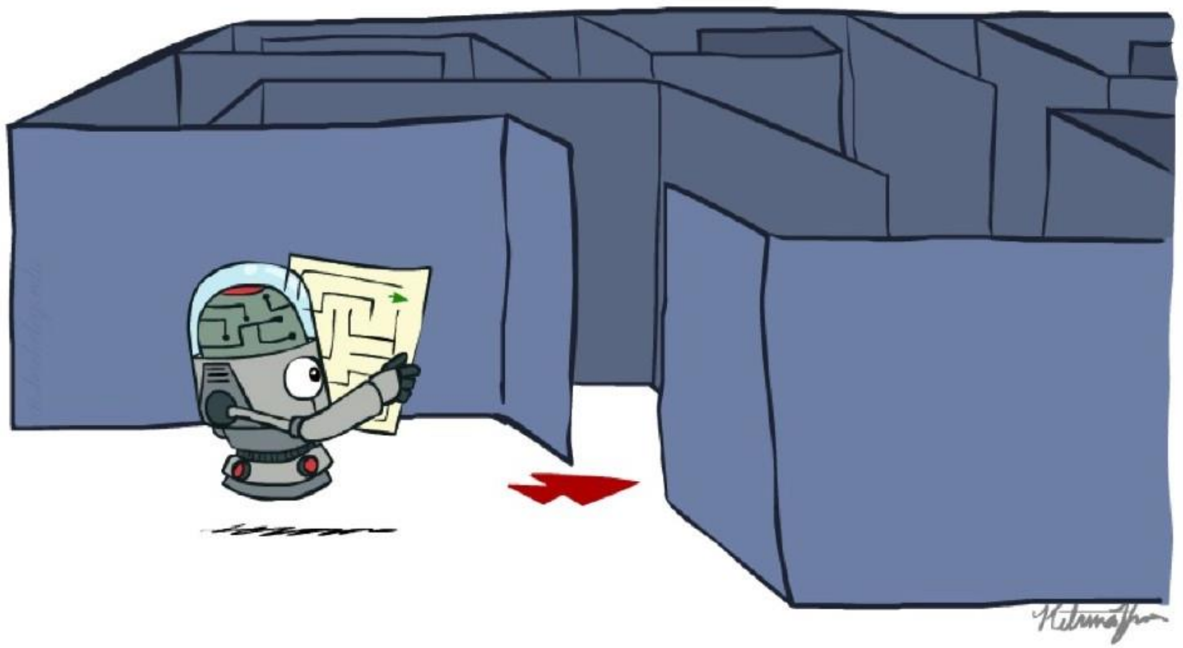
Informed search

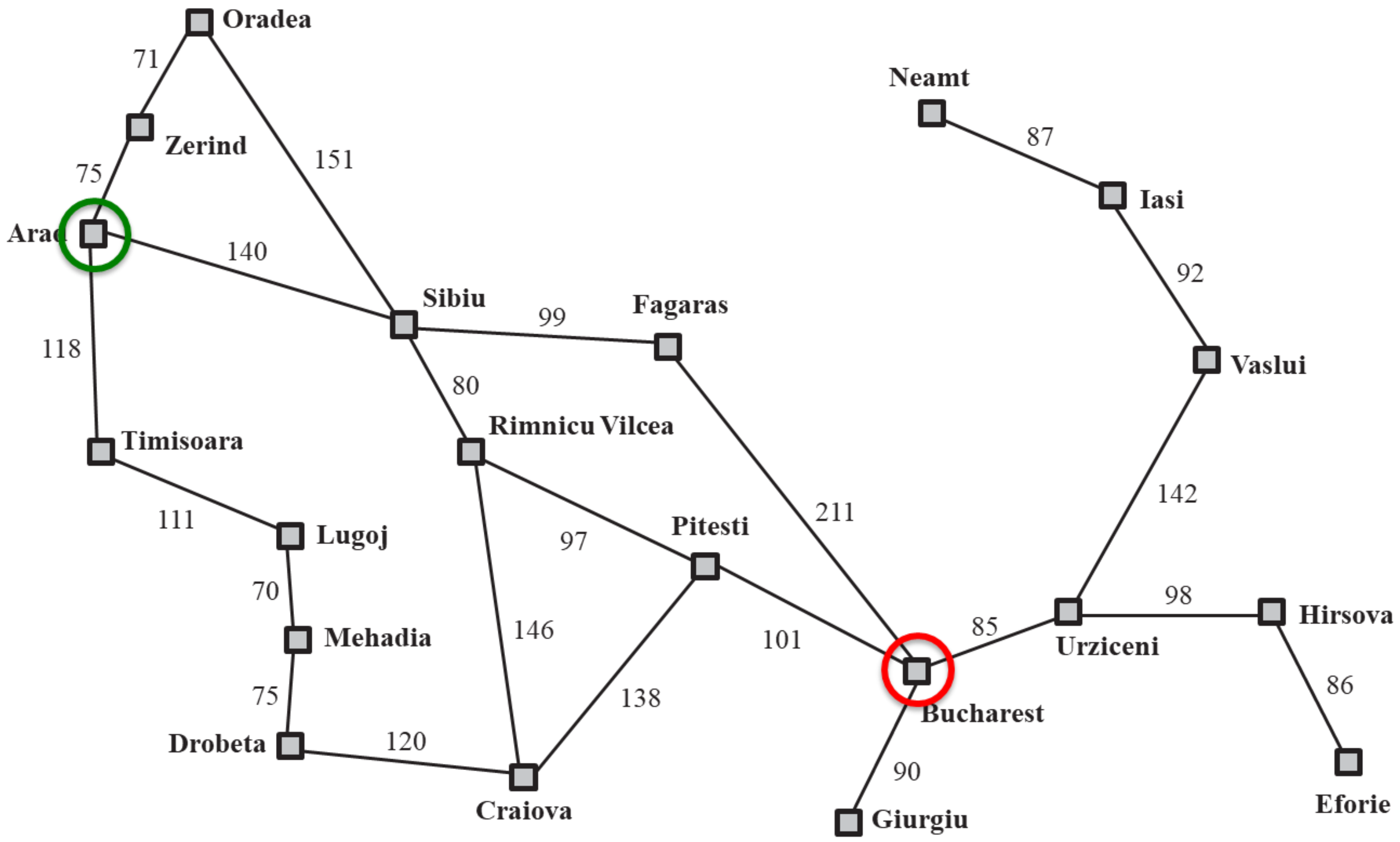
EMI| Semestre 1 | Pr Mohamed RHAZZAF

Recap on search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans



Example: route-finding in Romania



Search Strategies

- **Systematic Non-informed search: Find a path, when exploring space, that leads from the initial state to the final state**
  - depth first and breath first

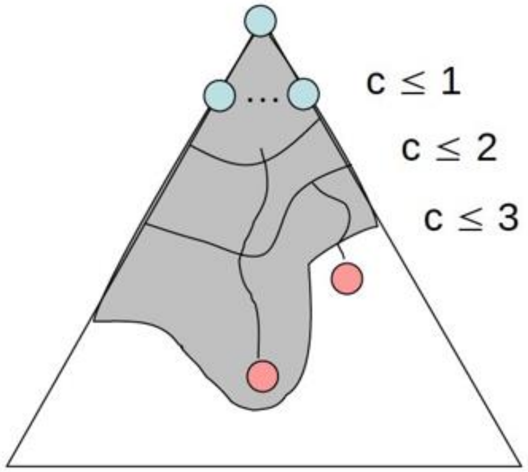
- **Search with information about edge (special case)**
  - uniform cost search (dijkstra)

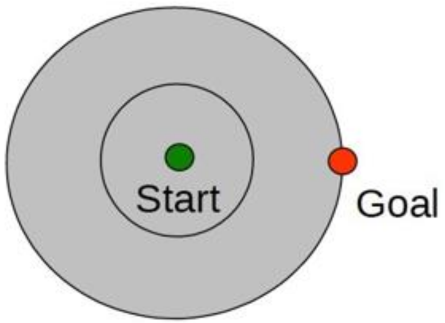
- **Search with information about node**
  - Hill-Climbing, Best-First and Beam

- **Search with information about nodes and edges**
  - A*

Uniform Cost Search

- **Strategy : expend lowest path cost from the start**
- **The good thing: UCS is complete and optimal**
- **The bad:**
  - Explores options in every “direction”
  - No information about goal location





Search Strategies

- **Systematic Non-informed search: Find a path, when exploring space, that leads from the initial state to the final state**
  - depth first and breath first

- **Search with information about edge (special case)**
  - uniform cost search (dijkstra)

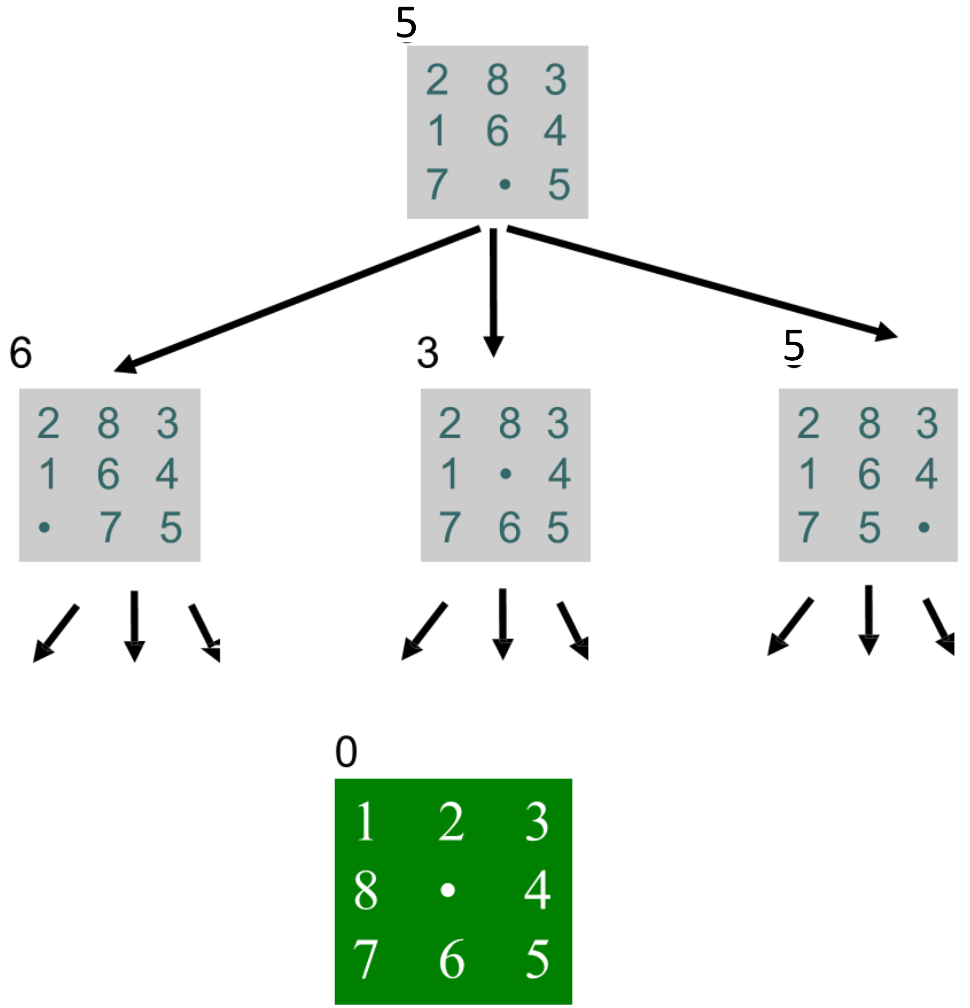
- **Search with information about node**
  - Hill-Climbing, Best-First and Beam

- **Search with information about nodes and edges**
  - A*

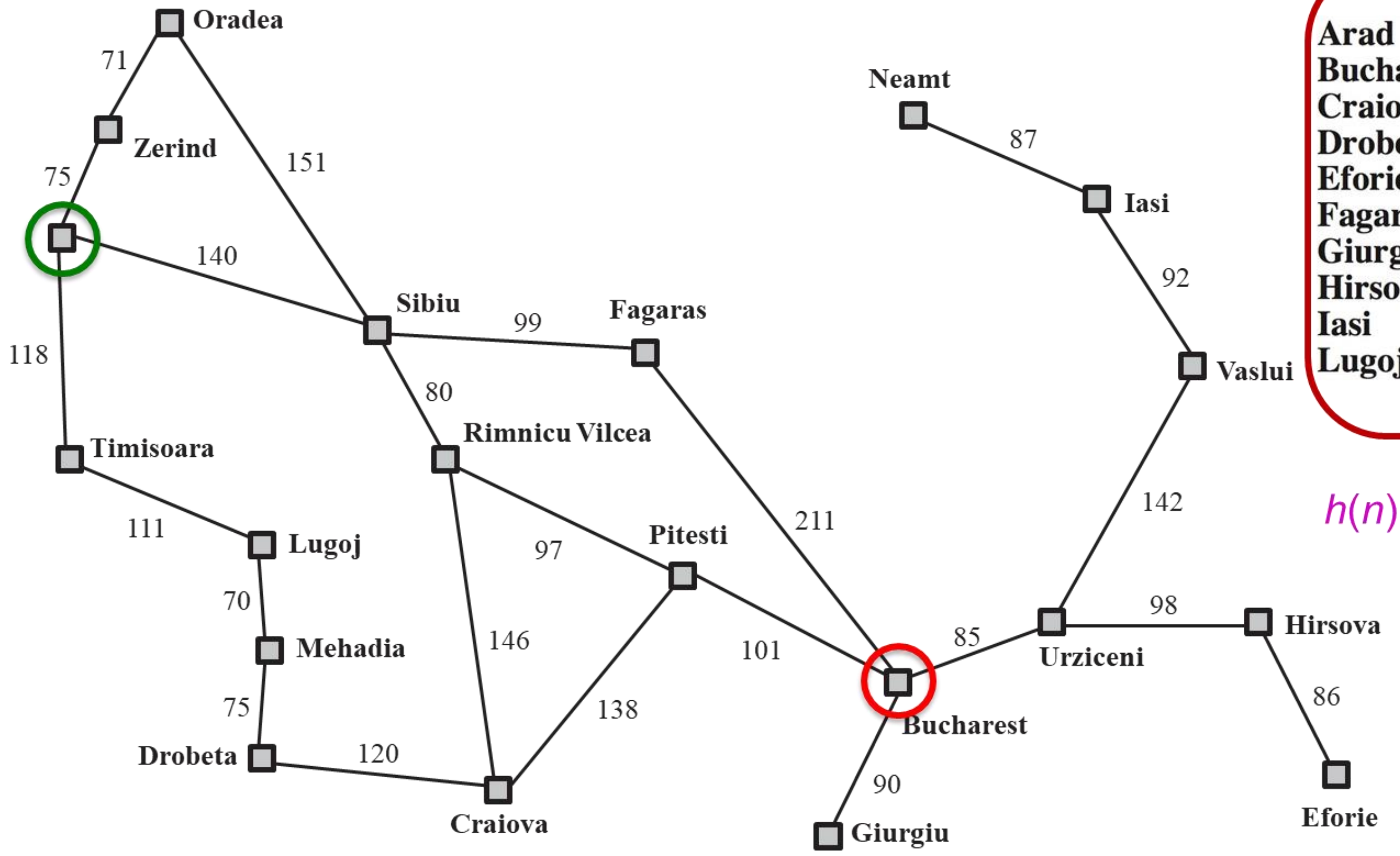
Heuristic method

- **Systematic search**
  - Explore all paths
  - Eventually find a solution
  - Eventually may take a long time
- **Heuristic search**
  - Use of heuristics: choose paths to explore in priority
  - Heuristic criterion: associate with each state a certain estimated value (its approximation of the goal)
    - Ex: h(n) = the distance between city n and destination city
  - For each problem, different estimate
  - Heuristic search: first explore the successor state representing the lowest cost
  - Efficiency of heuristic research depends on how the knowledge of the field is exploited

Example: 8 puzzle heuristic



Example: route-finding in Romania

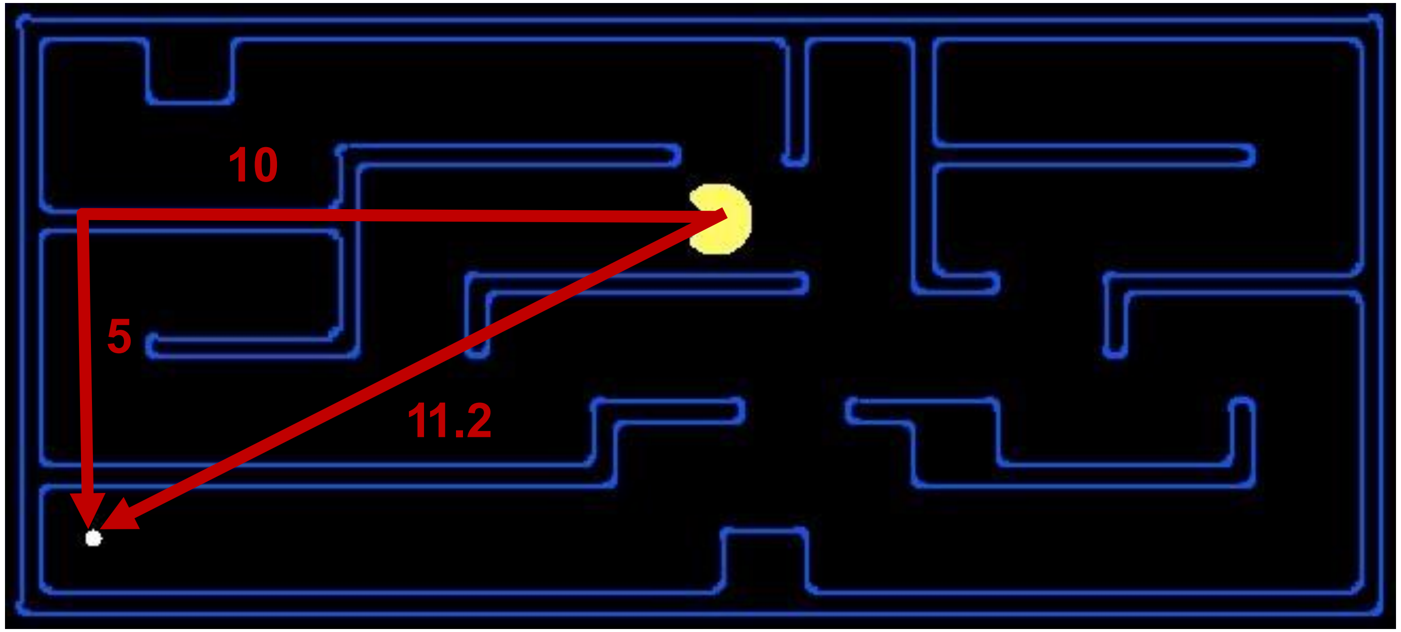


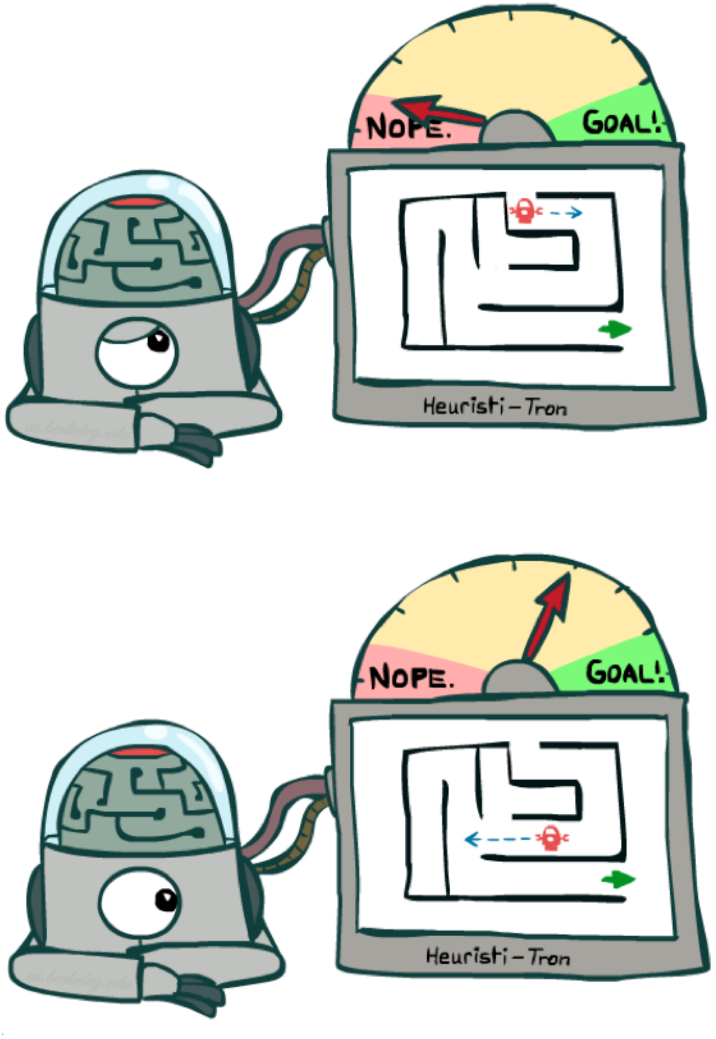
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

$h(n)$ = straight-line distance to Bucharest

Search Problems

- h(n) = Manhattan distance = |x| + |y|
- Is Manhattan better than straight-line distance?

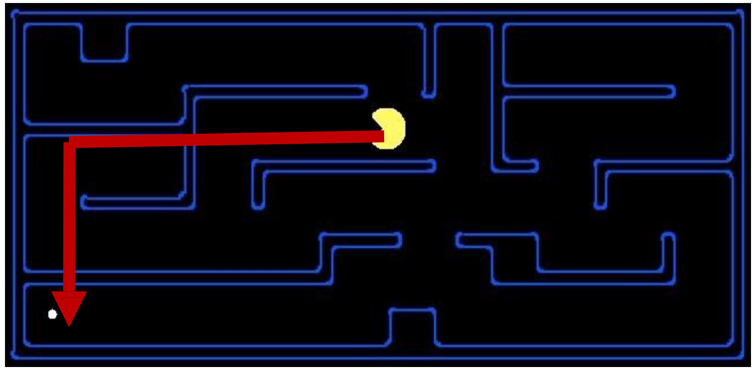




Admissible Heuristics

- **A heuristic h is admissible (optimistic) if:**
  - $0 \leq h(n) \leq h^*(n)$ **where $h^*(n)$ is the true cost to a nearest goal**

- **Example:**



- **Finding good, cheap admissible heuristics is the key to success**

Hill Climbing

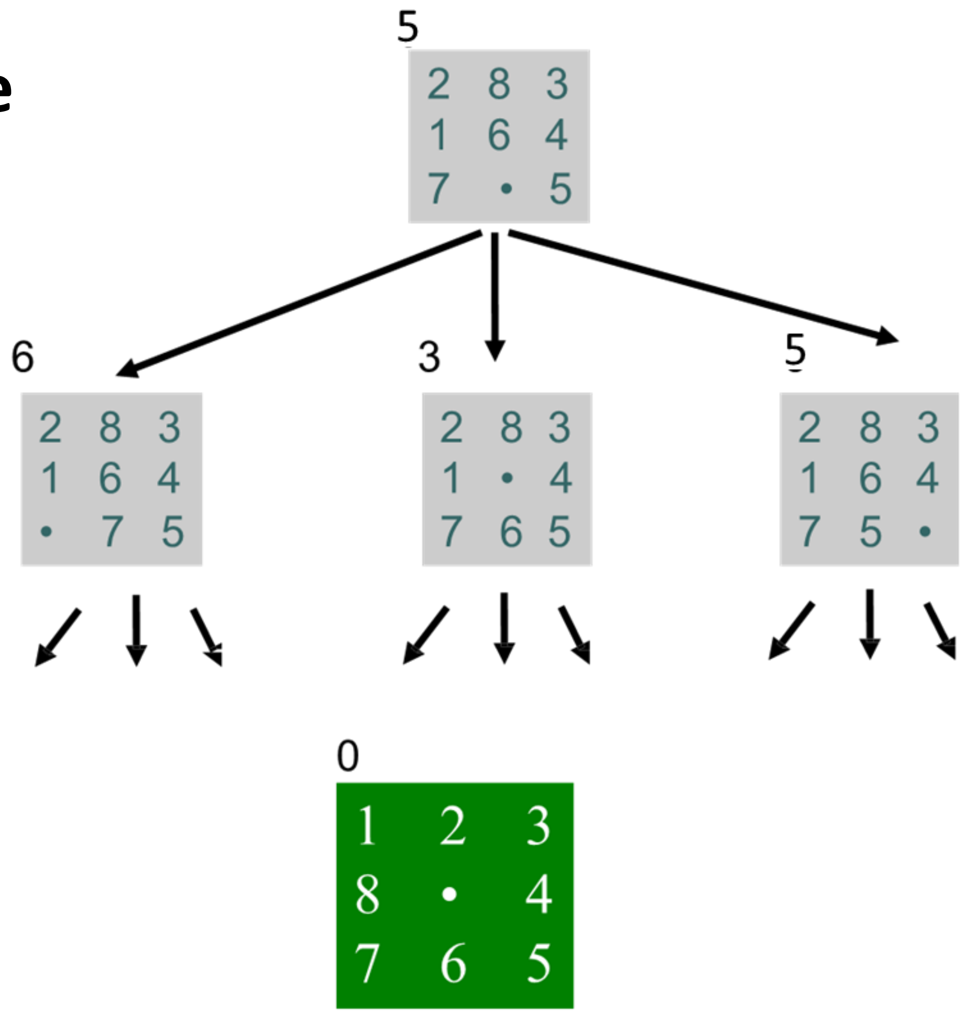
- **Simple, general idea**
  - **Initialization**: Start with a random or arbitrary initial solution.
  - **Evaluation**: Evaluate the quality of the current solution using a fitness or objective function.
  - **Neighbor generation**: Generate neighboring solutions by making small changes to the current one.
  - **Neighbor selection**: Compare the neighbors to the current solution. If a neighbor is better, choose it as the new current solution.
  - **Iteration**: Repeat the evaluation and neighbor selection steps until no neighboring solution is better than the current one.
  - **Termination**: The algorithm stops when it reaches a point where it cannot improve further, meaning it has found a local optimum

Hill-climbing algorithm

```
function HILL-CLIMBING(problem) returns a state
    current ← make-node(initial-state)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.value ≤ current.value then
                return current.state
        current ← neighbor
```

Examples

8 puzzle



Hill Climbing Variant

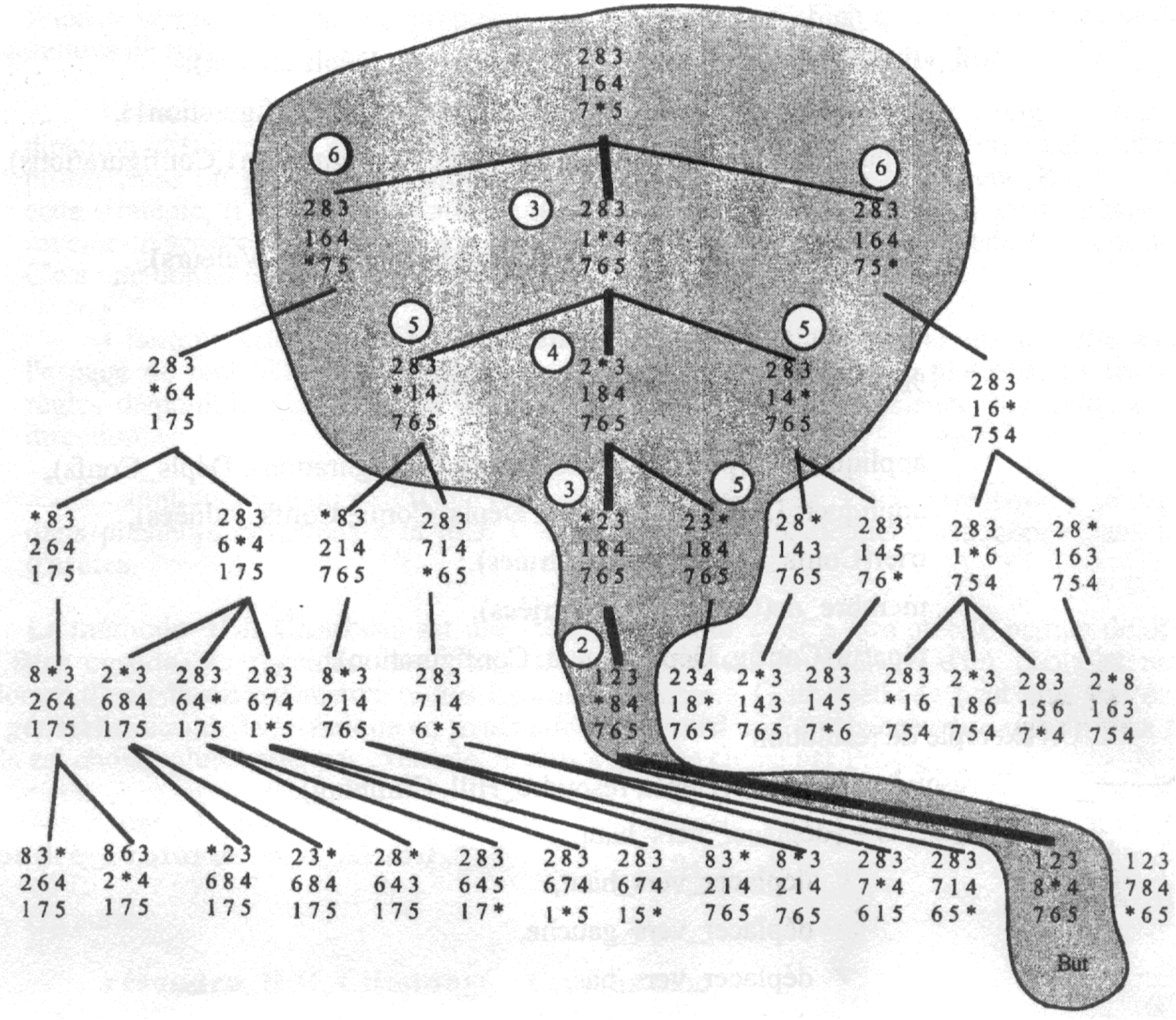
- **Problem with simple Hill climbing**
  - The algorithm considers only one path

  when the successors are not better than this state, the algorithm stops

- **Hill climbing v2**

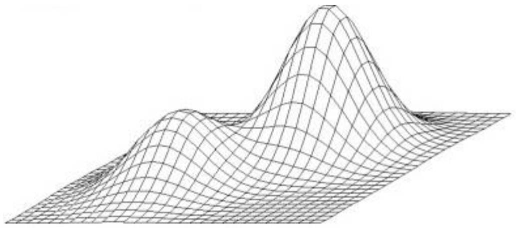
```
function HILL-CLIMBING_V2(problem) returns a state
    current ← make-node(initial-state)
    loop do
        neighbor ← a highest-valued successor of current
        ~~if neighbor.value ≤ current.value then~~
                return current.state
        current ← neighbor
```

Hill Climbing Variant



Hill Climbing

- **Key characteristics**
  - **Local search:** It only explores the neighborhood of the current solution, not the entire search space.
  - **Greedy:** It makes the best immediate move at each step without considering future consequences.
  - **No backtracking:** It cannot go back to previous states, meaning it can get stuck in local optima



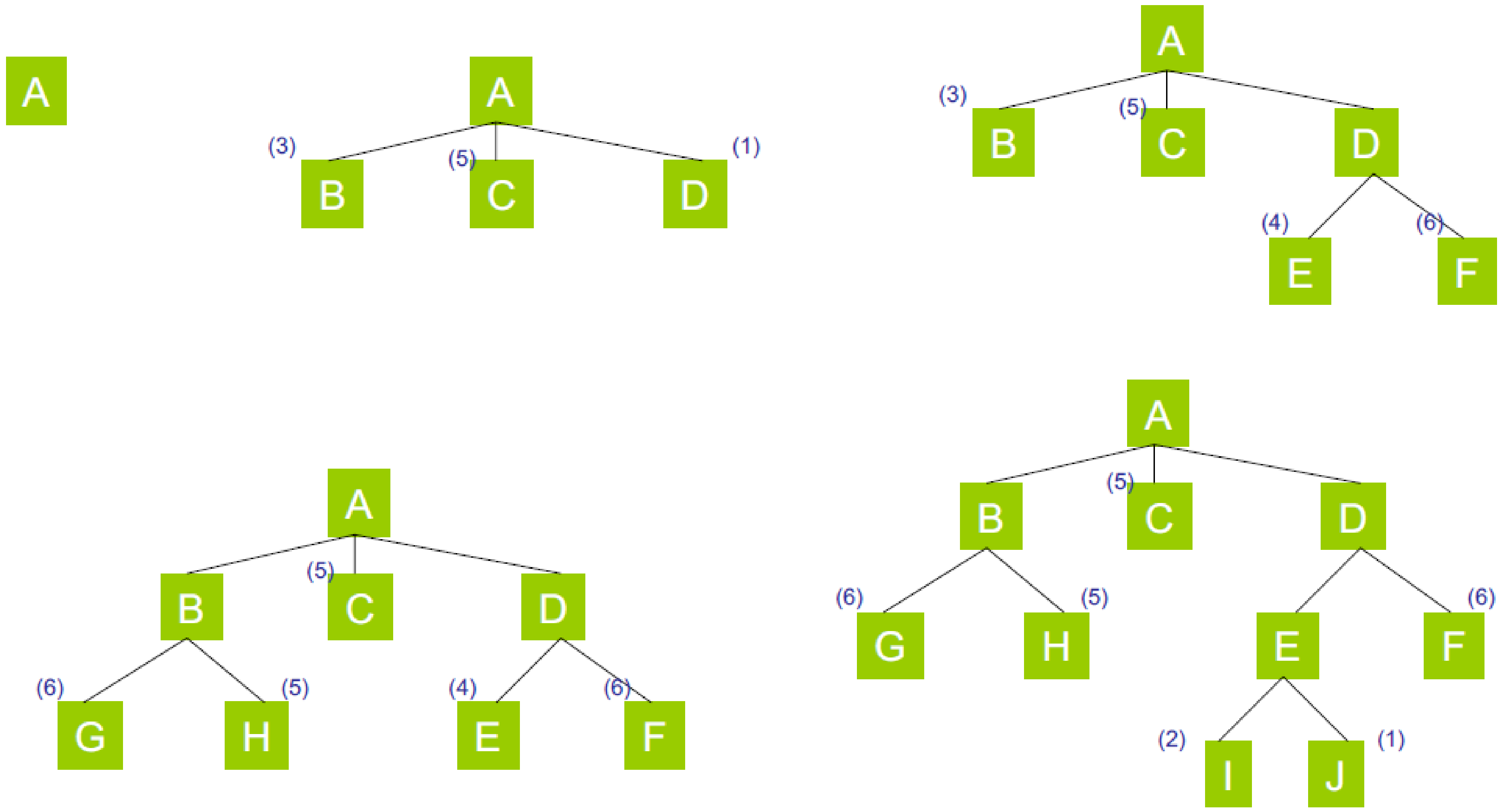
- **Hill Climbing**
  - Local technique, i.e. considers only the immediate consequences of a state

Best First

- **Best First**
  - Combination of depth and breath search
  - Follow one path at a time but change it as soon as a more promising path appears

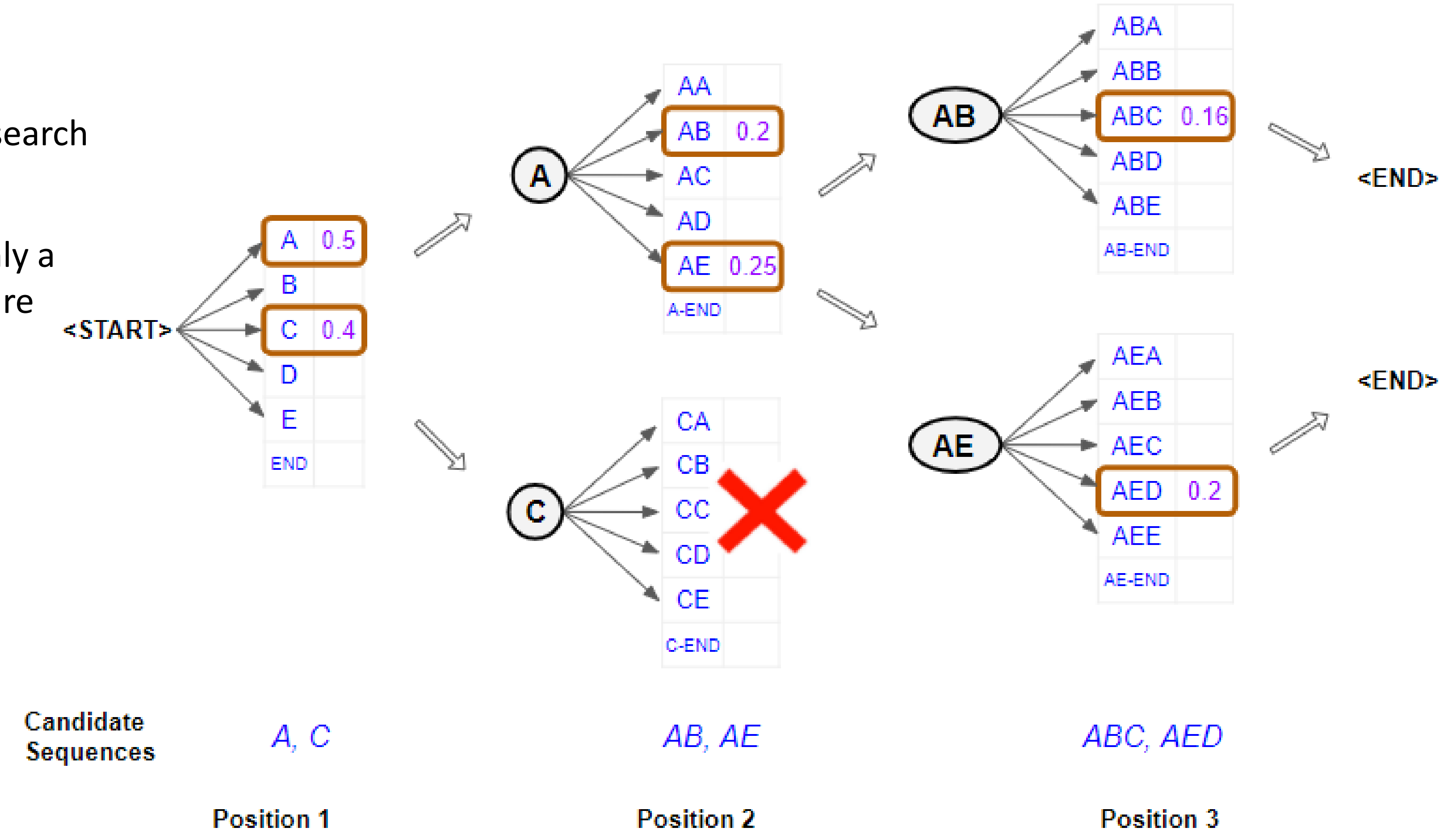
- **Method**
  - Select the most promising node of all generated nodes
  - Perform the expansion of the node using the applicable rules
  - Among the successors, if one of them is a solution, stop
  - Otherwise, all new nodes are added to all the nodes already generated
  - Again, the most promising node becomes the current node

Example Best First



Beam search

- **Beam search**
  - Is an optimization of best-first search
  - Reduces memory load
  - In the beam search method, only a certain number **K** of solutions are kept as candidates



Search Strategies

- **Systematic Non-informed search: Find a path, when exploring space, that leads from the initial state to the final state**
  - depth first and breath first
- **Search with information about edge (special case)**
  - uniform cost search (dijkstra)
- **Search with information about node**
  - Hill-Climbing, Best-First and Beam
- **Search with information about nodes and edges**
  - A*

Generic definition of f

- **In practice we do not know the distance to the goal! That's what we're looking for**
- **On the other hand, we know the optimal distance in the explored part between the root and a node already explored**
- **It is convenient to separate $\boldsymbol{f(n)}$ into two parts:**
  - $\boldsymbol{g(n)}$: the real cost of the optimal path from the root to n in the part already explored
  - $\boldsymbol{h(n)}$: estimated cost of the rest of the path from n to the goal. $\boldsymbol{h(n)}$ is called the heuristic function.

A* Search

Generic definition of f

- **Depending on the weight we want to give to either party, we define f as follows**

```
f(n) = (1-w).g(n) + w.h(n)
```

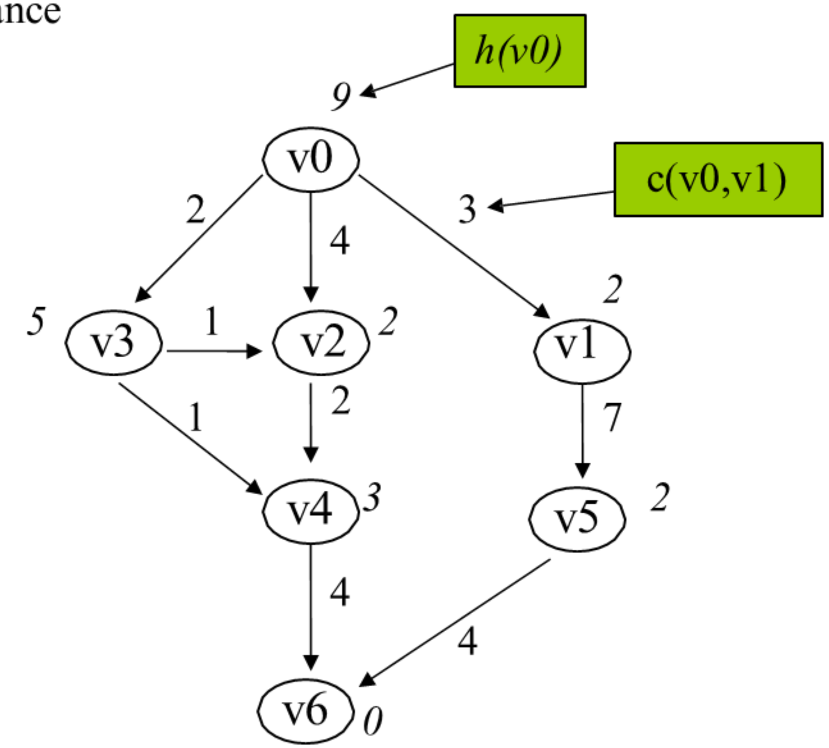
where w is a real number greater than or equal to 0 and less than or equal to 1

- **Depending on the values given to w, one obtains classical search algorithms:**
  - Dijkstra : $w = 0,\ (f(n) = g(n))$
  - Best-first search : $w = 1,\ (f(n) = h(n))$
  - A* : $w = 0.5,\ (f(n) = g(n) + h(n))$ e.g Information about nodes and edges

A* Example

Distance between cities:

v0: departure
v6: destination
h: distance
C: actual distance



Exploration:

1. (v0, 9, void)
2. (v1,5,v0) (v2,6,v0), (v3,7,v0)
3. (v2,6,v0) (v3,7,v0), (v5,12,v1)
4. (v3,7,v0),(v4,9,v2),(v5,12,v1)
5. (v4,6,v3),(v4,9,v2),(v5,12,v1)

Solution: **v0,v3,v4,v6**

Comparison

Uniform Cost (g)

A* (g+h)

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- Protein design
- Chemical synthesis
- …