ARTIFICIAL INTELLIGENCE

LECTURE 3

Ph. D. Lect. Horia Popa Andreescu

2012-2013 3rd year, semester 5

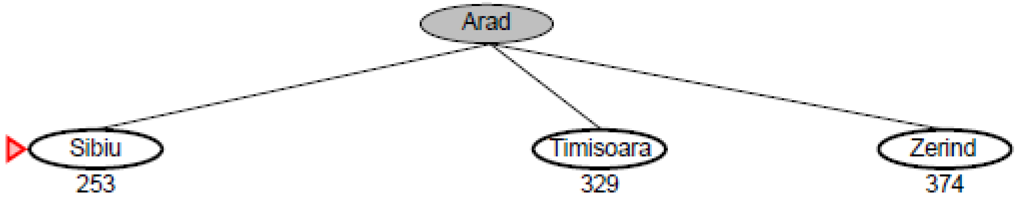
- The slides for this lecture are based (partially) on chapter 4 of the Stuart Russel Lecture Notes [R, ch4], and on the same chapter from Russel & Norvig's book [RN, ch. 4]

INFORMED SEARCH

- Informed search (Heuristic search)
  - The search strategies in which we have additional information about states, mainly problem-specific knowledge, beyond that provided in the problem definition, are called informed search.
  - The general approach is the best-first search, using an evaluation function f(n) to decide about the node n to be expanded. It will choose the node with the best value for f(n).
  - Algorithms specific to informed search treated:
    - Best-first search
    - Greedy best-first search
    - A* (and variants: IDA*, SMA*)
    - RBFS
    - Hill-climbing
    - Simulated annealing
    - Genetic algorithms (brief introduction)

Best-first search

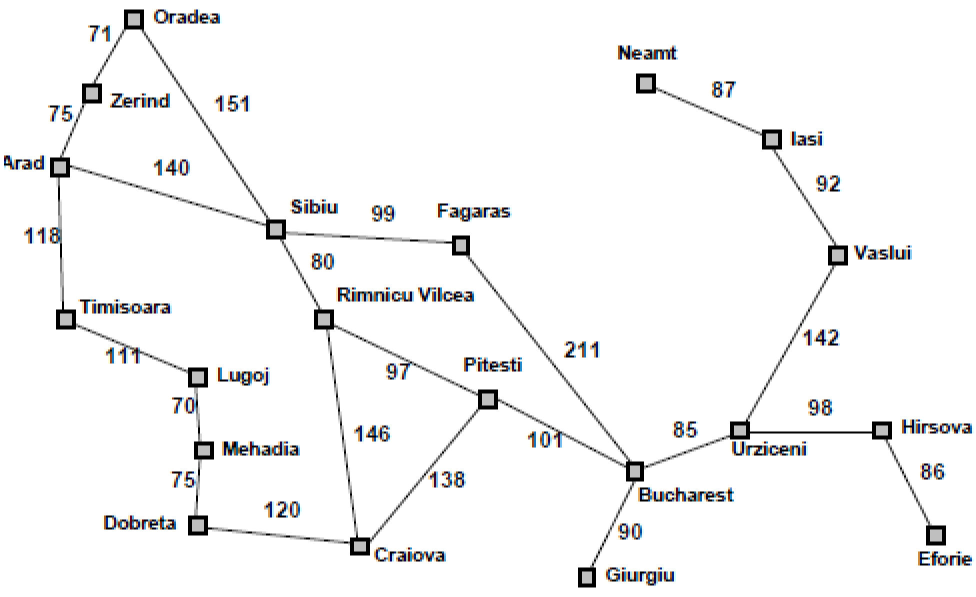
- The node with the lowest value for the evaluation function is selected
- A family of B-F uses instead of f(n), a heuristic function
  - h(n) = estimated cost of the cheapest path to the goal node



GREEDY BEST-FIRST SEARCH

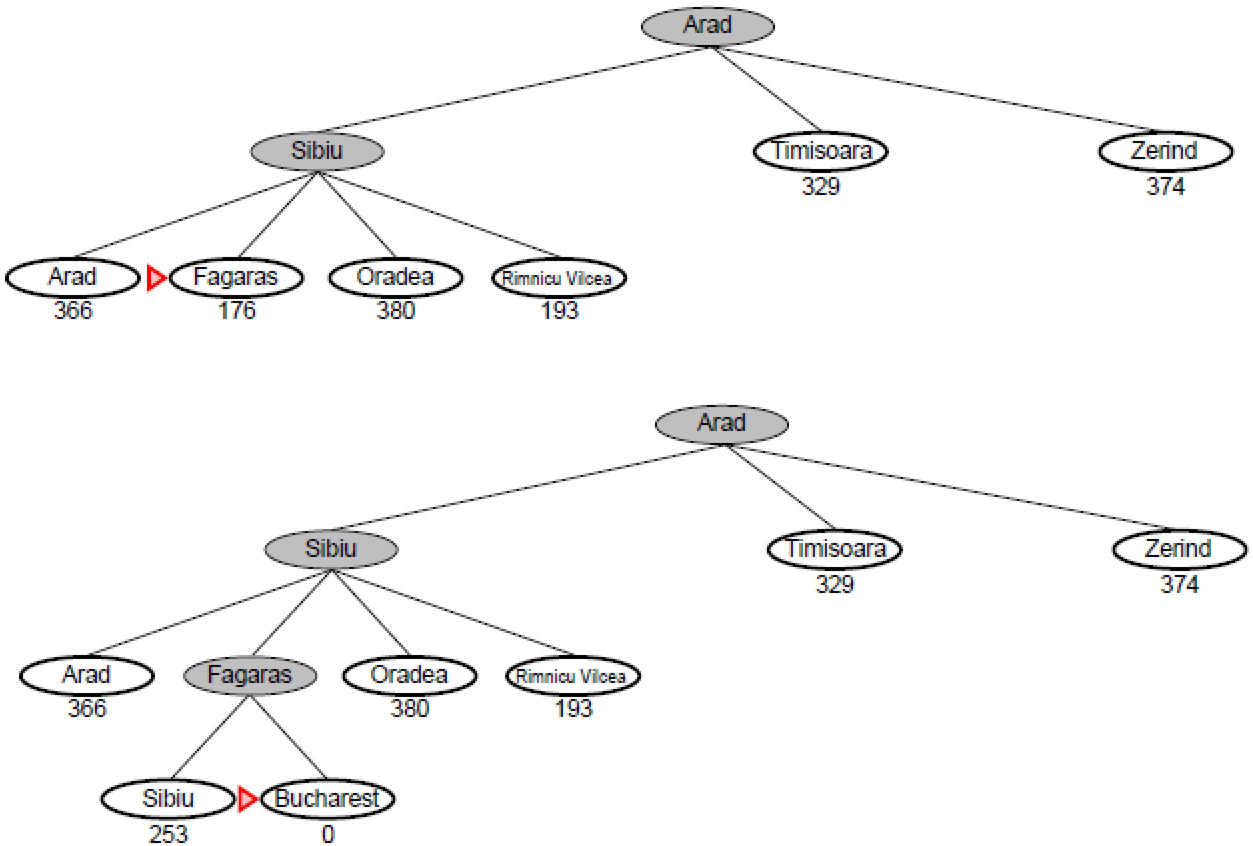
- [R, ch4/slide5]

Romania with step costs in km



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

A* SEARCH

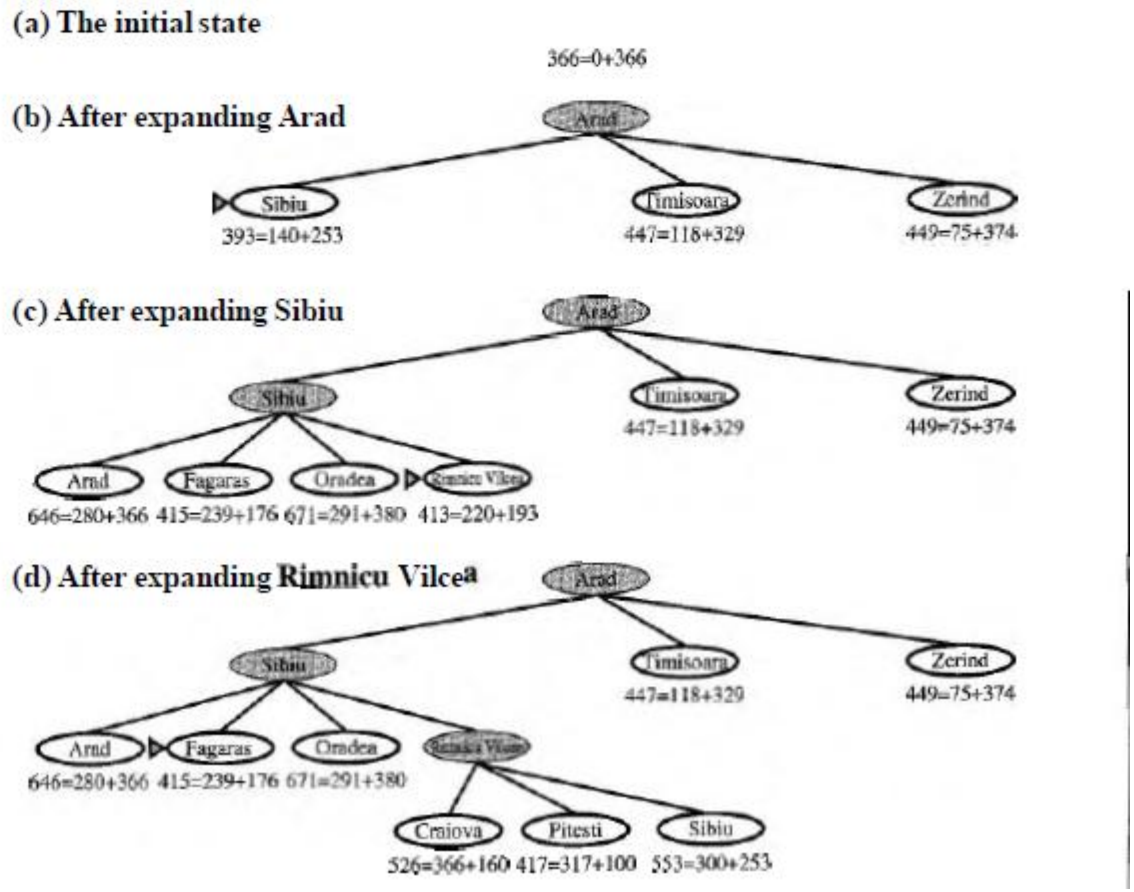
- Uses the function
  - $f(n) = g(n) + h(n)$

Where f(n) – the cost from start, through n, to the goal

g(n) – the cost from start to node n

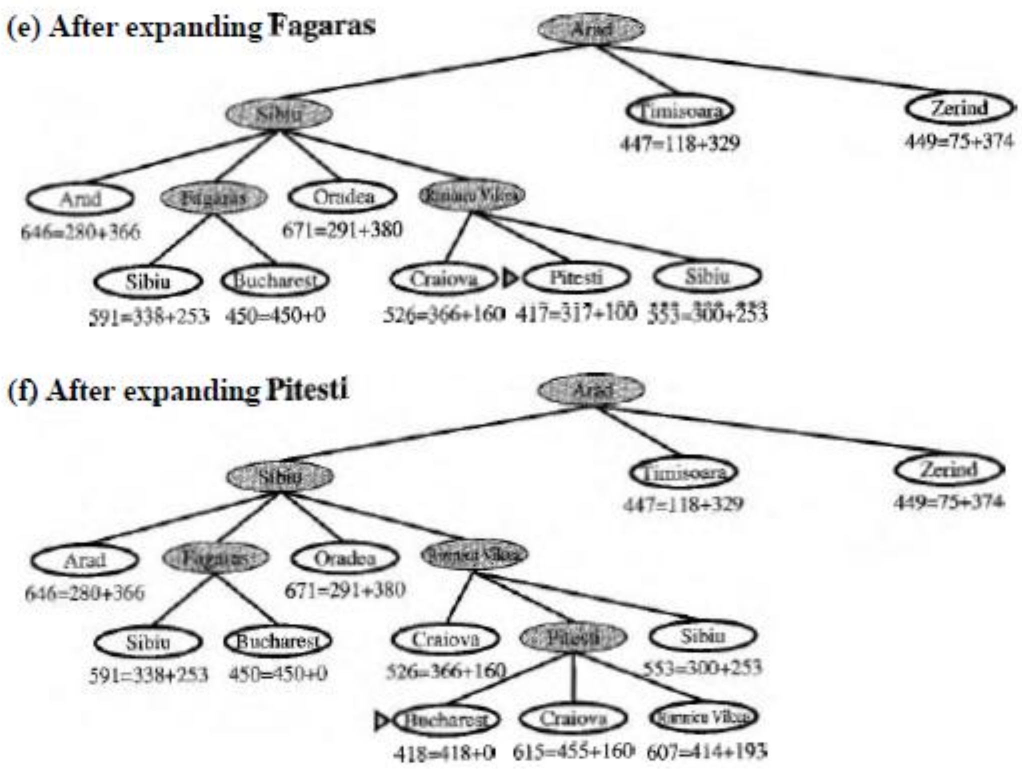
h(n) – the cost from node n to the goal

A* SEARCH (EXAMPLE 1)



A* SEARCH (EXAMPLE 2)

- [RN, 98] A* example



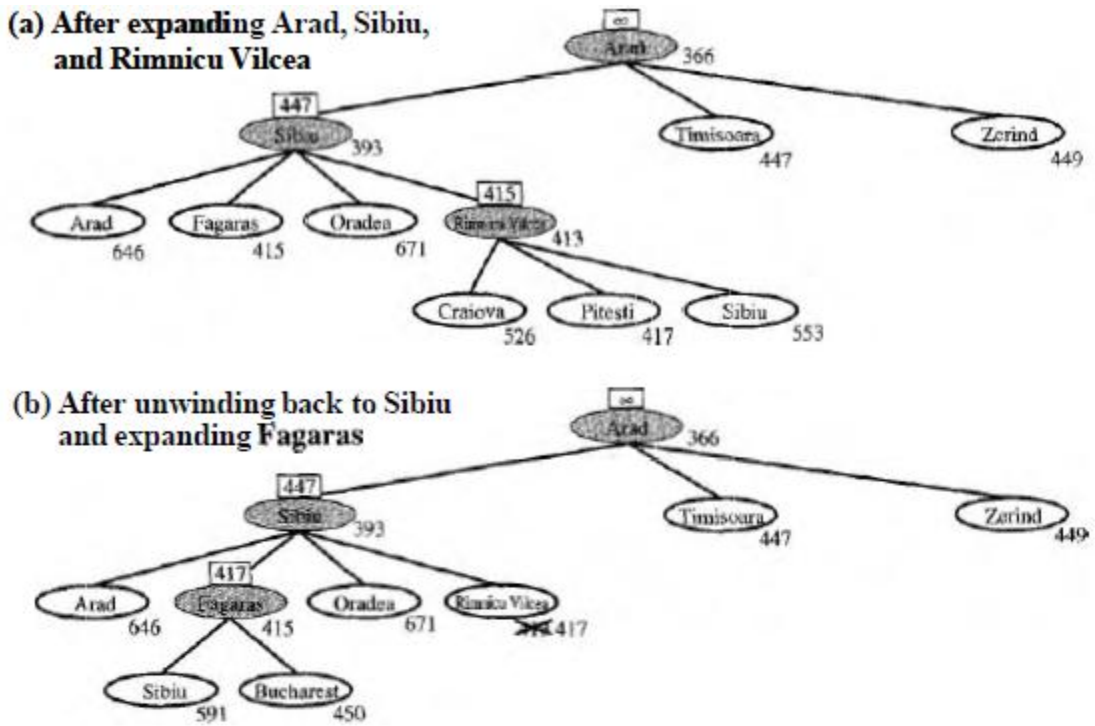
IDA* - ITERATIVE DEEPENING A*

- Iterative deepening A* (IDA*) uses the idea from Iterative Deepening (ID) algorithm (lecture 3) to the A* algorithm in order to decrease memory requirements.
- The cutoff used is based on f(n) instead of the depth, as for ID. The cutoff value is the smallest f-cost of any node that is grater than the value of the previous cutoff.
- It is practical since it avoids keeping a sorted queue of nodes

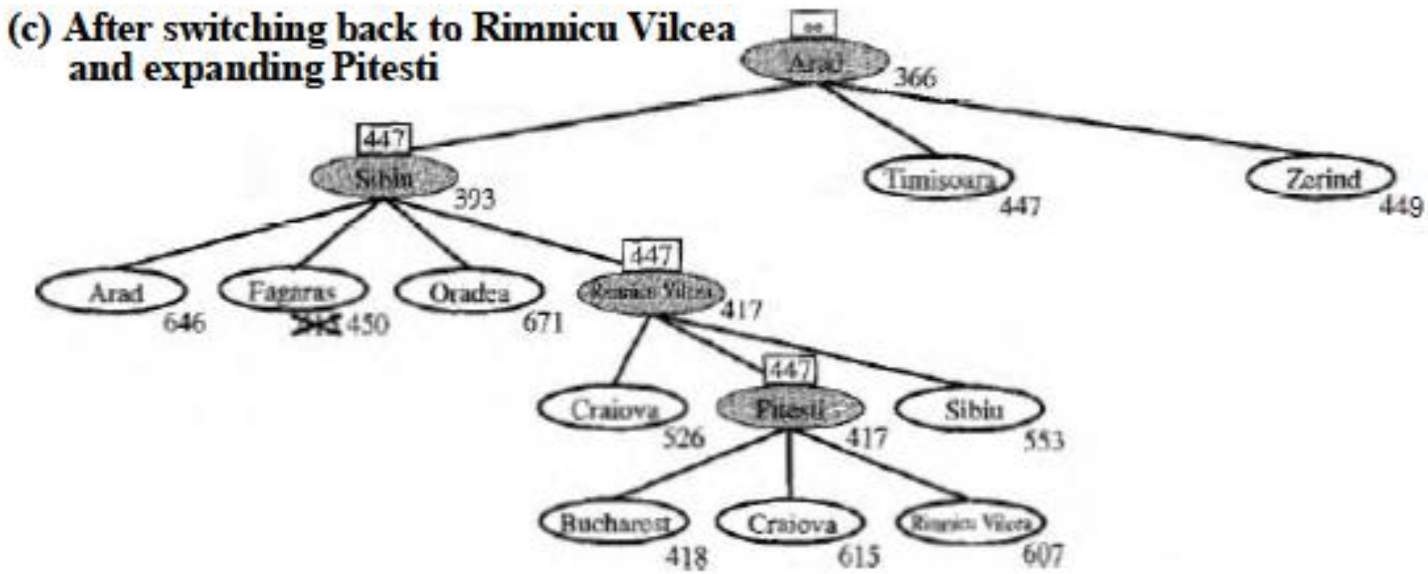
RBFS – Recursive best-first search

- RBFS tries to mimic the behavior of best-first search, but using only linear space.
- It works as a recursive depth-search algorithm, but instead of expanding nodes indefinitely, it keeps track of the f-value of the best alternative path available from any ancestor of the current node.
- If the current node exceeds this limit, then the recursion unwinds back to the alternative path.
- As is unwinds, it also updates the values of the nodes with those of the best f-value of its chldren.
- The consequence is that for each ”forgotten” subtree it keeps a value that will determine in the future if that subtree is worth re expanding.

RBFS – Example [RN, 103]



RBFS – EXAMPLE (2)



(c) After switching back to Rimnicu Vilcea and expanding Pitesti

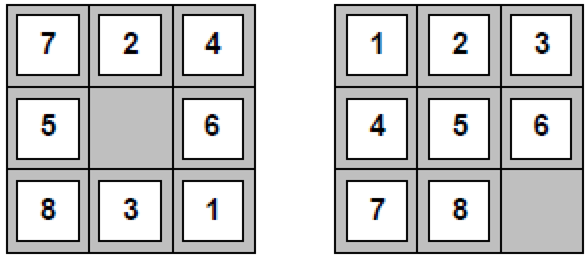
SMA* - Simplified Memory Bounded A*

- Similar to A*, only that it uses a limited amount of memory.
- When the memory if full, it has to discard a node, and it does so with the “worst” one – the one with the highest f-value.
- When discarding a node it backs up the value of the discarded node to it’s parent, similar to RBFS.

HEURISTIC FUNCTION

- In practice, choosing a better heuristic function proofs itself very useful.
- For example, for the 8-puzzle problem we can choose
  - h1(n) = the number of misplaced tiles
  - h2(n) = the sum of the Manhattan distances of the misplaced tiles

HEURISTIC FUNCTION [R, 4A/32]



Start State Goal State

h1(n) = 6
h2(n) = 4+0+3+3+1+0+2+1=14
If h2(n) >= h1(n) for all n, then h2 dominates h1 and is better for search
Typical search costs:

d=14 IDS = 3,473,941 nodes
A* (h1) = 539 nodes
A* (h2) = 113 nodes
d=24 IDS = 54,000,000 nodes (approximatively)
A* (h1) = 39,135 nodes
A* (h2) = 1,641 nodes

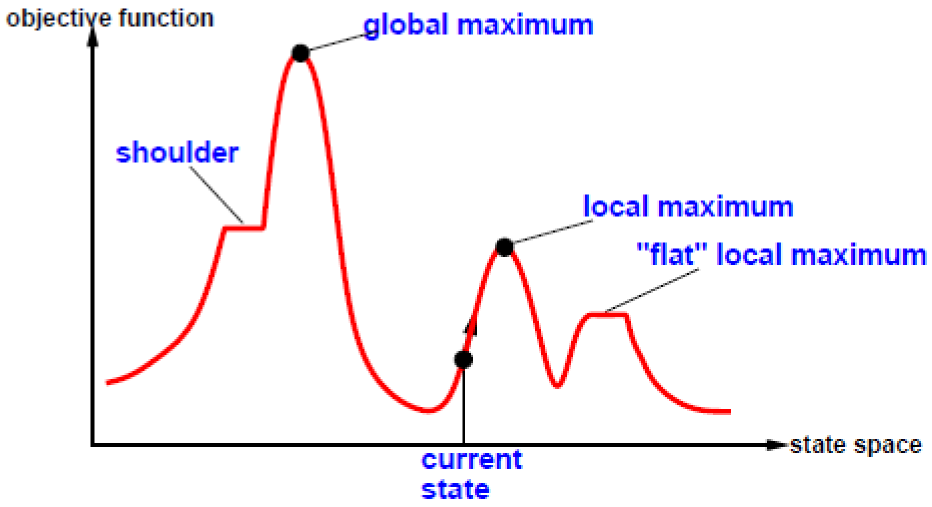
HILL-CLIMBING SEARCH [R, 4B/6]

```
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
```

The problem is to overcome local maximum, flat local maximum and find, as possible the global maximum. See figure on the next slide [R, 4b/7]

Random restart hill climbing overcomes local maxima

Random sideways moves escape shoulders, but loop on flat maxima.


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

SIMULATED ANNEALING [R, 4B/8]

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

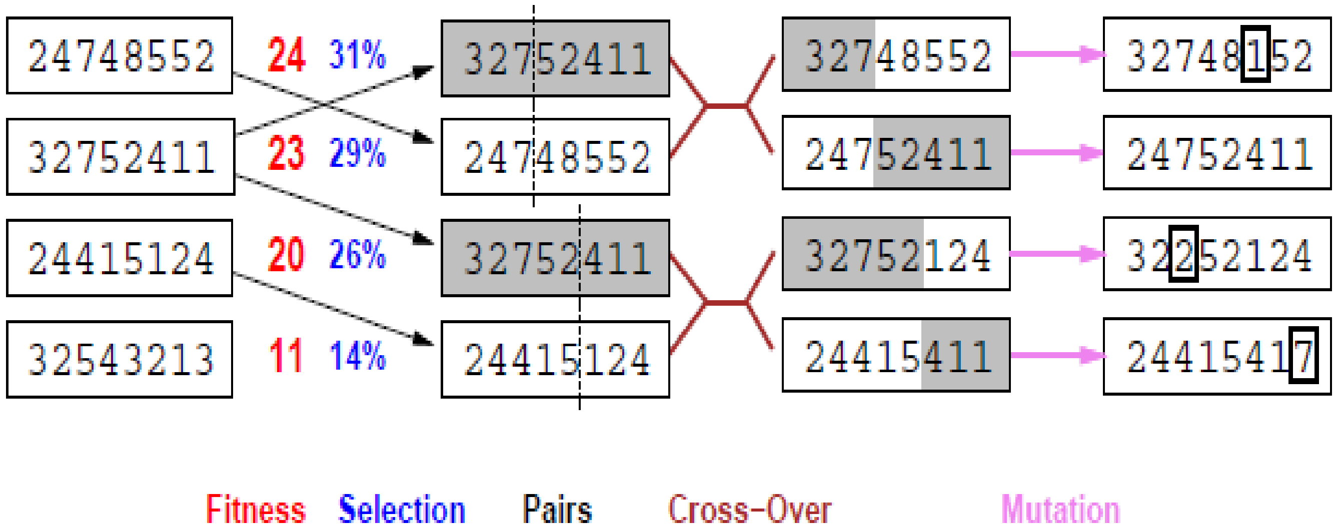
```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward 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
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

LOCAL BEAM SEARCH

- Idea: keep k states instead of 1, choose top k of all their successors
- It is different from k searches running in parallel
- Problem: often all k searches land on the same local hill
- Solution: choose k successors randomly, biased toward good ones.
- It is similar to natural selection which is “captured” better in genetic algorithms

GENETIC ALGORITHMS [R, 4B/11]

= stochastic local beam search + generate successors from pairs of states



Continuous state spaces

- The idea is to use discretization methods that turn continuous spaces into discrete ones.
- For example for the problem: we have 3 airports in Romania

BIBLIOGRAPHY

- [RN] Russel S., Norvig P. – Artificial Intelligence – A Modern Approach, 2nd ed. Prentice Hall, 2003 (1112 pages)
- [R] Stuart Russel – Course slides (visited oct. 2012 at http://aima.cs.berkeley.edu/instructors.html#homework)
- [W1] Mark Watson – Practical Artificial Intelligence Programming With Java AI 3rd ed., 2008
- [C] D. Cârstoiu – Sisteme Expert, Editura ALL, București, 1994