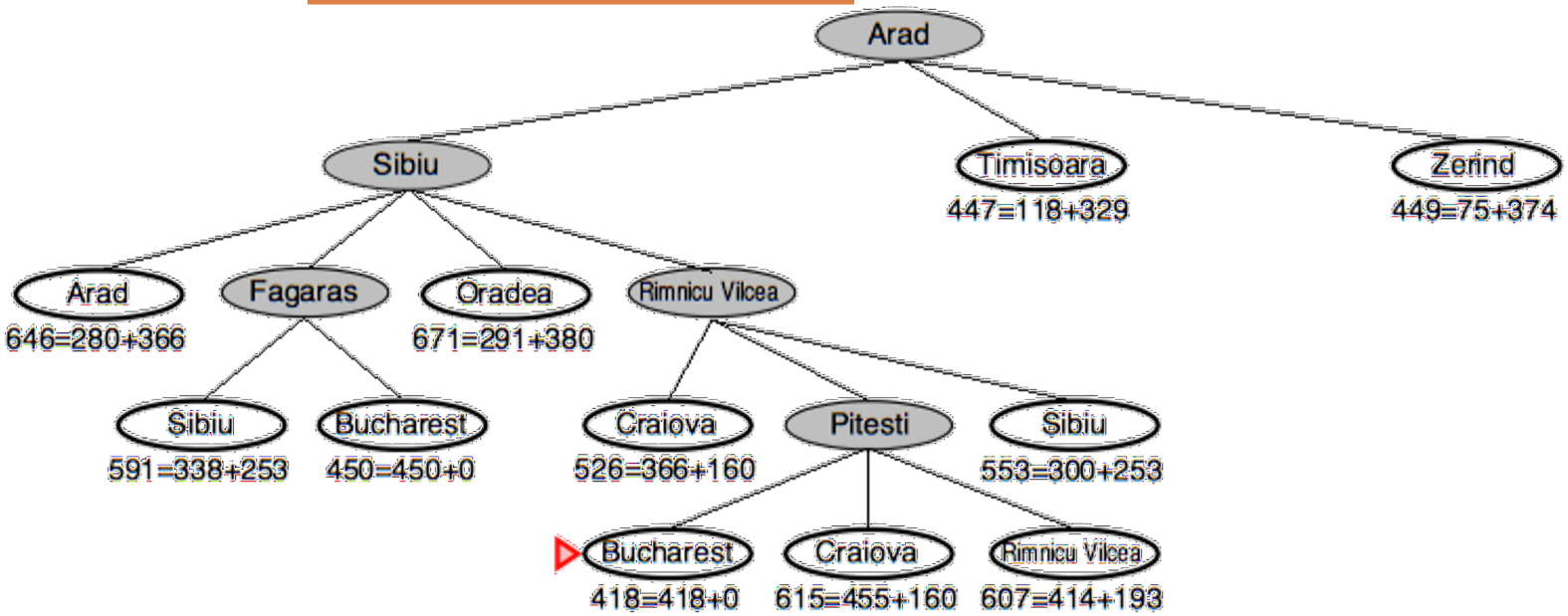




A*: example

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





A*: Evaluation

- To find a low-cost solution, it seems reasonable to try the node with the lowest value of g(n) + h(n) first
- More than reasonable, if the heuristic function h(n) satisfies certain conditions, the A * search will be both complete and optimal ...

Admissible heuristic

- A* is optimal if the search space is a tree and h (n) is an admissible heuristic
- **Admissible heuristics:** never overestimate the cost of reaching the goal. It is an optimistic function.
  - Example:
    - Straight-line distance: the road between two cities can be longer than the straight-line distance, but never shorter (optimistic estimate).
    - Estimated cost h(n) <= actual cost

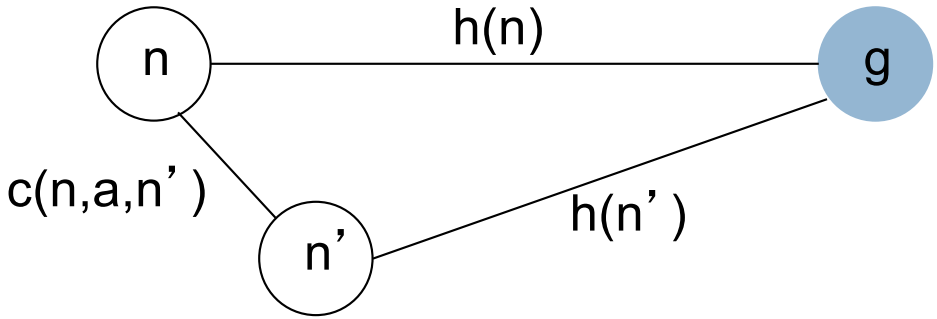
Consistency (monotonicity)

- A* is optimal if the search space is a graph[1] and h (n) is a consistent (or monotonic) heuristic
- A heuristic **h(n)** is consistent (or monotonic) if for every node **n** and every successor **n'** of **n** generated by any action **a**, the estimated cost of reaching the goal from **n** is not greater than the cost of the step of arrive at **n'** plus the estimated cost of reaching the objective from **n'**.

1 graph = tree + list of visited nodes

Consistency (monotonicity)

- For every node n: $h(n) \leq c(n, a, n') + h(n')$



n: node
N': successor of n
g: goal
a: action

- Comments:
  - Any consistent heuristic is permissible
  - Often when h(n) is permissible, it is also consistent

A*: Summary

- **Complete?**
  - Yes
- **Optimal?**
  - Yes, with respect to cost, i.e., it minimizes cost when heuristics are permissible.
- **Custo de Tempo:** in the worst case $O(b^d)$.
- **Custo de Memória:** in the worst case $O(b^d)$.
  - A* expands all nodes with $f(n) < C^*$
  - A* expands some nodes with $f(n) = C^*$
  - A* does not expand any nodes with $f(n) > C^*$

Exercises...

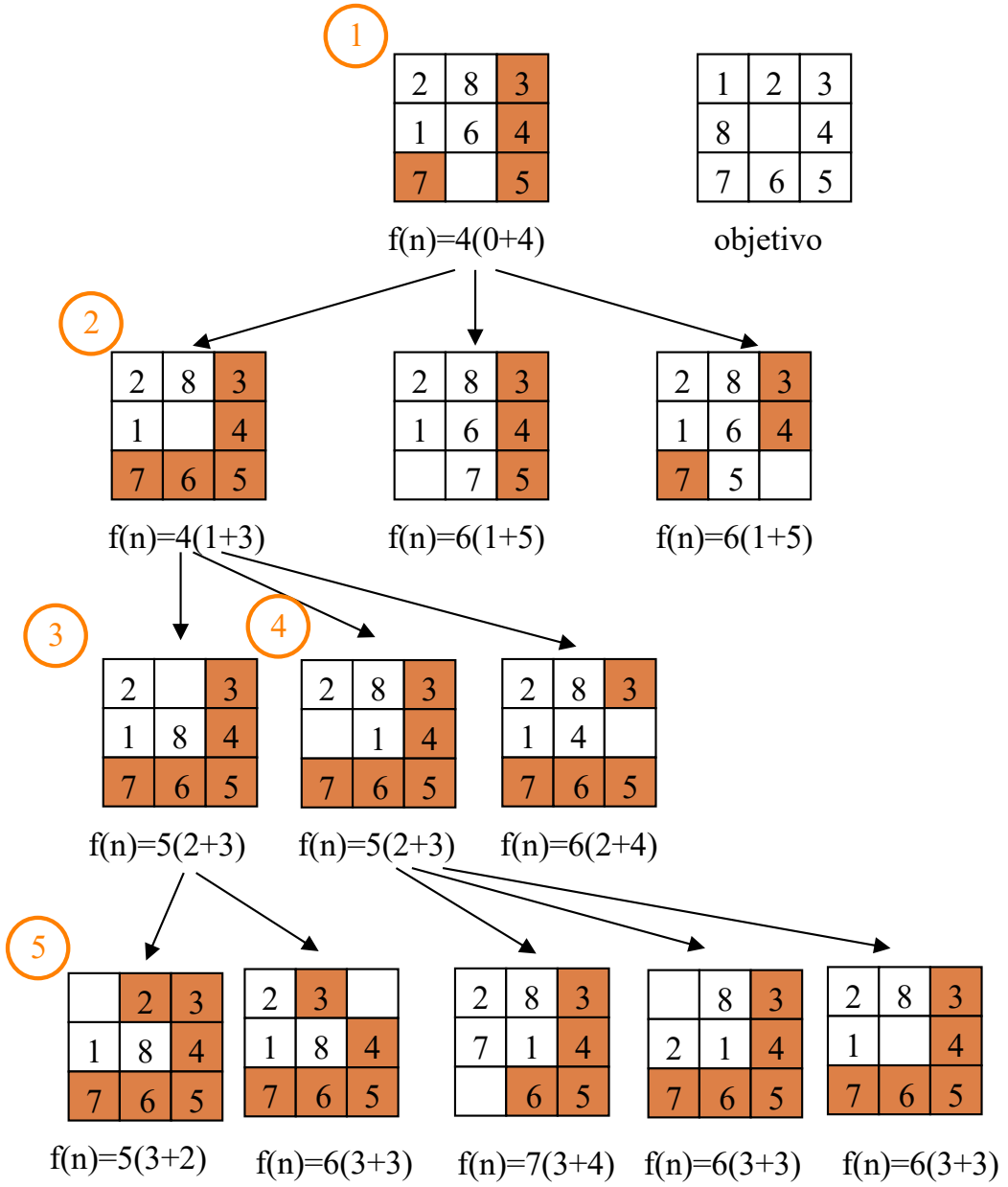
- Apply the A* algorithm to the 8-piece puzzle problem using the following heuristic: “h(n) is the number of blocks in the wrong position”. Assume that the cost to make each move is 1 and the sequence of application of the rules is: up, down, left, right.

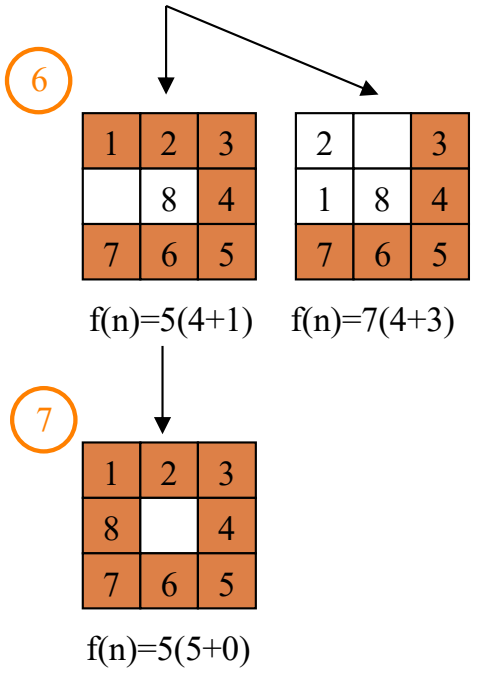
2	8	3
1	6	4
7		5

Start state

1	2	3
8		4
7	6	5

Goal state


1
2 8 3
1 6 4
7 5
f(n)=4(0+4)
1 2 3
8 4
7 6 5
objetivo
2
2 8 3
1 4
7 6 5
f(n)=4(1+3)
2 8 3
1 6 4
7 5
f(n)=6(1+5)
2 8 3
1 6 4
7 5
f(n)=6(1+5)
3
2 3
1 8 4
7 6 5
f(n)=5(2+3)
4
2 8 3
1 4
7 6 5
f(n)=5(2+3)
2 8 3
1 4
7 6 5
f(n)=6(2+4)
5
2 3
1 8 4
7 6 5
f(n)=5(3+2)
2 3
1 8 4
7 6 5
f(n)=6(3+3)
2 8 3
7 1 4
6 5
f(n)=7(3+4)
8 3
2 1 4
7 6 5
f(n)=6(3+3)
2 8 3
1 4
7 6 5
f(n)=6(3+3)


6
1 2 3
8 4
7 6 5
f(n)=5(4+1)
2 3
1 8 4
7 6 5
f(n)=7(4+3)
7
1 2 3
8 4
7 6 5
f(n)=5(5+0)

Exercise 2

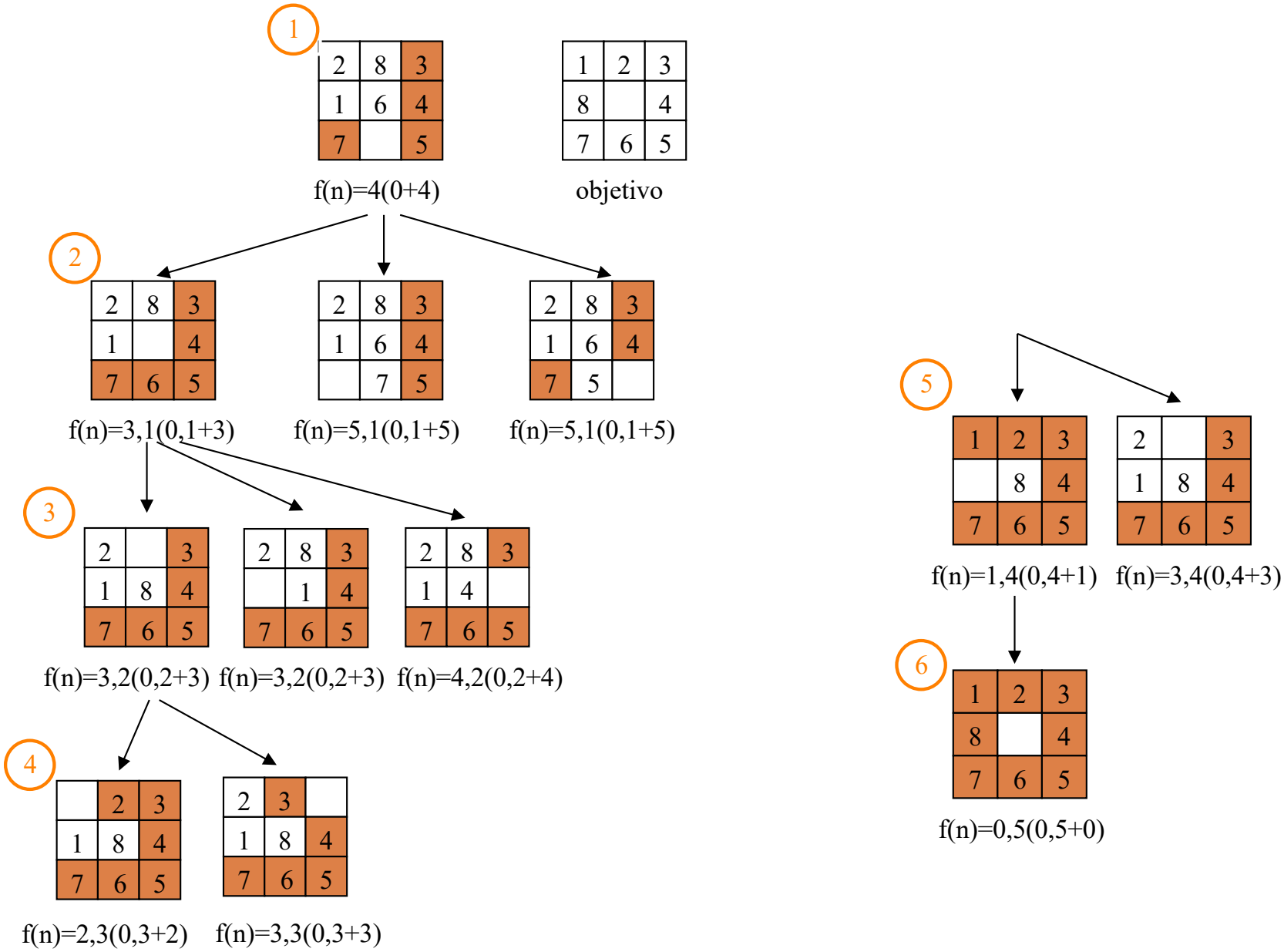
- Apply the A* algorithm to the 8-piece puzzle problem using the following heuristic: "h(n) is the number of blocks in the wrong position". Assume that the cost to make each move is 0.1 and the sequence of application of the rules is: up, down, left, right.

2	8	3
1	6	4
7		5

Start state

1	2	3
8		4
7	6	5

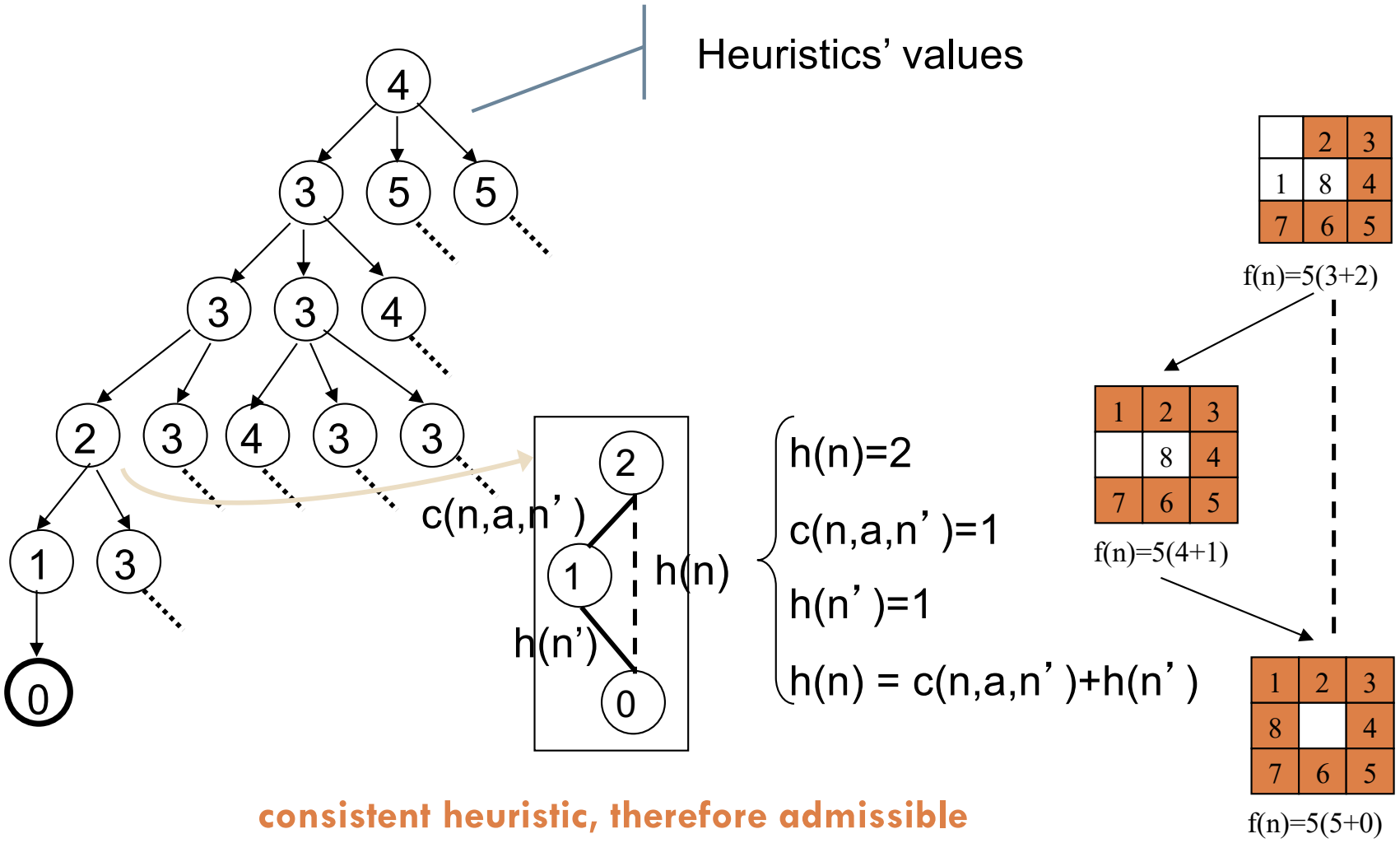
Goal state


1
2 8 3
1 6 4
7 5
f(n)=4(0+4)
1 2 3
8 4
7 6 5
objetivo
2
2 8 3
1 4
7 6 5
f(n)=3,1(0,1+3)
2 8 3
1 6 4
7 5
f(n)=5,1(0,1+5)
2 8 3
1 6 4
7 5
f(n)=5,1(0,1+5)
3
2 3
1 8 4
7 6 5
f(n)=3,2(0,2+3)
2 8 3
1 4
7 6 5
f(n)=3,2(0,2+3)
2 8 3
1 4
7 6 5
f(n)=4,2(0,2+4)
4
2 3
1 8 4
7 6 5
f(n)=2,3(0,3+2)
2 3
1 8 4
7 6 5
f(n)=3,3(0,3+3)
5
1 2 3
8 4
7 6 5
f(n)=1,4(0,4+1)
2 3
1 8 4
7 6 5
f(n)=3,4(0,4+3)
6
1 2 3
8 4
7 6 5
f(n)=0,5(0,5+0)

Exercise 3

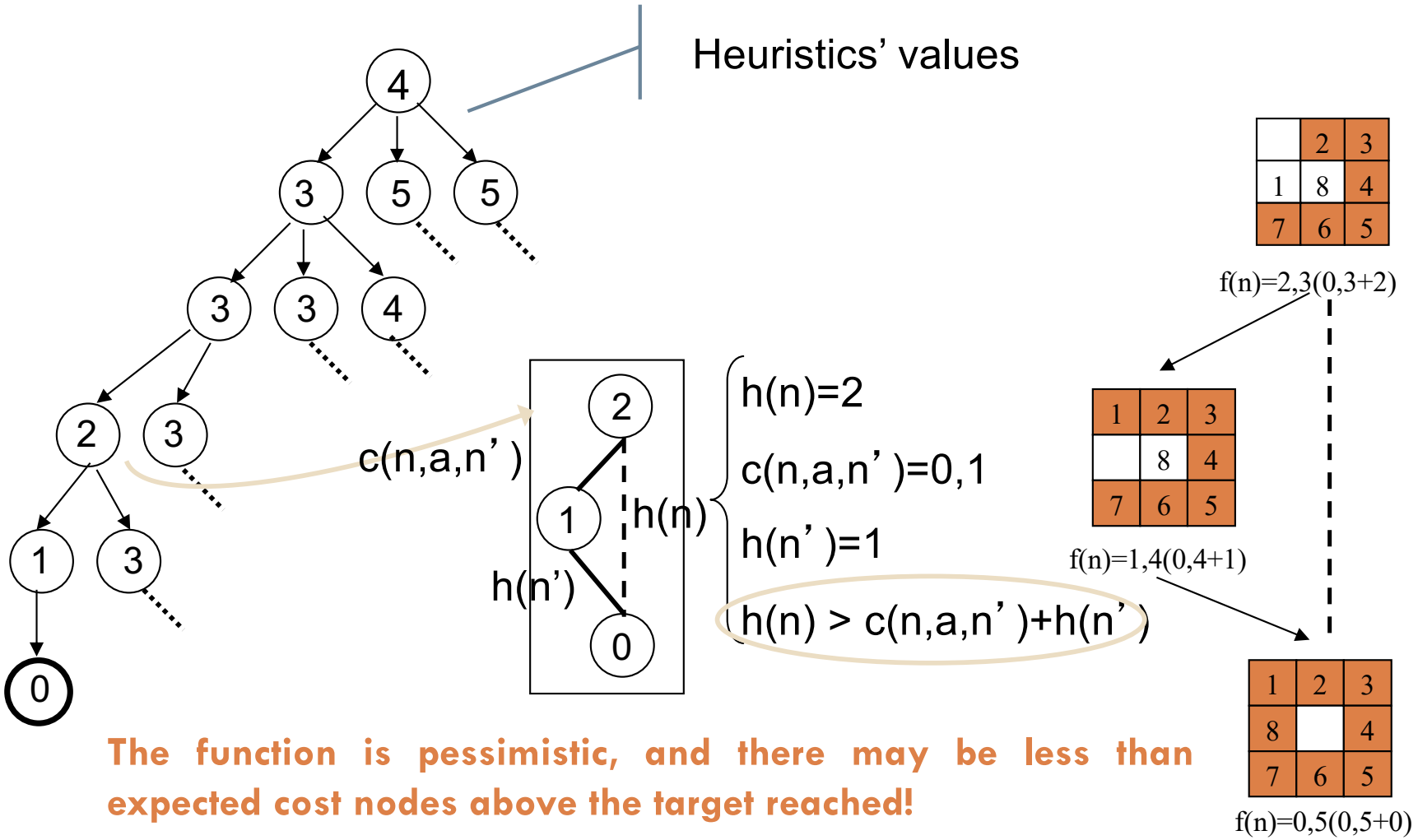
- In the previous two exercises, both found the same answer. However, in only one of them can we guarantee that the response is optimal.
  - Which one?
  - Why?

Exercise 1: heuristics



consistent heuristic, therefore admissible

Exercise 2: heuristics



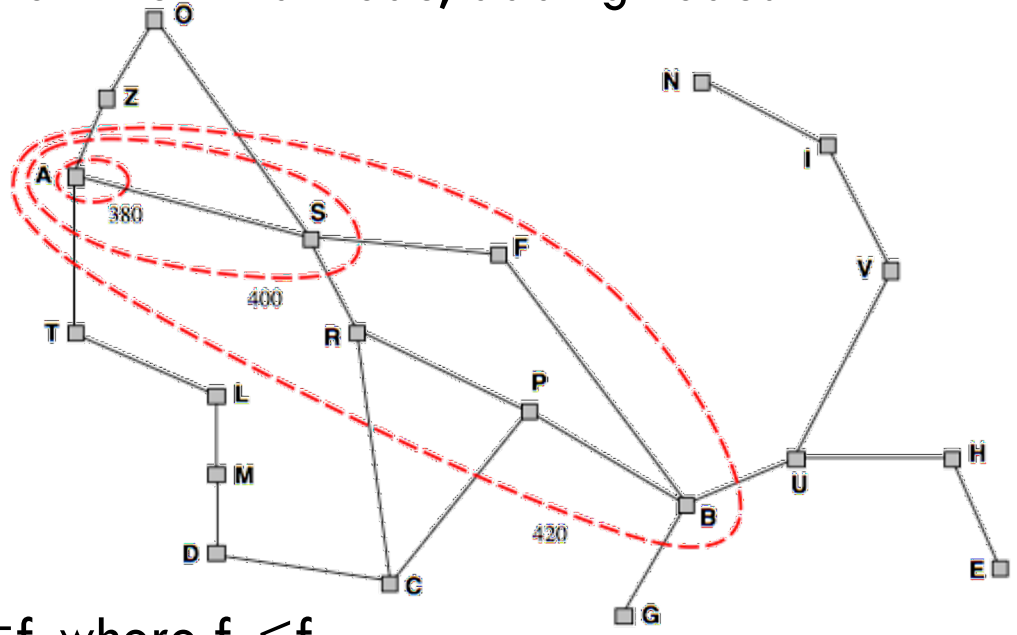
The function is pessimistic, and there may be less than expected cost nodes above the target reached!

Consistency: considerations

- If the function is consistent, then the values of f along any path are non-decreasing
- If the sequence of nodes expanded by A* is in non-decreasing order, then the first objective node selected for expansion must be the optimal solution, as all subsequent nodes will be at least as expensive as it

Costs such as contours

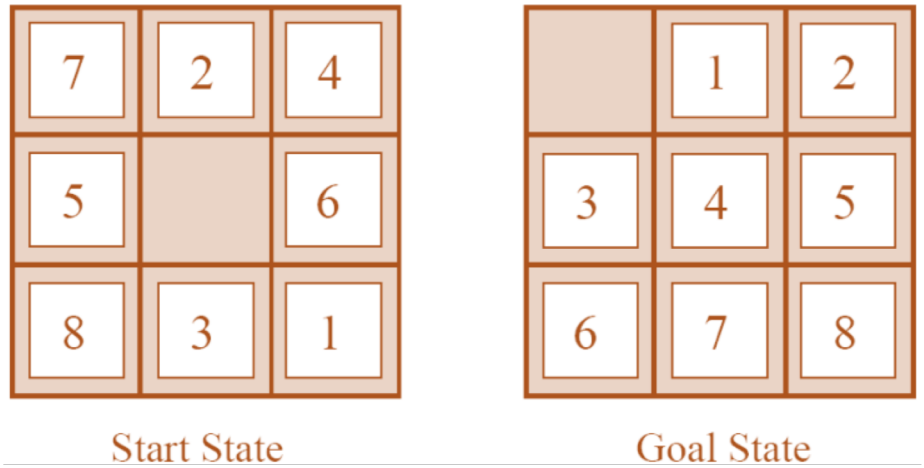
- If **f** is consistent, costs have outlines in the state space, similar to a topographic map
- In the search for uniform cost (A* using h(n) = 0) the contour will be circular around the initial state
- With more accurate heuristics, the contours will stretch towards the objective state
- Considering that A* expands the edge node that has the lowest cost of f, we can verify that an A* search diverges from the initial node, adding nodes in increasing cost ranges.



- Contour i contains all nodes with $f=f_i$ where $f_i < f_{i+1}$

- Iterative deepening A*
- The cut used is the cost of f(g + h) instead of the depth
- At each iteration the cutoff value is the lowest cost of f of any node that has exceeded the cutoff in the previous iteration
- Avoids substantial overhead associated with maintaining an ordered queue of nodes

Heuristic functions



- Cost of the average solution generated: 22 steps
- Branch factor $\cong 3$
- Number of states examined in a search without information: $3^{22} = 3.1 \times 10^{10}$ states

- Heuristics used in the eight-piece puzzle:
  - **h1**: the number of blocks in the wrong positions. For the initial state, h1 = 8.
  - h2: the sum of the distances (horizontal and vertical: Manhattan distance) of the blocks from their objective positions. For the initial state: $h_2 = 3+1+2+2+2+3+3+2 = 18$

Manhattan distance: $|x_1 - x_2| + |y_1 - y_2|$

Heuristic functions: evaluation

- Average number of nodes expanded using A* algorithm with h1 and h2 and search for iterative depth (BAI) for 1,200 randomly generated problems

d	BAI	A* (h1)	A* (h2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	3644035	227	73
14	-	539	113
16	-	1301	211
18	-	3056	363
20	-	7276	676
22	-	18094	1219

Local search algorithms

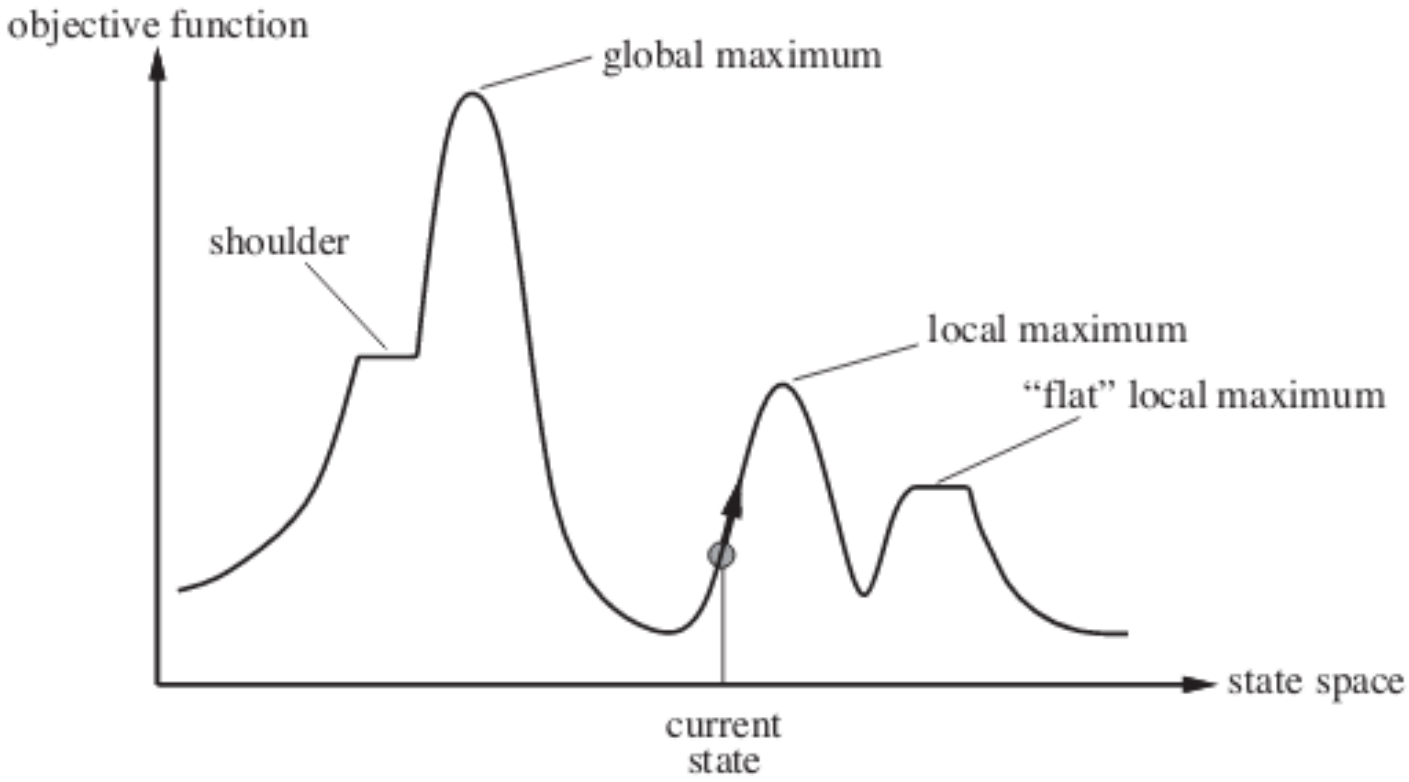
- Search algorithms
  - They systematically explore search spaces. They keep one or more paths in memory by recording the alternatives that have been explored. When an objective is found, the path to that objective is also a solution to the problem
- Local search algorithms
  - Class of algorithms that can be used if the path to the solution is not important

Local search algorithms

- They use a single state (current state) and generally move only to the neighbors of that state
- Objective: find a state according to an objective function. f(n) = h(n)
- Benefits:
  - Use very little memory
  - They can find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable

Optimization issues

- The goal is to find the best state according to an objective function



Hill climbing search

- Repetitive loop that moves continuously towards the increasing value, that is, uphill

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

input: *problem*, a problem

local variables: current_*node* (a node) and *neighbor* (a node)

current_node ← CREATE-NODE(START-STATE[*problem*])

repeat

neighbor ← a current successor with higher value

if VALUE[*neighbor*] ≤ VALUE[current_node] **the return** STATE[current_node]

current_node ← *neighbor*

Hill climbing search

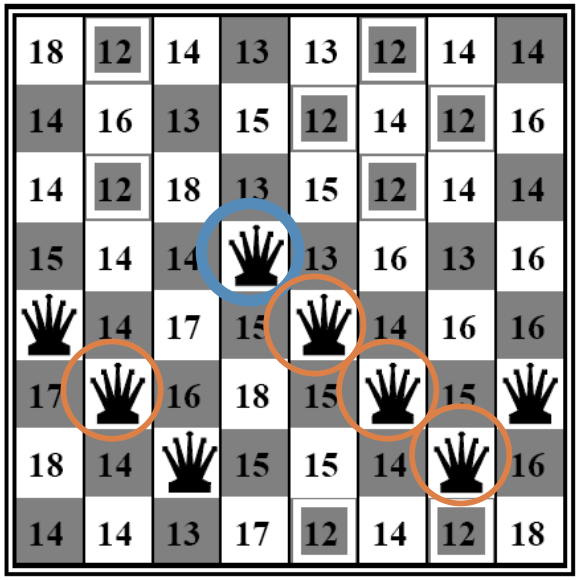
- The first child who improves the father's worth will be chosen
  - In the event of a tie, the draw is generally drawn

Hill climbing: example

- Problem: 8 queens
  - Each state has 8 queens on the board, 1 per column
  - Successor function returns all possible states generated by moving a single queen to another square in the same column (each state has 8x7 = 56 successors)
  - The cost function of the heuristic h is the number of queens that are attacking each other
  - Solutions have h = 0

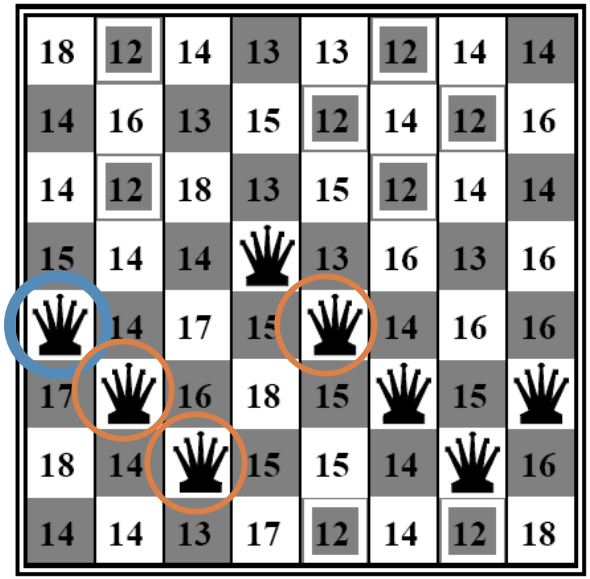
Hill climbing: computing h

- Values of h for successors of the current state



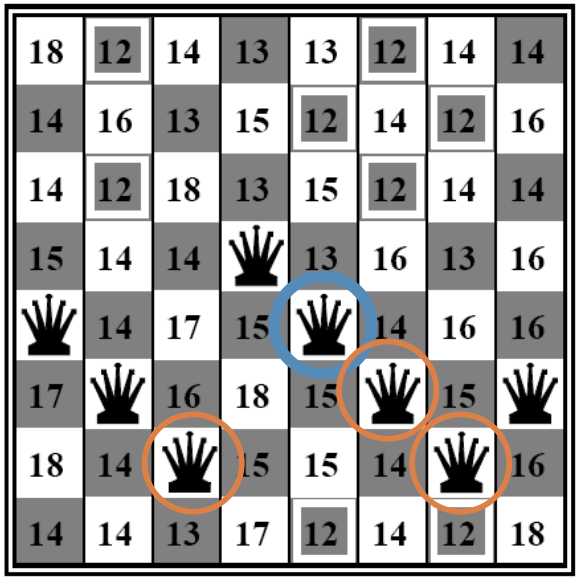
- h=4

Hill climbing: computing h



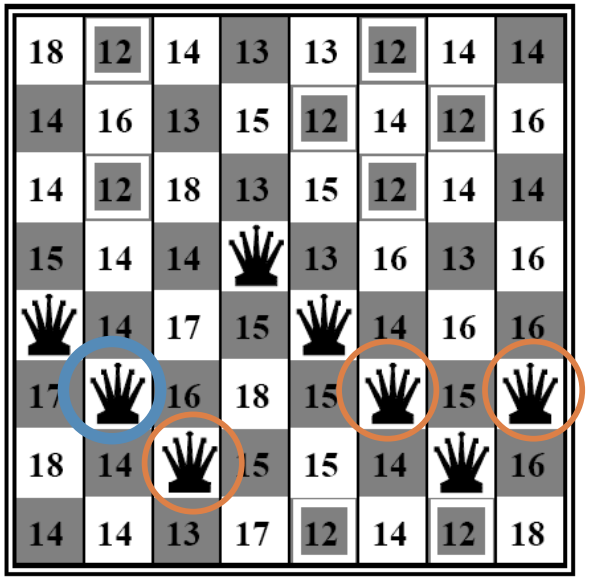
- h=4+3

Hill climbing: computing h



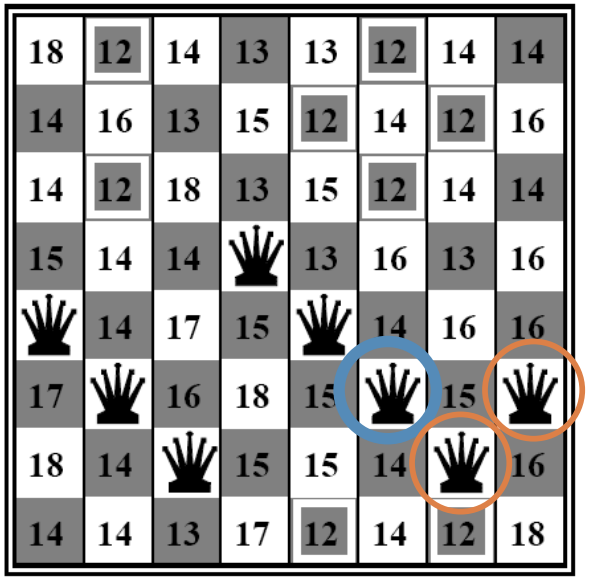
- h=4+3+3

Hill climbing: computing h



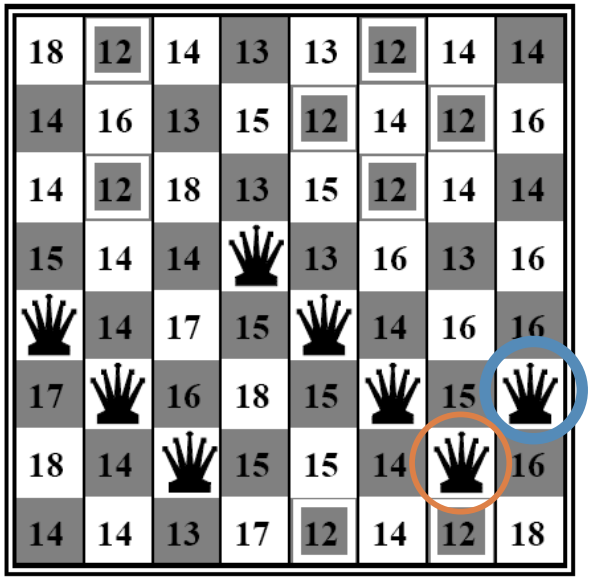
- h=4+3+3+3

Hill climbing: computing h



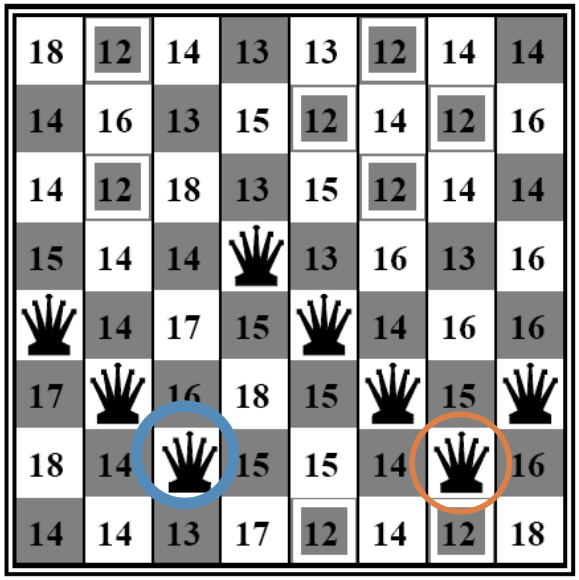
- h=4+3+3+3+2

Hill climbing: computing h



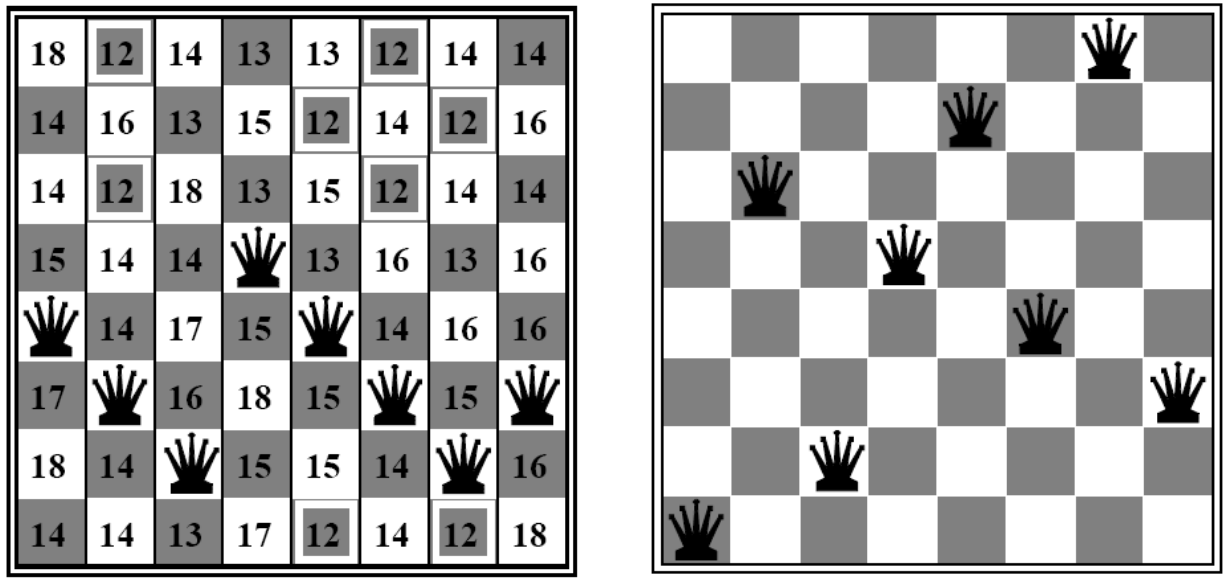
- h=4+3+3+3+2+1

Hill climbing: computing h



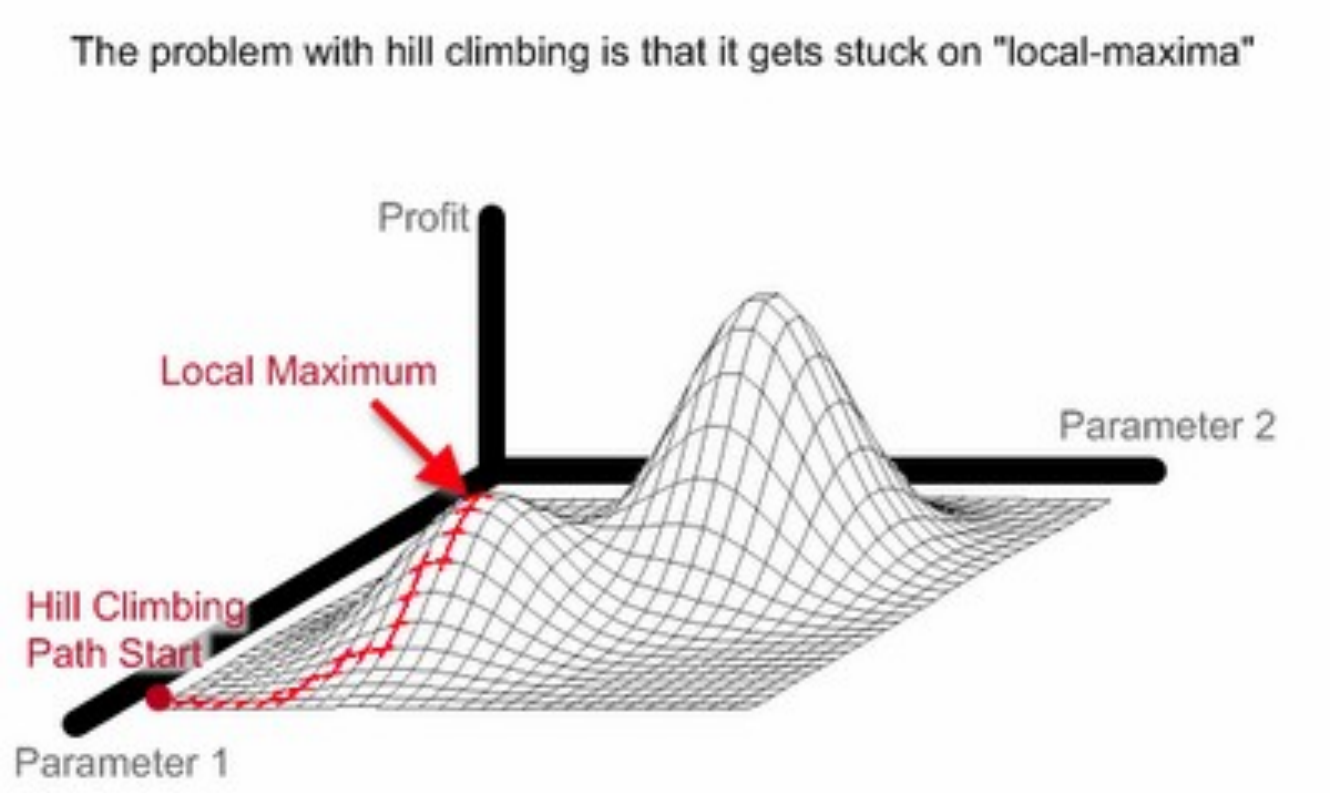
- h=4+3+3+3+2+1+1 = 17

Hill climbing: local minima



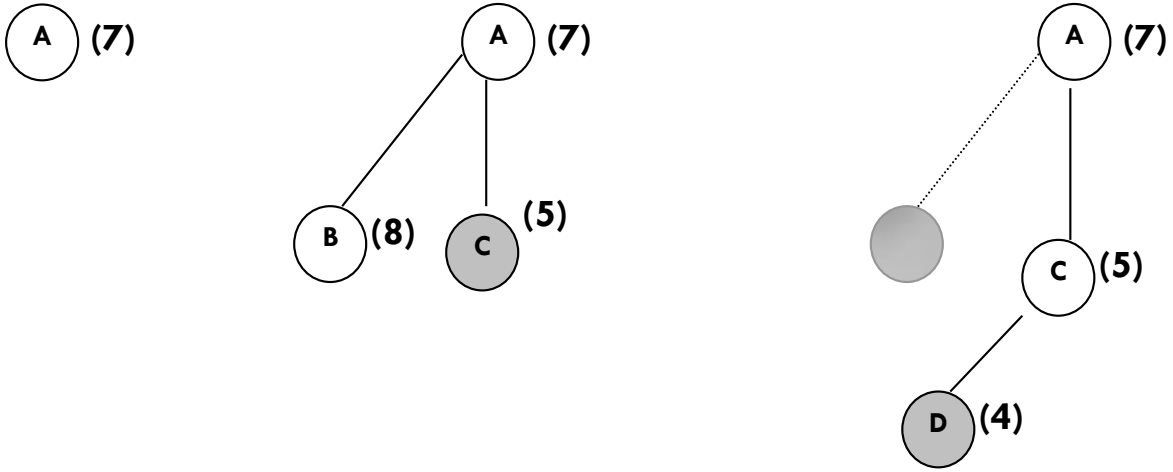
- Left: State with h = 17, showing the value of h for each possible successor obtained by moving a queen within her column. Marked houses are the best move
- Right: Local minimum: the state has h = 1, but every successor has a higher cost

Hill climbing: local minimum / local maximum



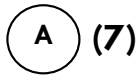
Hill climbing: evaluation

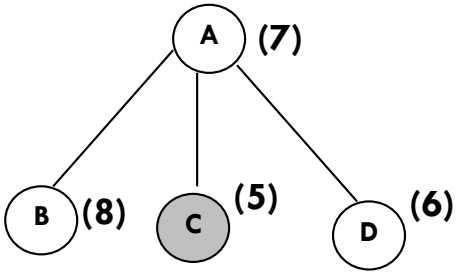
- Complete?
  - Not
- Optimal?
  - Not
- Time Cost:
  - best case: O(d)
  - worst case: $O(b^d)$
- Memory Cost:
  - nodes are not kept. Uses little memory

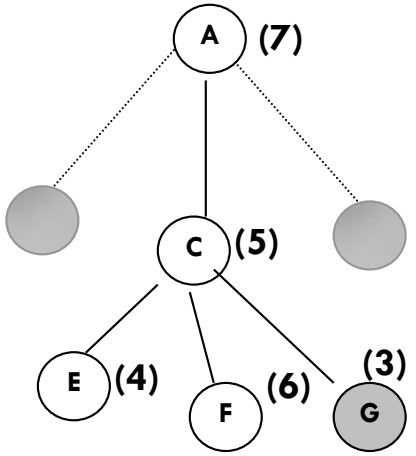


Hill climbing by the steepest path

- Complete?
  - Not
- Optimal?
  - Not
- Time Cost:
  - best case: O(d)
  - worst case: $O(b^d)$
- Memory Cost:
  - nodes are not kept. Uses little memory







Hill climbing: variations

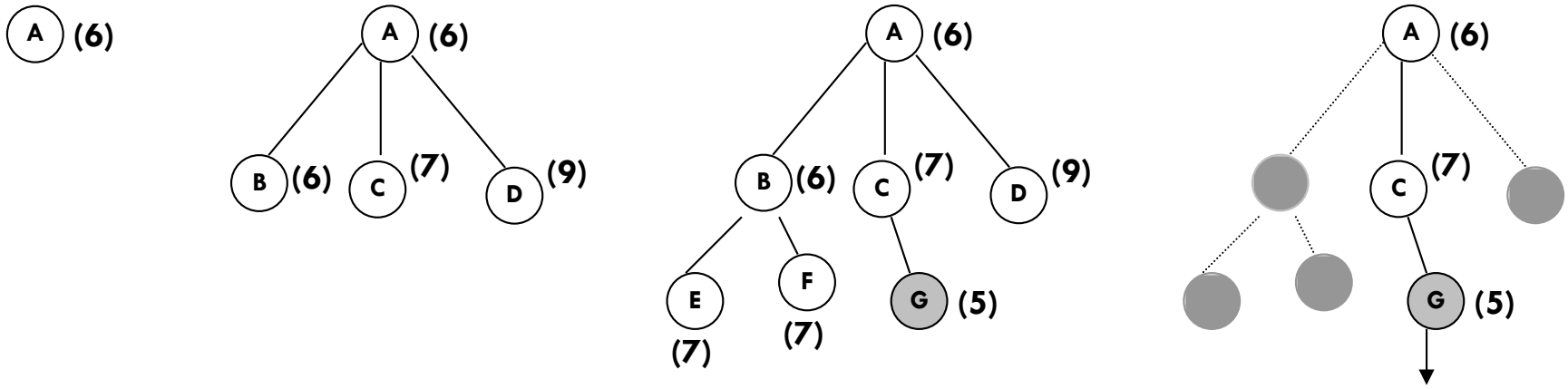
- Algorithm with a large number of variations:
  - **Stochastic Hill Climbing**: choose at random between the movements up the hill
  - **First Choice Hill Climbing**: stochastic hill climbing generating successors at random until generating a successor better than the current state
  - **Hill Climbing with random restart**: series of hill climbing searches based on randomly generated initial states
- ...

Hill climbing: problems

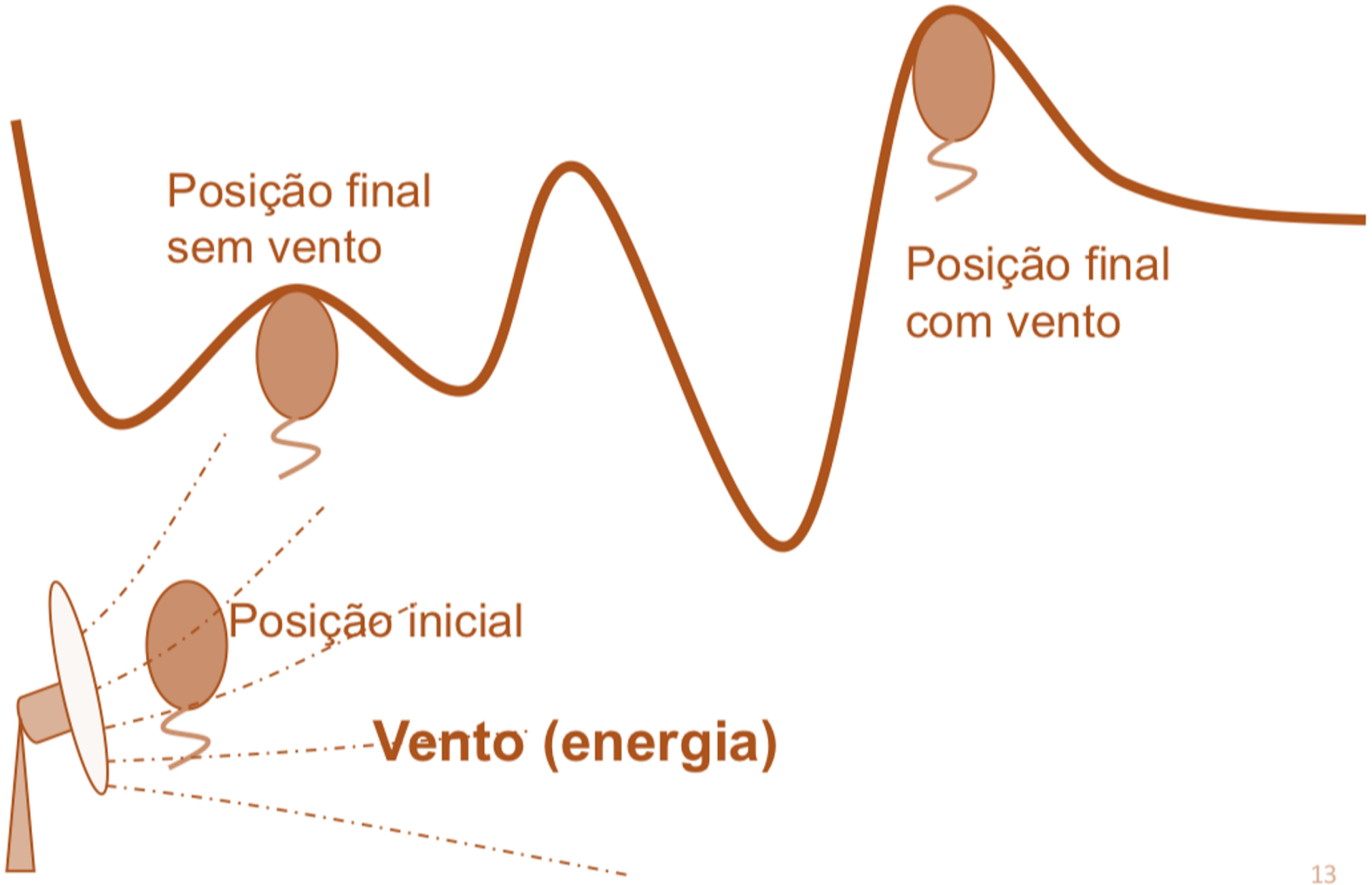
- The pitfalls:
  - They happen because Climb-up does not backtrack
  - Depend on initial state
- Ways to get around such traps:
  - Start over with another operator
  - Apply a search engine merge
  - Enforced Hill-Climbing - Forced Hill Climbing
  - Allow the search to go down a few steps from the slope (Simulated Annealing)

Forced Hill Climbing

- It was initially used in the planning system
- It aims to continue looking (expanding) the successor nodes of the current state, with BREADTH-FIRST, until it finds a node that is better than the current state.



Simulated Annealing Search



13

Simulated Annealing search

- A hill algorithm that never moves downhill is undoubtedly incomplete (local maximum)
- Purely random path is complete, but extremely inefficient
- Simulated annealing: combines these two algorithms.
- General ideas:
  - Change from the point of view "going up the hill" to "going down the gradient"
  - To place a ping-pong ball in the deepest crevice on a rugged surface, simply release the ball. When it stops, the ball is shaken so that it leaves the local minimum
  - Annealing, in metallurgy, is the process of hardening metals and glass by heating them to high temperatures and then gradually cooling them
  - **Idea**: start shaking hard (high temperature), and gradually reduce the intensity of shaking (lower the temperature)

Simulated Annealing Search

function SIMULATED-ANNEALING (*problem*, *rate*) **returns** a state that is a solution

input: *problem* (a problem), *rate* (a time mapping for "temperature")

local variables: *current (a node)*, *next (a node)*, *T (the "temperature" that controls the likelihood of descending steps")*

current ← CREATE-NODE(START-STATE ([*problem*])

for t ←1 **to** ∞, **do:**

T ← *rate*[t] // decreases the value of T over time

if *T*=0 **then return** *current*

next ← a current successor selected at random

ΔE ← VALUE[*next*] – VALUE[current]

if ΔE>0 **then** *current* ← *next* // *movement improves the situation!*

senão *current* ← *next* // only with probability $e^{\Delta E/T}$

Simulated Annealing: remarks

- Internal selection loop very similar to going up the slope. However, instead of choosing the best move, he chooses a random move
- If the movement improves the situation, it will always be accepted. If it gets worse, the algorithm will accept the move with some probability less than 1
- The probability is reduced:
  - Exponentially with the "poor quality of the movement"
  - As the temperature T decreases: bad movements are more likely to be allowed in the beginning, when the temperature is high, and then become more unlikely
    - $e^{\Delta E/T}$

Simulated Annealing: remarks

- Complete?
  - Yes (if there is a solution)
- Optimal?
  - T is allowed to cool very slowly.
  - Over time (cooling), this algorithm starts to function as the Hill Slope.
- Algorithm used in the 1980s in VLSI layouts
- Disadvantages:
  - How to determine the tempering schedule?
  - Test different schedules and choose the best one!

Local beam search

- Instead of maintaining a single state in memory, it keeps track of k states
  - It starts with k randomly generated states. In each step, all successors of all k states are generated
  - If either is an objective, the algorithm for
  - Otherwise, it will select the best k successors from the complete list and repeat the action

Extra-class Activities

Lecture 5

- Activities
  - Leitura:
    - RUSSELL, S. NORVIG, P. Inteligência Artificial. 3ª edição. Capítulos 3 e 4.
  - Recommended exercises:
    - 3.28, 3.29
    - 4.1 a 4.12, 4.15 a 4.17

References

Lecture 5

- RUSSELL, S. NORVIG, P. Inteligência Artificial. 3ª edição.
- Simões, A. S. Slides de aula: IA para Controle e Automação.