Lecture 3: Informed search and optimization

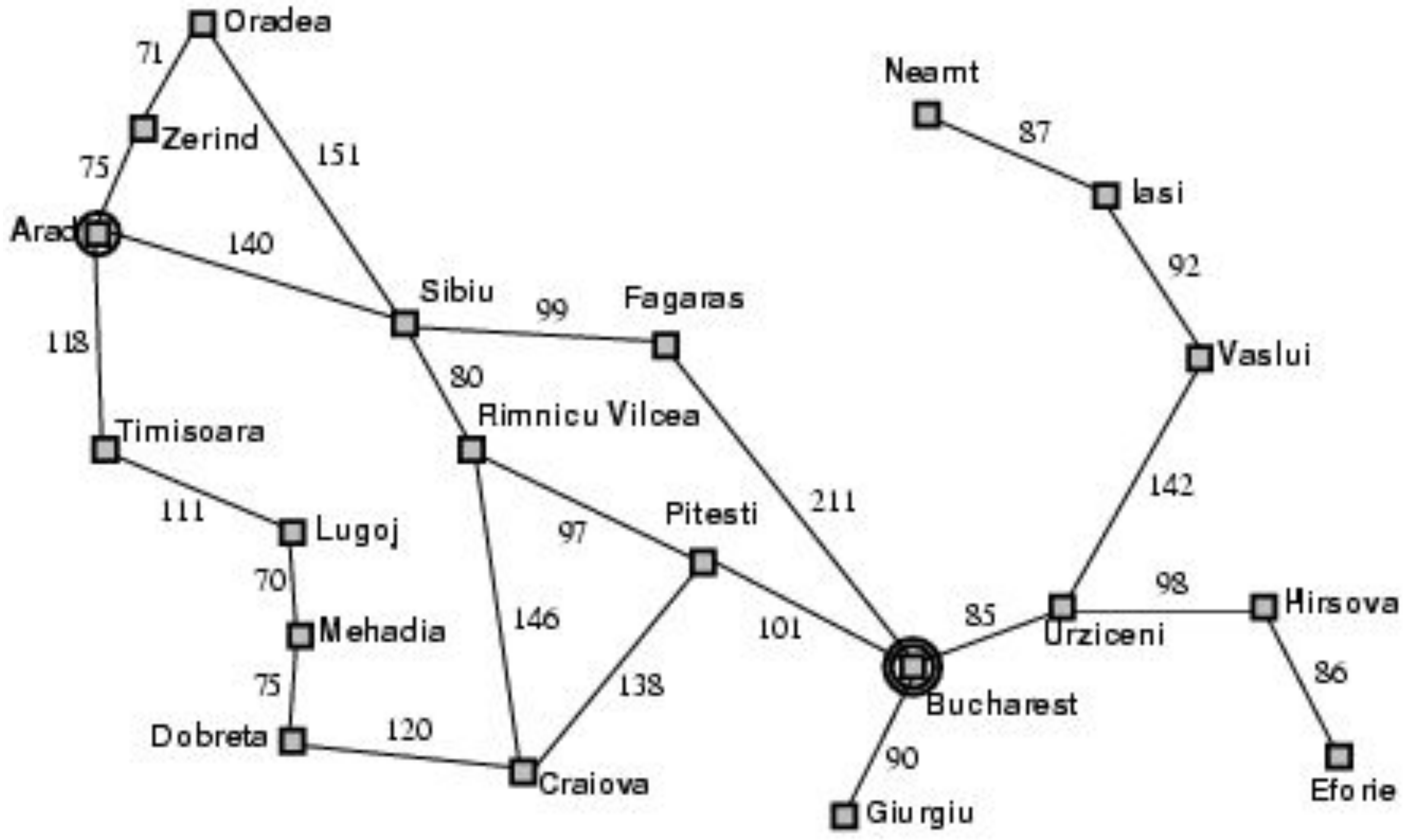
Artificial Intelligence
CS-GY-6613

Julian Togelius
julian.togelius@nyu.edu

The chef recommends:

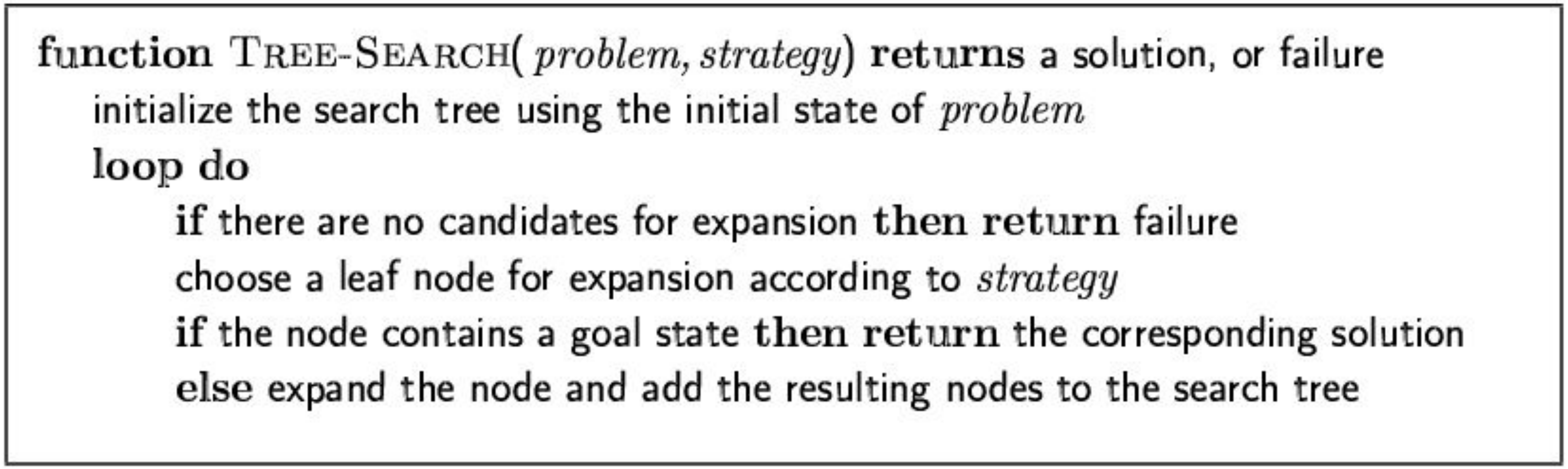
- Best-first search
- Greedy best-first search
- A*
- Heuristics
- Optimization versus tree search
- Hill-climbing
- Simulated annealing
- Evolutionary algorithms
- Assignment 1

Remember Romania



Tree search

- offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

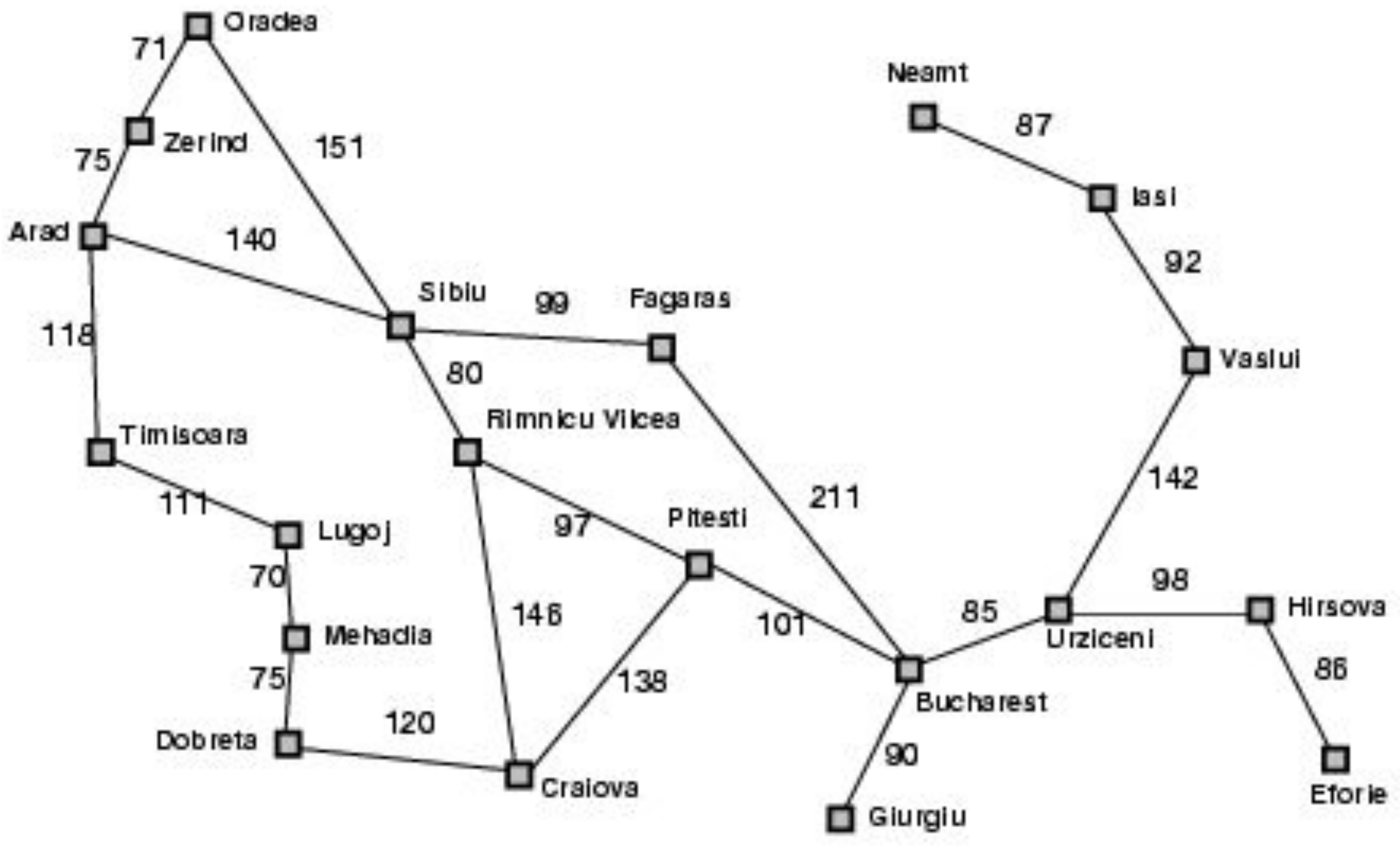


```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

Uninformed search

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Uniform cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Now with straight line distances!



Straight–line distance
to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

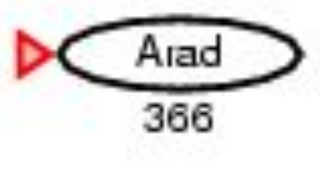
Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)
  = estimate of cost from n to goal

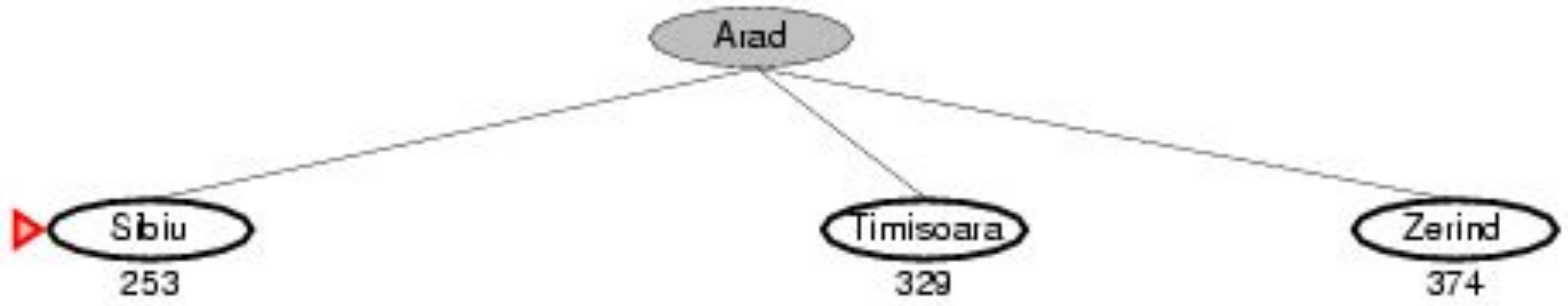
- e.g., h_{SLD}(n) = straight-line distance from ***n*** to Bucharest

- Greedy best-first search expands the node that *appears* to be closest to goal

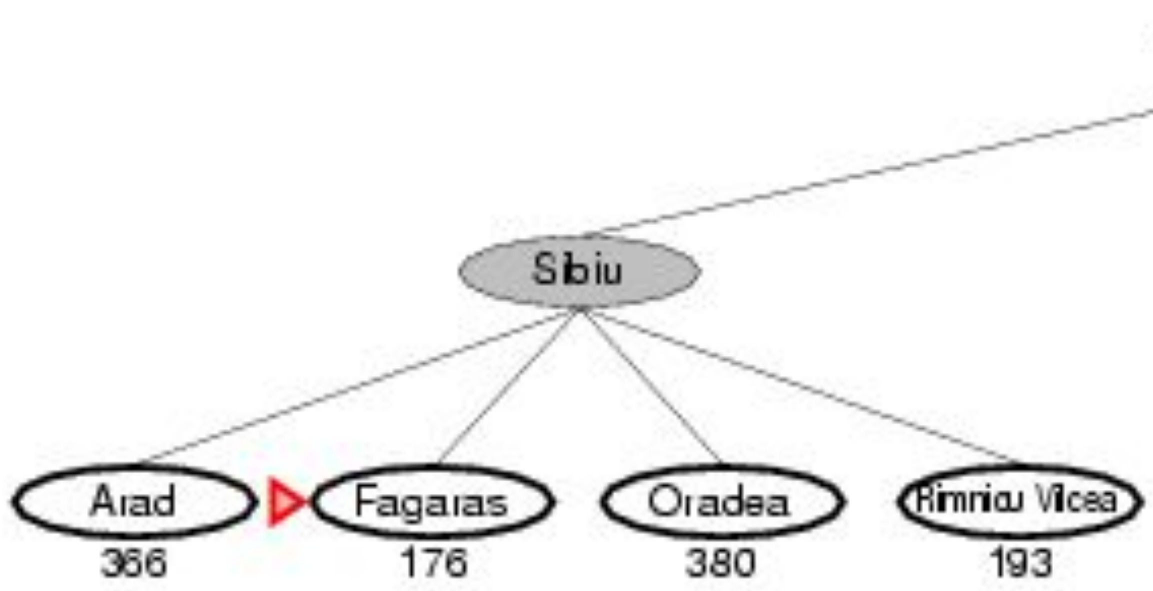
Greedy best-first example



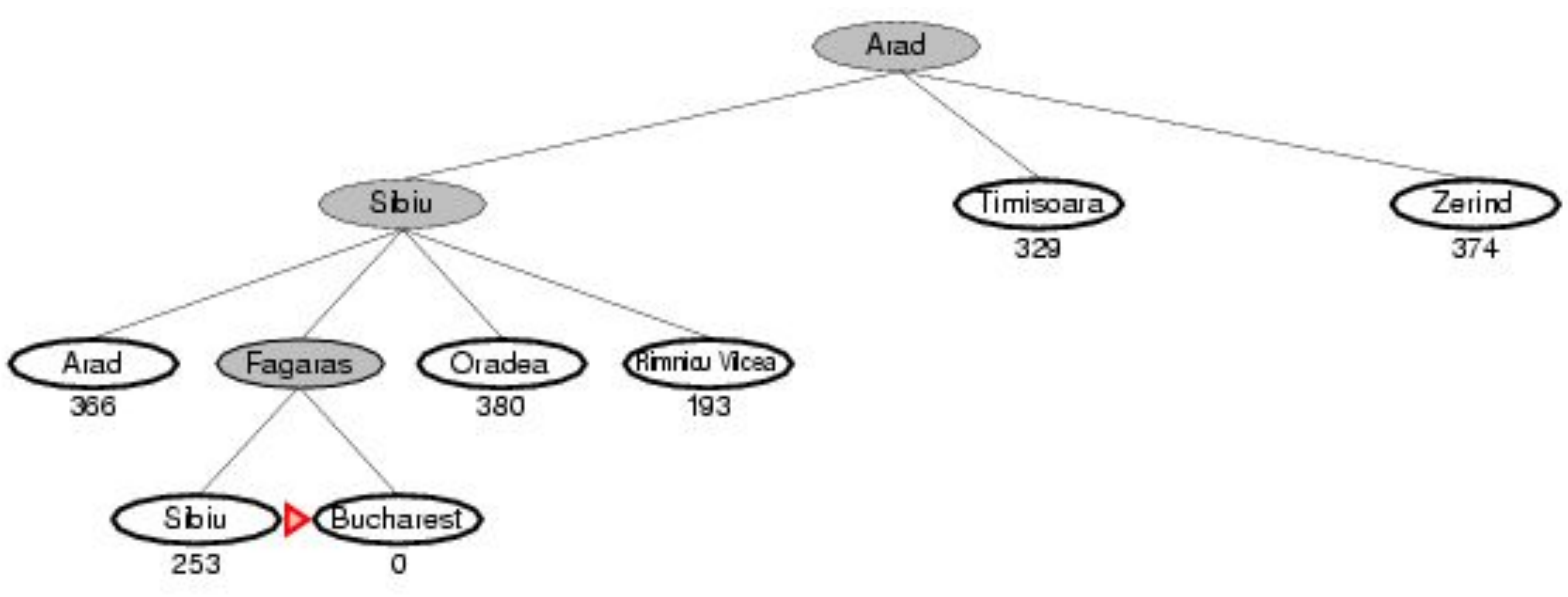
Greedy best-first example



Greedy best-first example



Greedy best-first example



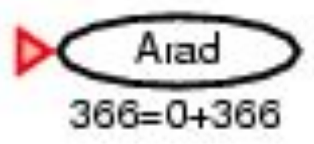
Greedy best-first

- *Complete?* No – can get stuck in loops, e.g., Lugoj > Mehadia > Lugoj > Mehadia >…
- *Time?* O(b^m), but a good heuristic can give dramatic improvement
- *Space?* O(b^m) -- keeps all nodes in memory
- *Optimal?* No

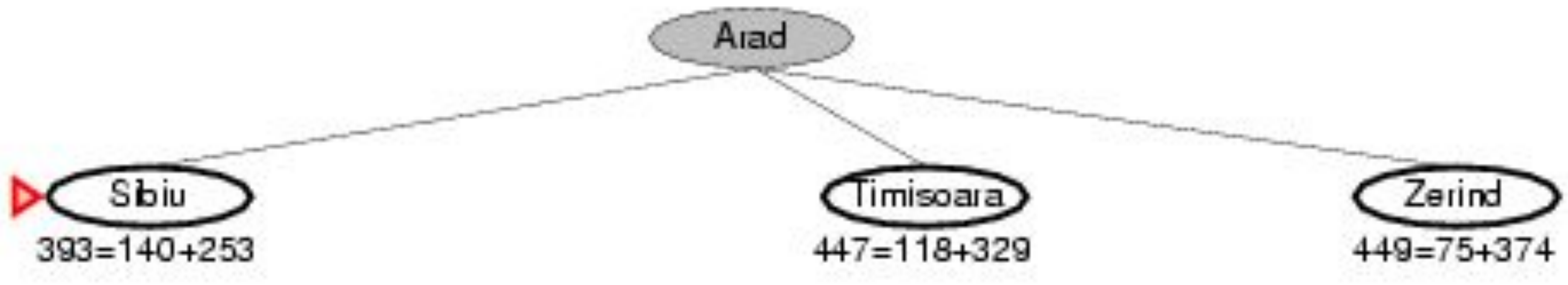
A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

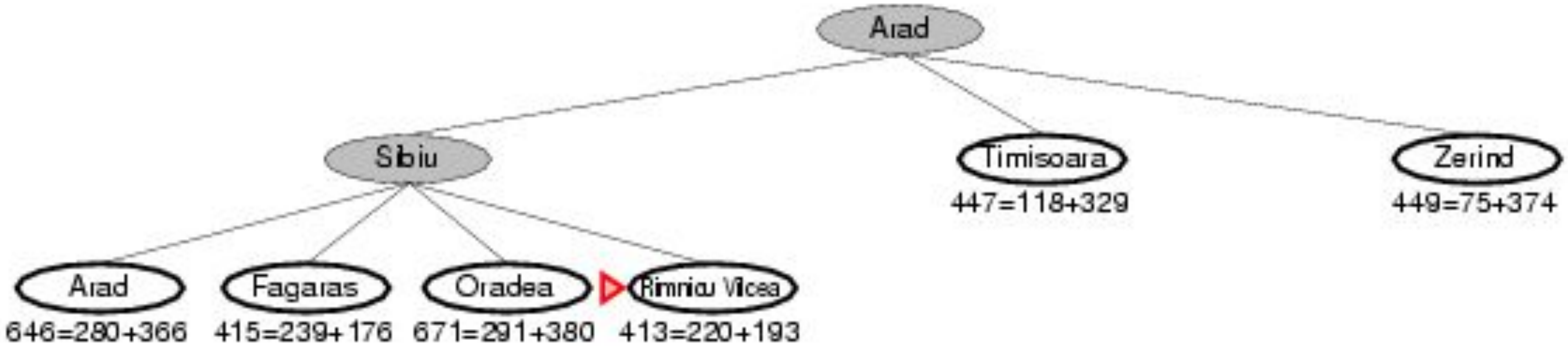
A* search example



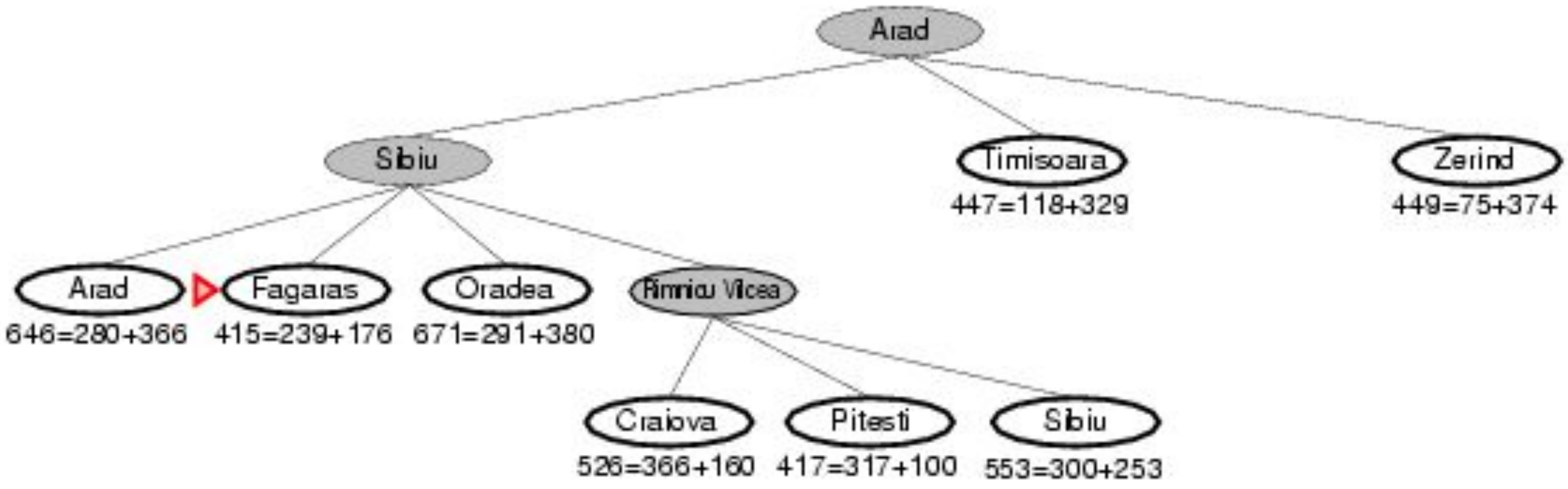
A* search example



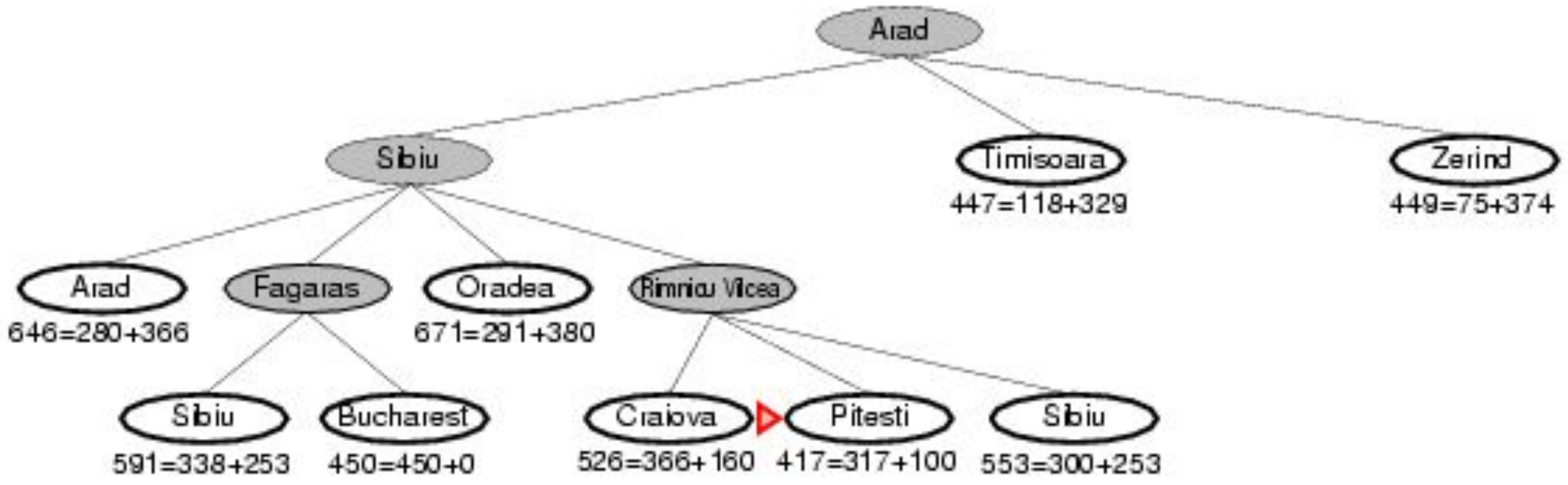
A* search example



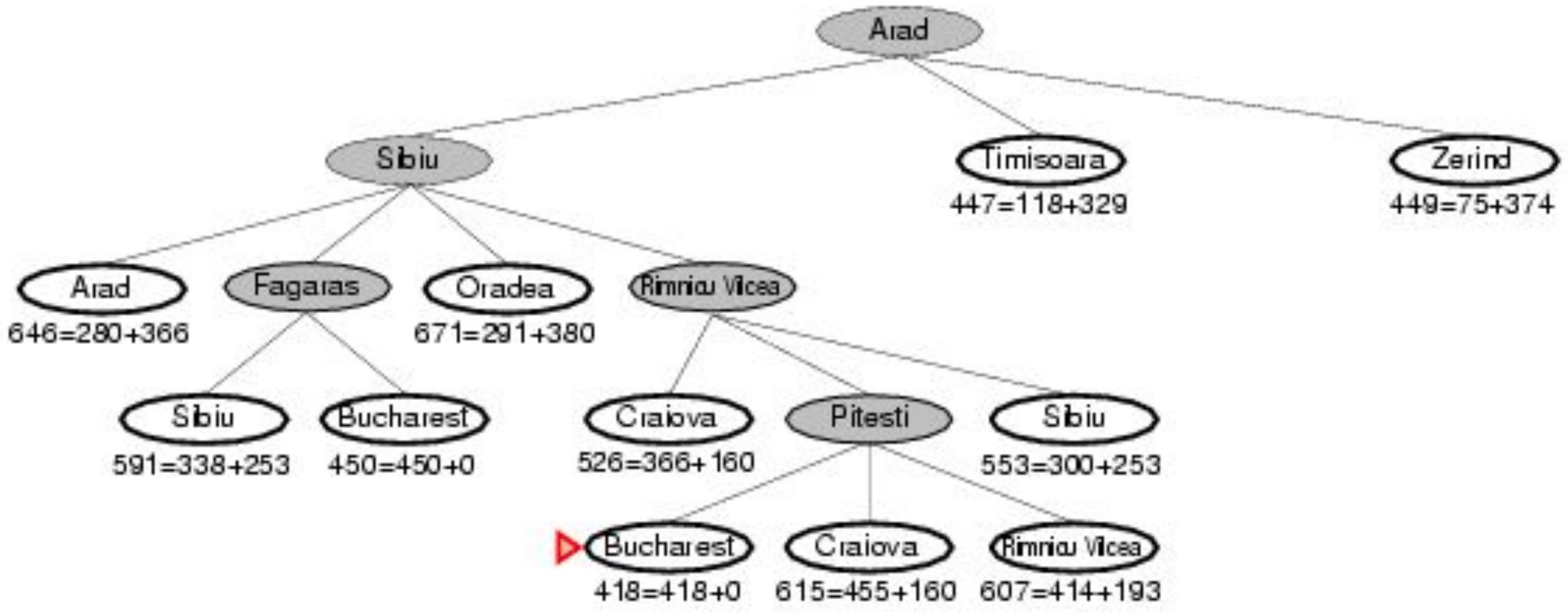
A* search example



A* search example



A* search example



1. Create a search graph G, consisting solely of the start node, *sn*. Put *sn* on a list called OPEN.

2. Create a list called CLOSED that is initially empty.

3. If OPEN is empty, exit with failure.

4. Select the first node on OPEN, remove it from OPEN, and put it on CLOSED. Call this node *n*.

5. If *n* is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from *n* to *sn* in G. (The pointers define a search tree and are established in Step 7.)

6. Expand node n, generating the set M, of its successors that are not already ancestors of n in G. Install these members of M as successors of *n* in G.

7. Establish a pointer to n from each of those members of M that were not already in G (i.e., not already on either OPEN or CLOSED). Add these members of M to OPEN. For each member, m, of M that was already on OPEN or CLOSED, redirect its pointer to *n* if the best path to m found so far is through *n*. For each member of M already on CLOSED, redirect the pointers of each of its descendants in G so that they point backward along the best paths found so far to these descendants.

8. Reorder the list OPEN in order of increasing f values. (Ties among minimal f values are resolved in favor of the deepest node in the search tree.)

9. Go to Step 3.

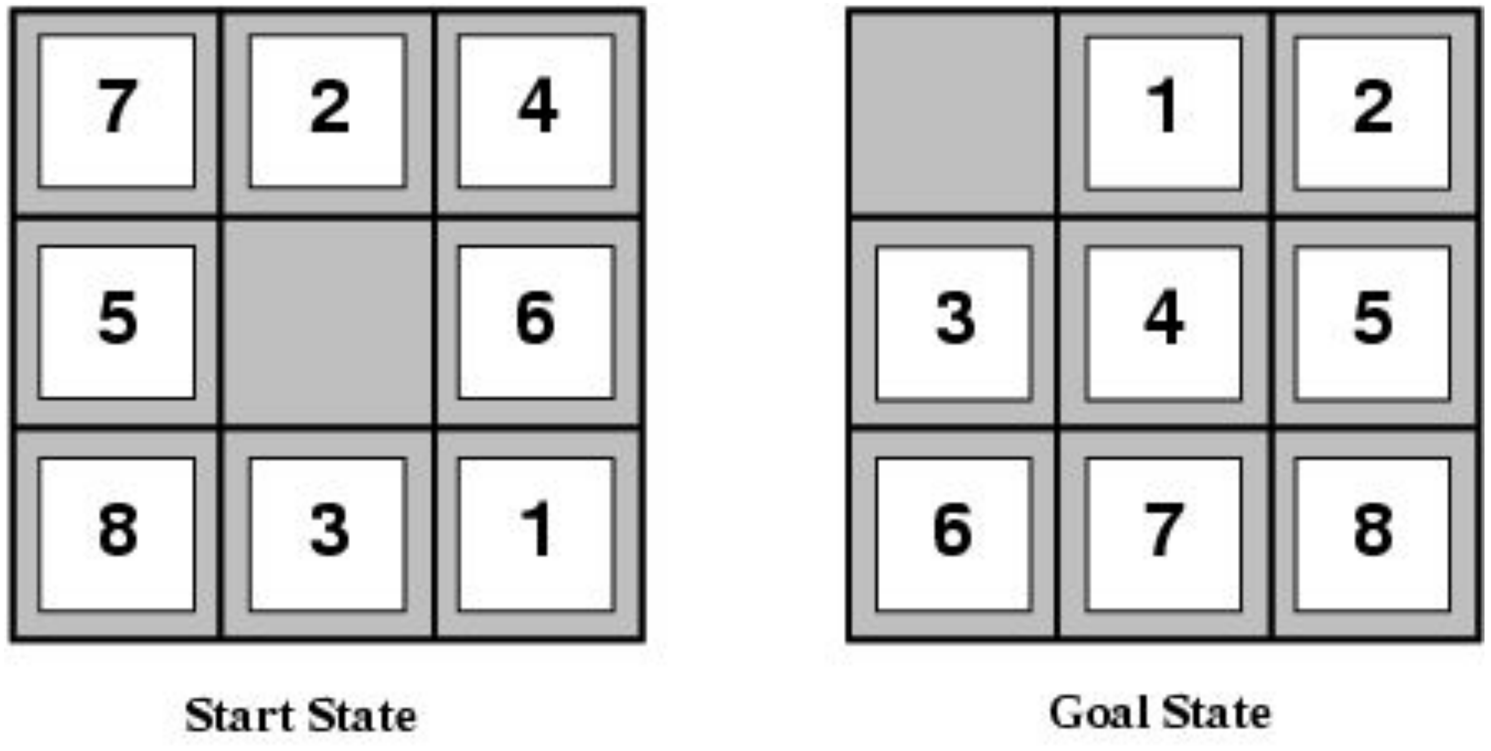
In-class exercise

Admissible heuristics

- A heuristic *h(n)* is admissible if for every node n, *h(n)* ≤ *h*(n)*, where *h*(n)* is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: *hSLD(n)* (never overestimates the actual road distance)
- Theorem: If h(n) is admissible, A* is optimal

Admissible heuristics

- E.g., for the 8-puzzle:
  $h_1(n)$ = number of misplaced tiles
  $h_2(n)$ = total Manhattan distance
  (i.e., no. of squares from desired location of each tile)



Dominance

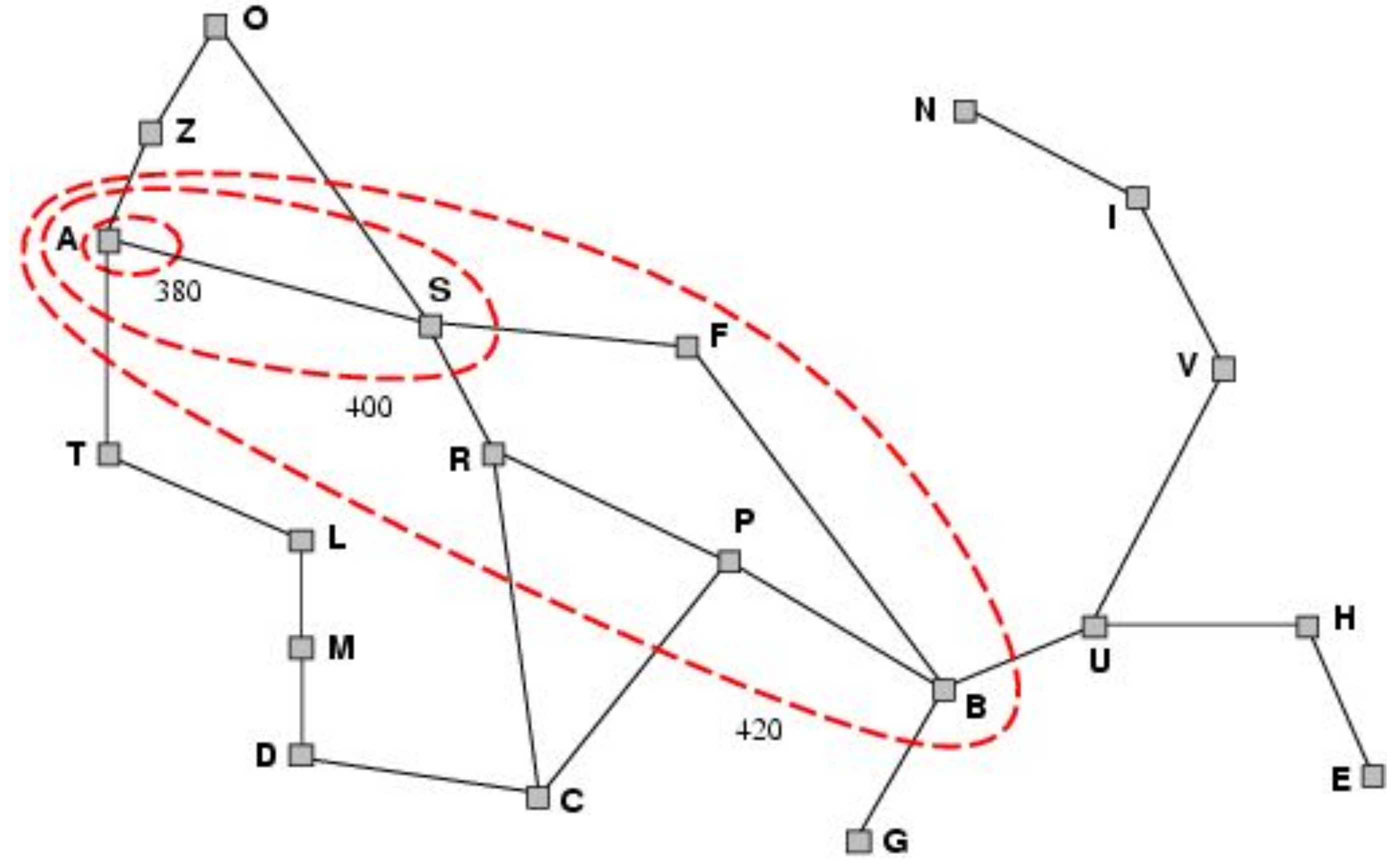
- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1
- h_2 is better for search

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Optimality of A*

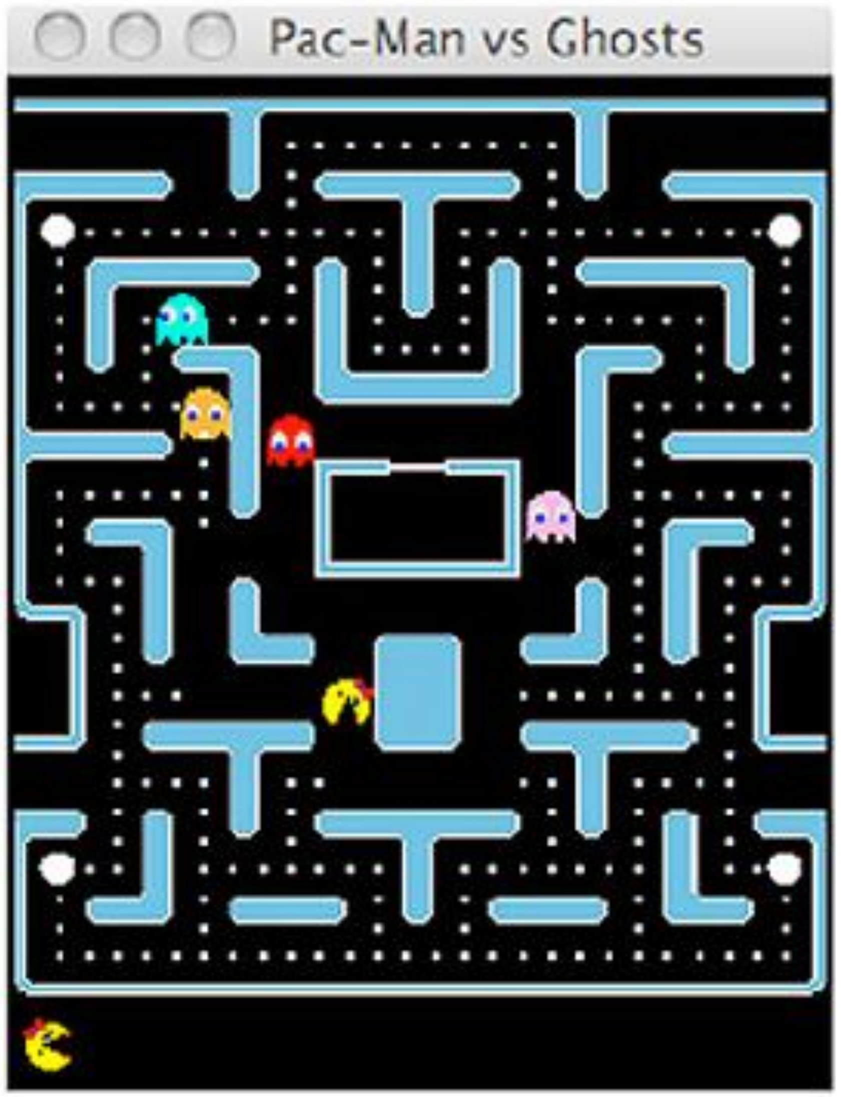
- A* expands nodes in order of increasing f value
- Gradually adds "f-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



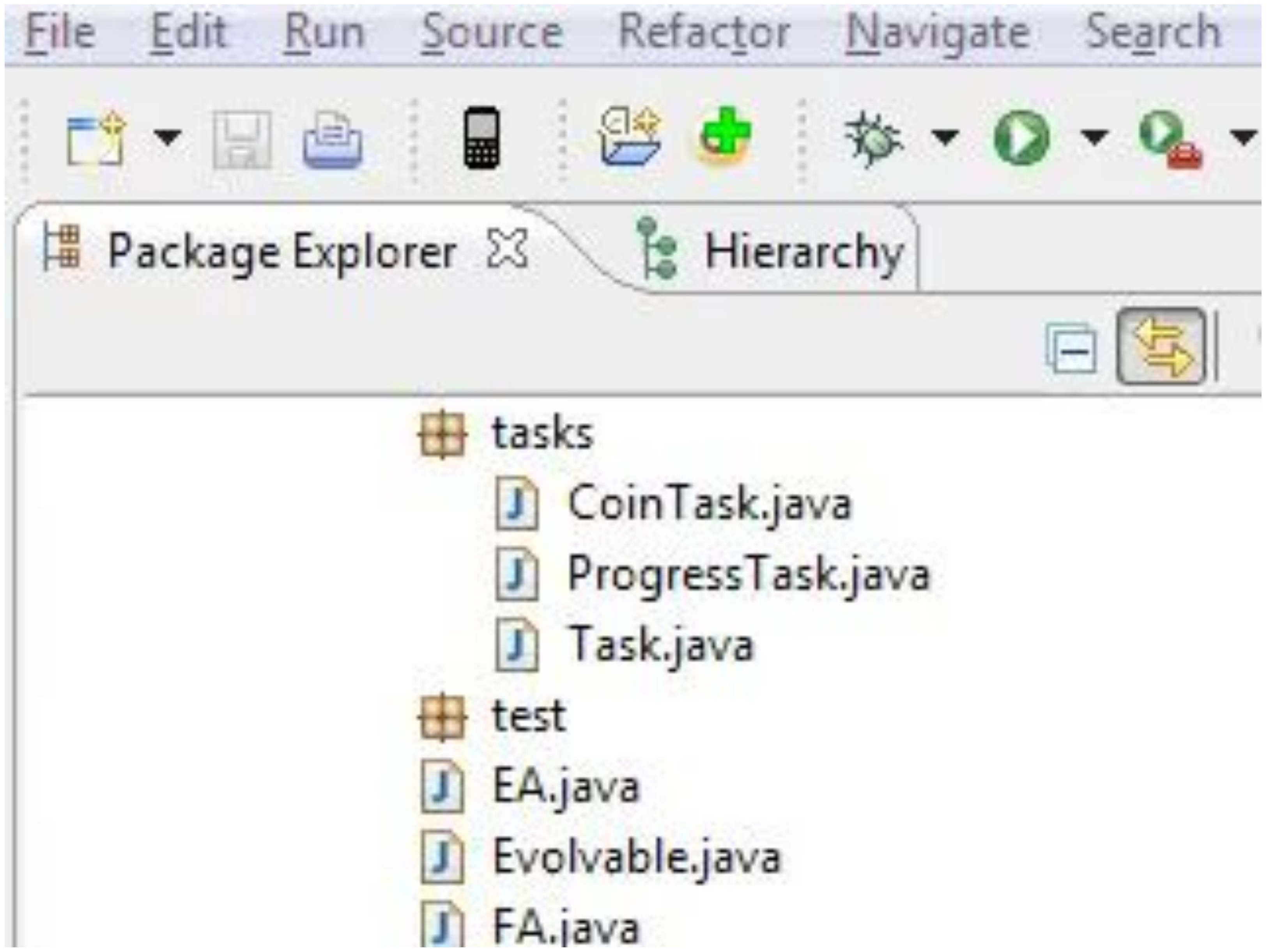
Properties of A*

- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time? Exponential in the worst case
- Space? Keeps all nodes in memory
- Optimal? Yes

Admissible heuristics for…



A* in Mario



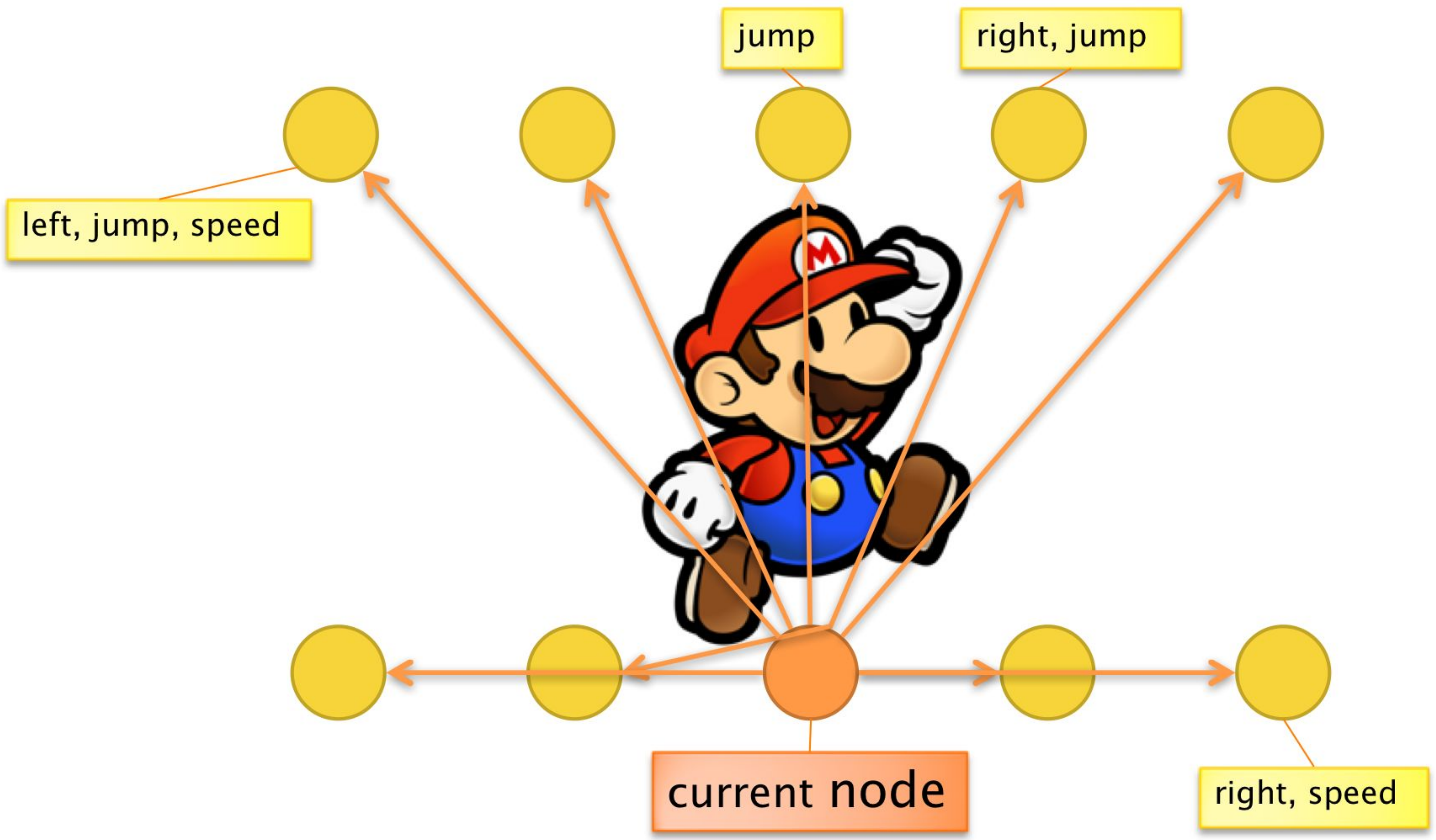
https://www.youtube.com/watch?v=DlkMs4ZHHr8

A* IN MARIO: CURRENT POSITION

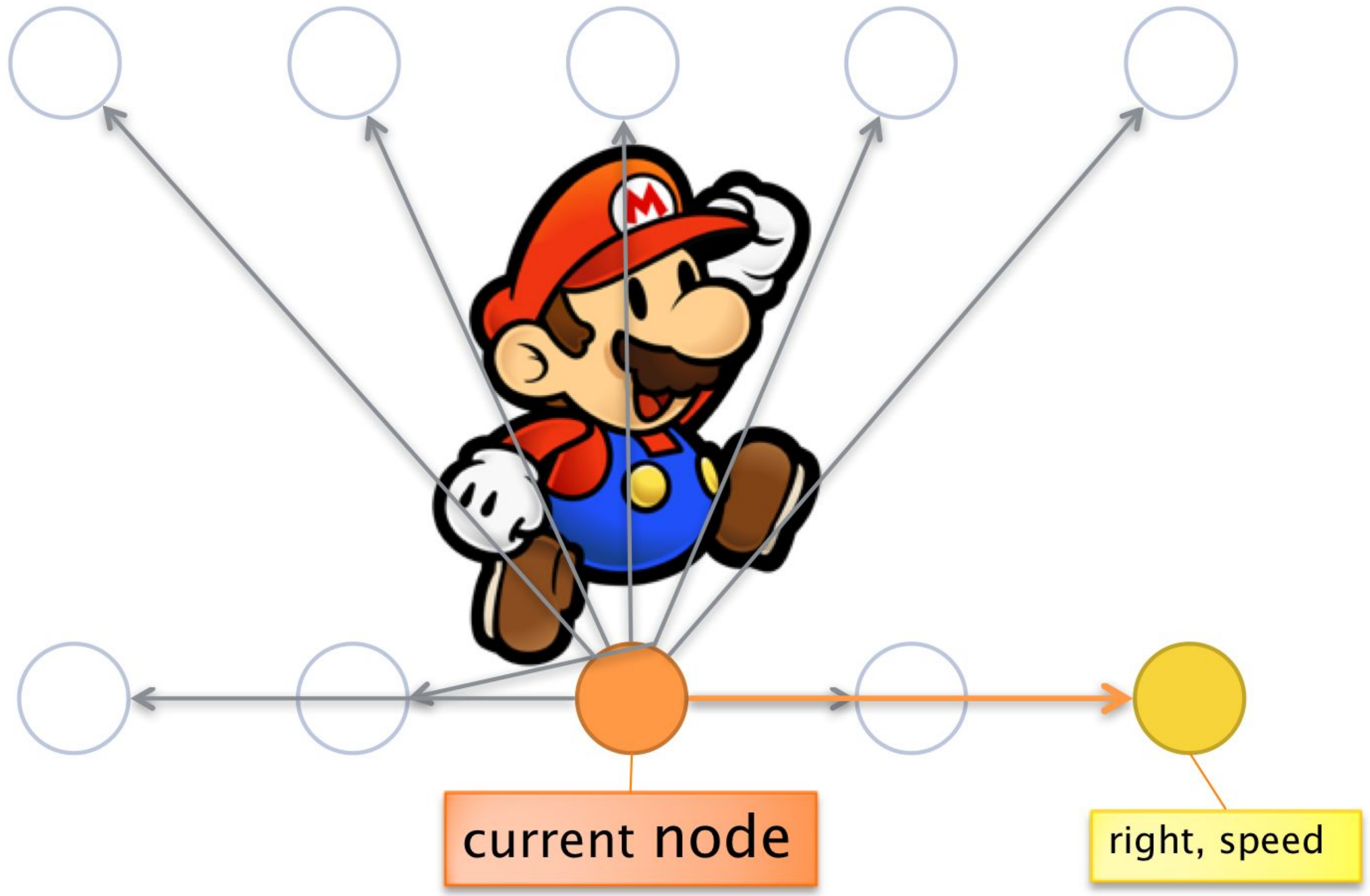
current node

Goal:
right border
of screen →

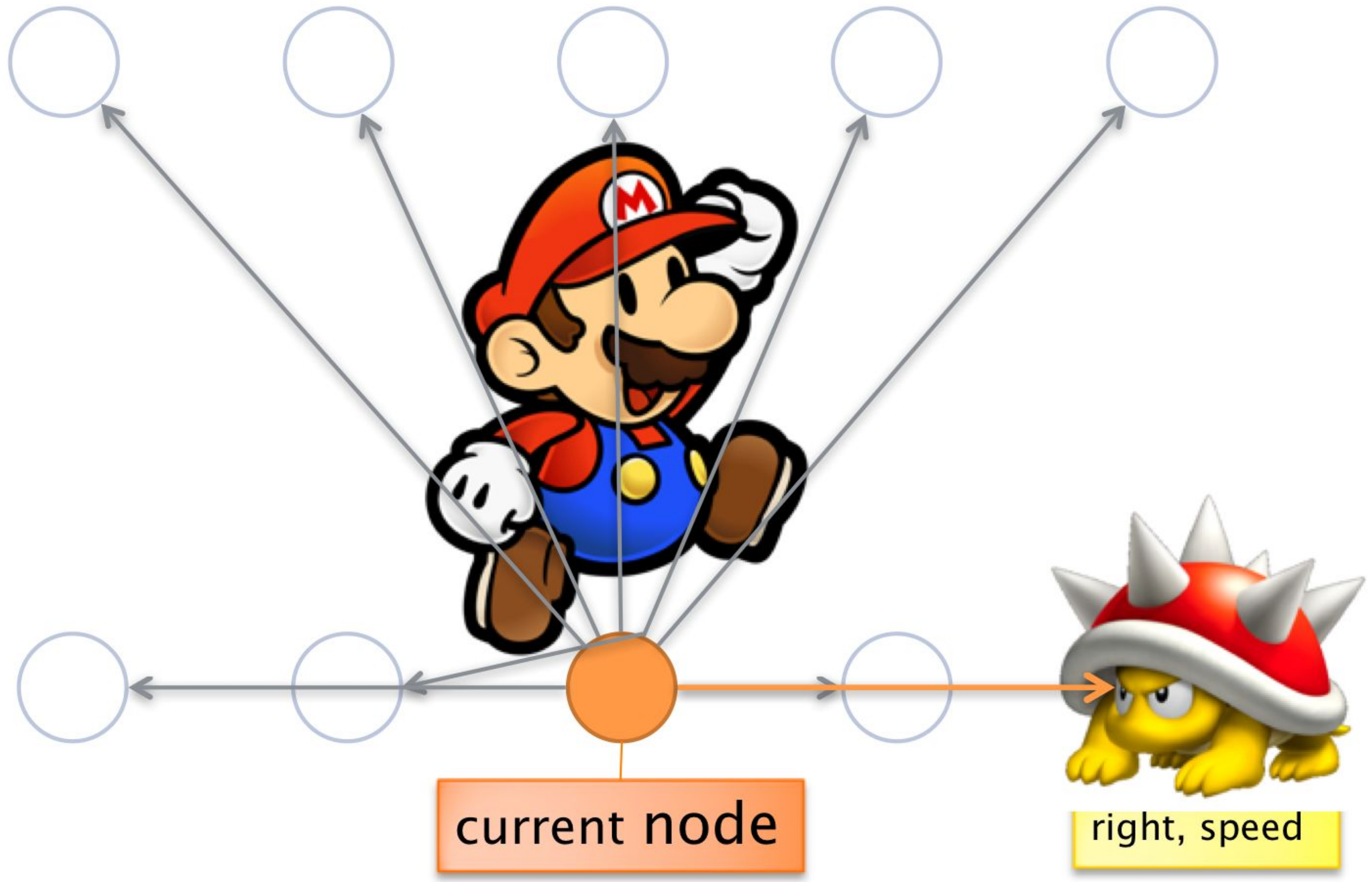
A* IN MARIO: CHILD NODES



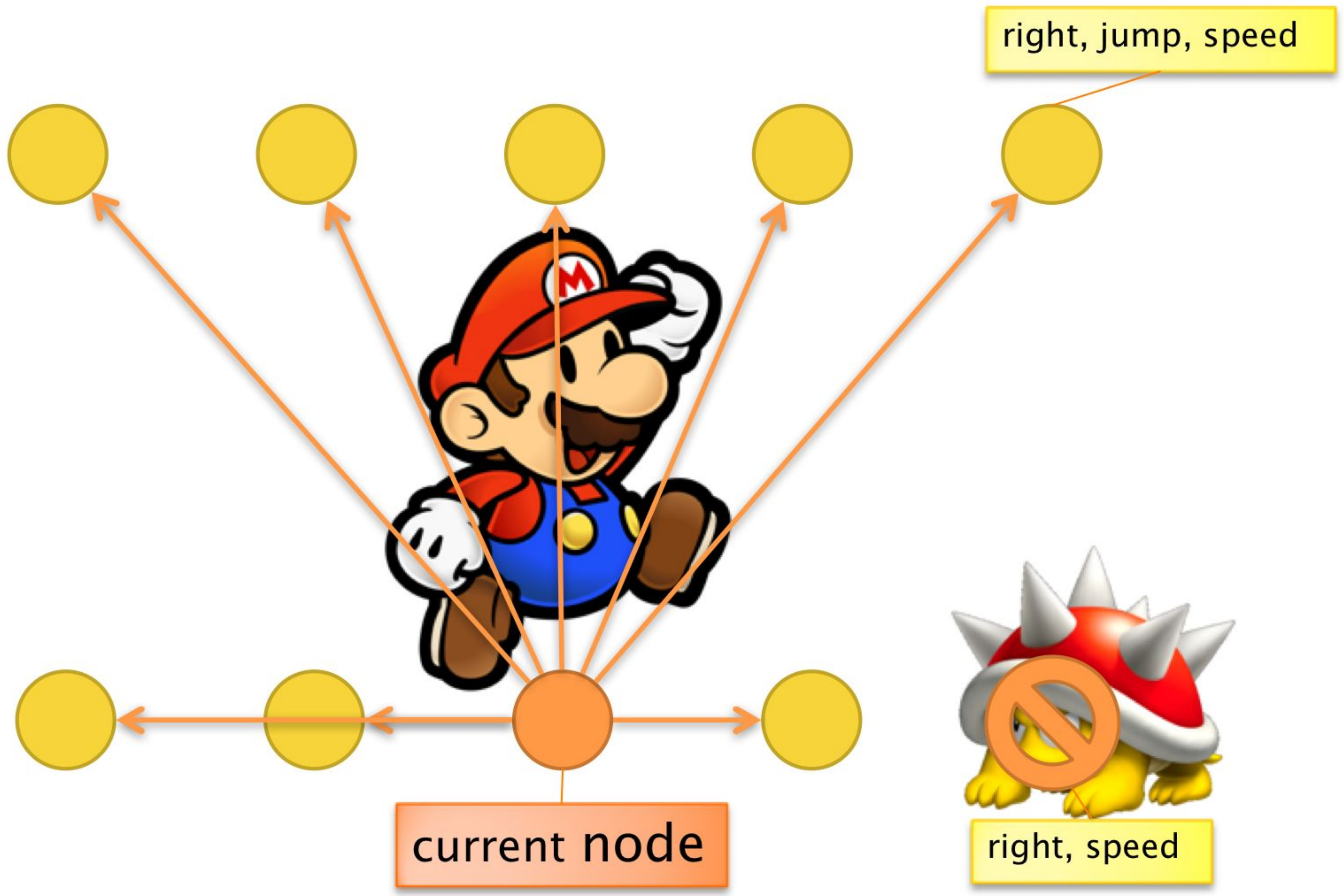
A* IN MARIO: BEST FIRST



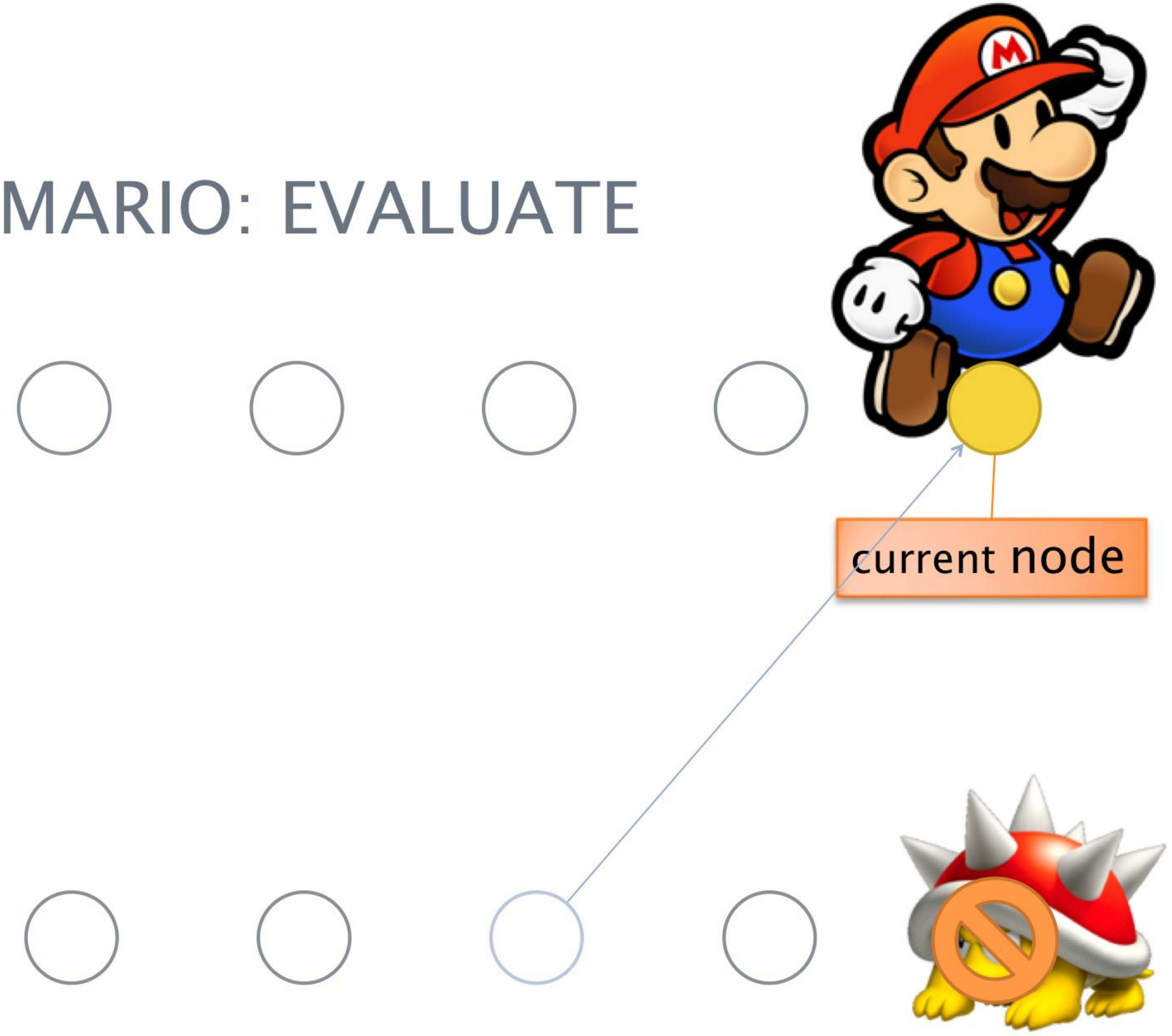
A* IN MARIO: EVALUATE NODE



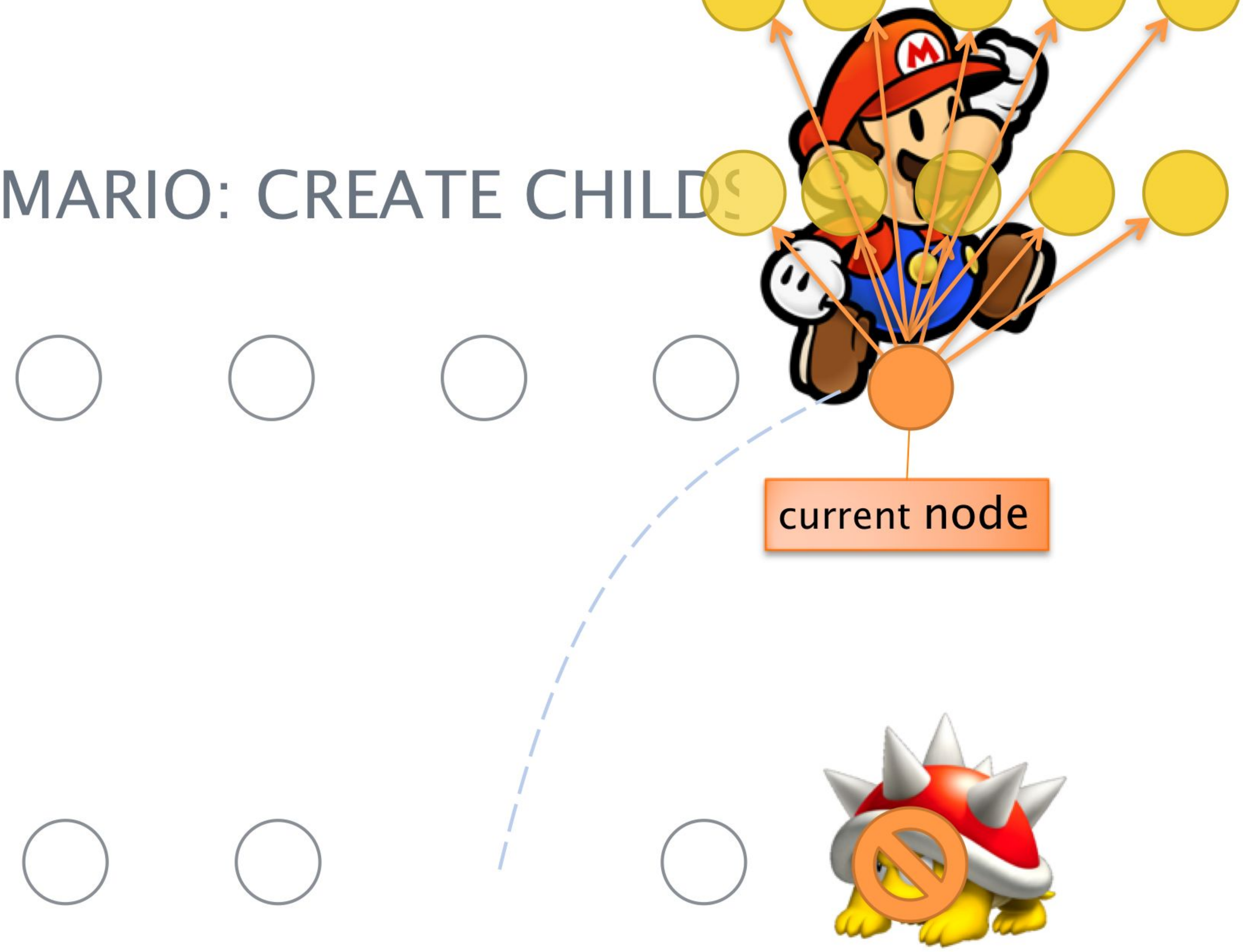
A* IN MARIO: BACKTRACK



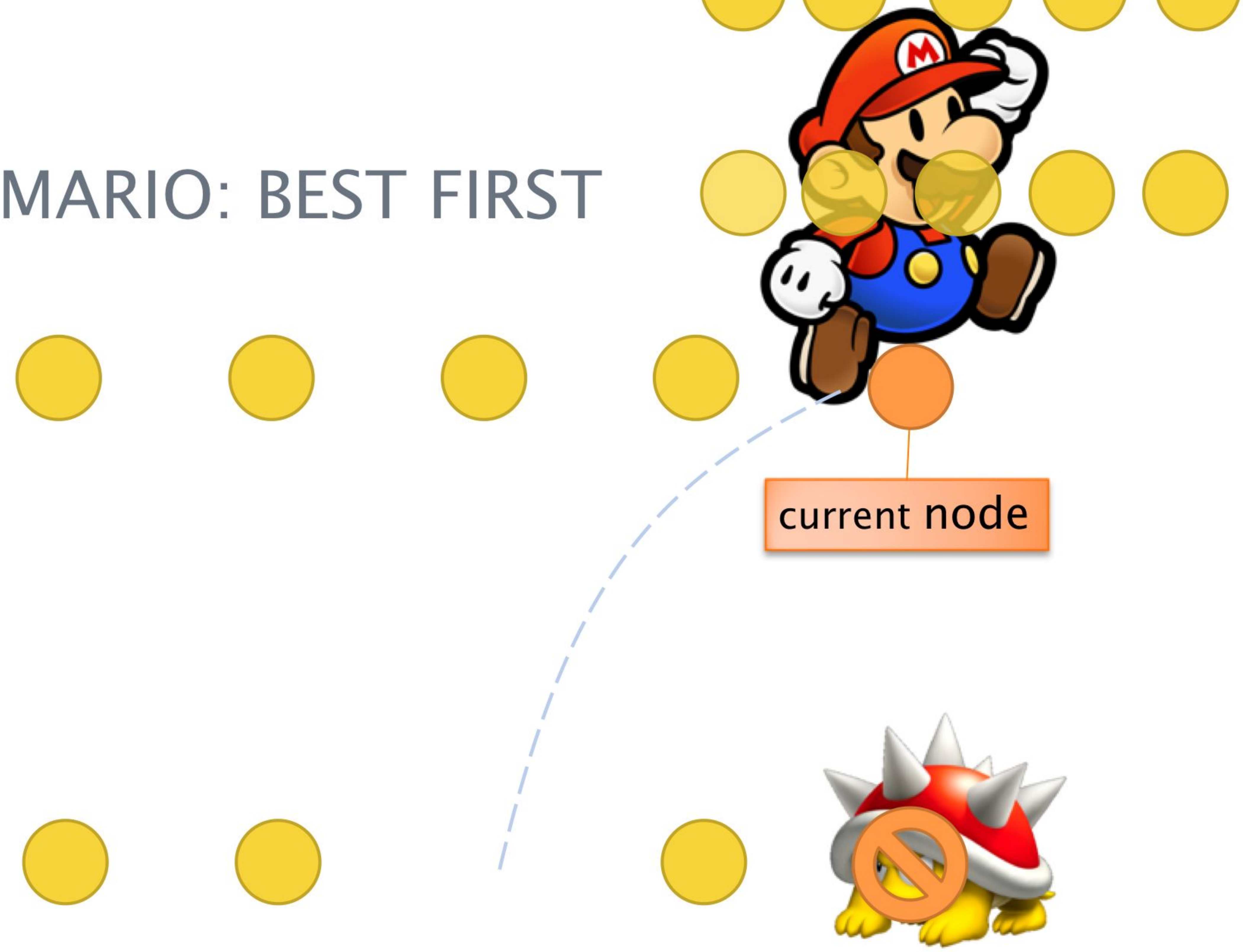
A* IN MARIO: EVALUATE



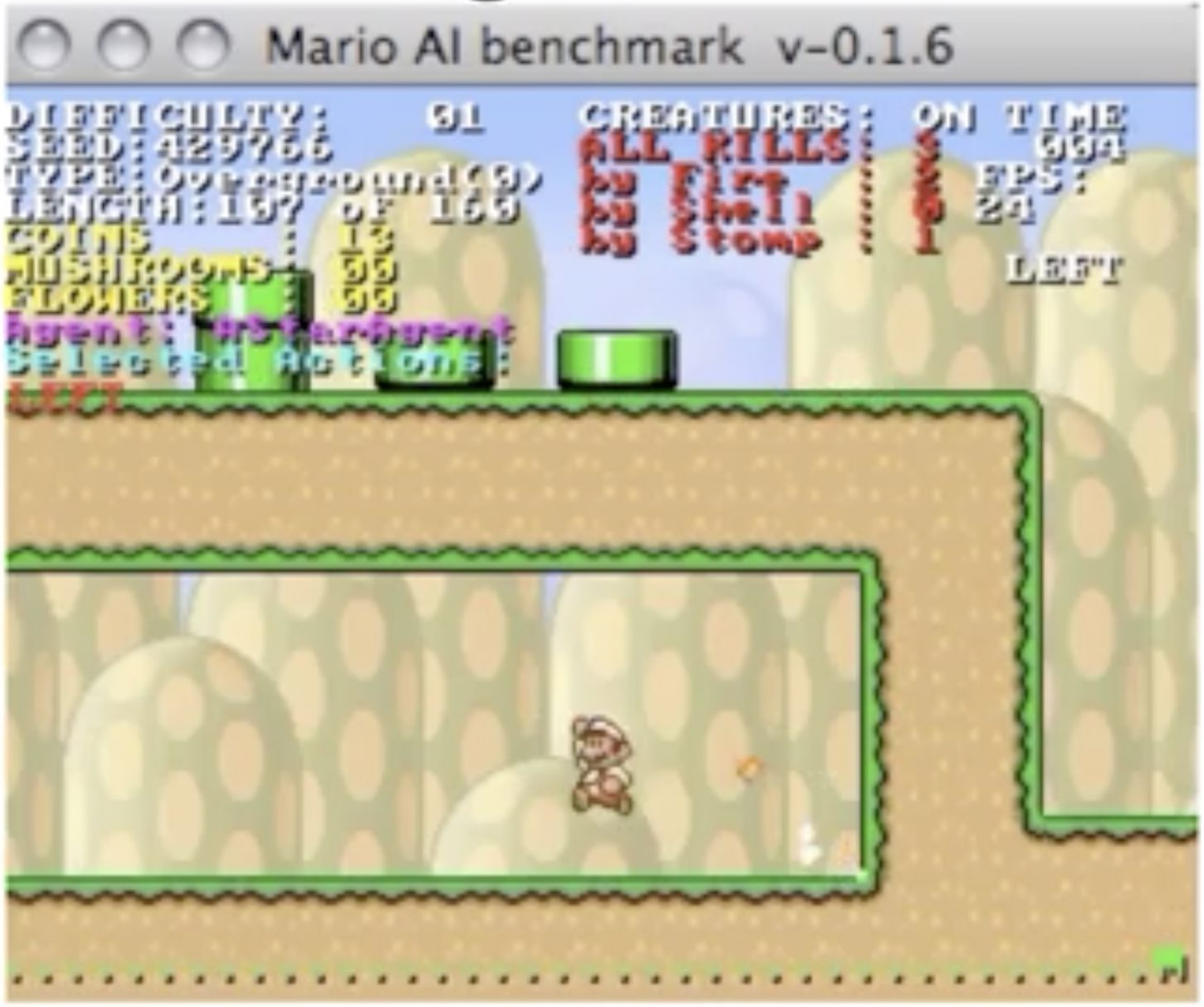
A* IN MARIO: CREATE CHILDS



A* IN MARIO: BEST FIRST



Running into a wall



```
DIFFICULTY:      01      CREATURES:  ON   TIME
SEED:429766              ALL KILLS :  5    994
TYPE:Overground(0)       by Fire   :  [illegible]  FPS:
LENGTH:157 of 166        by Shell  :  [illegible]  24
COINS      :  13         by Stomp  :  1
MUSHROOMS  :  00                           LEFT
FLOWERS    :  00
Agent: AStarAgent
Selected Actions:
LEFT
```

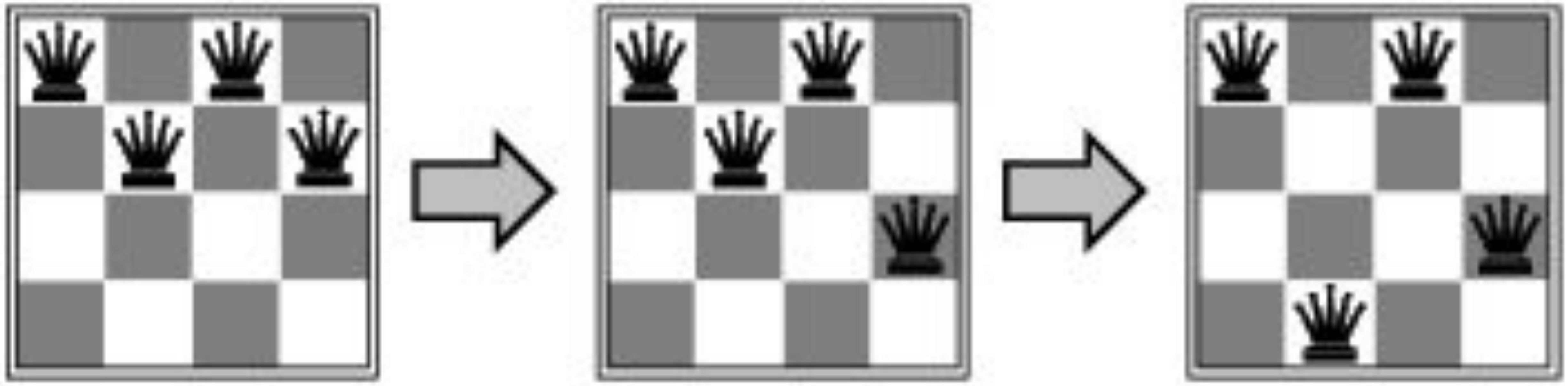
Tree search versus optimization

Local search algorithms

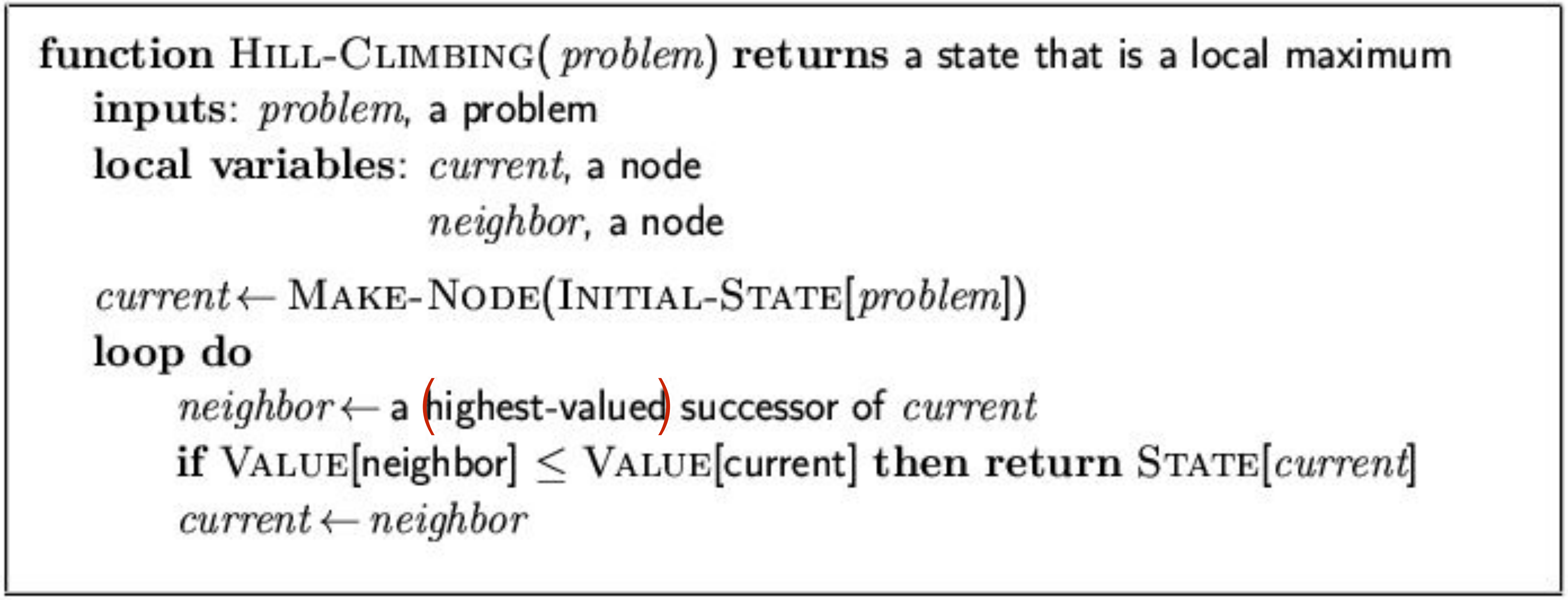
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms: keep a single "current" state, try to improve it

n-queens

- Put n queens on an n × n board with no two queens on the same row, column, or diagonal



Hill-climbing

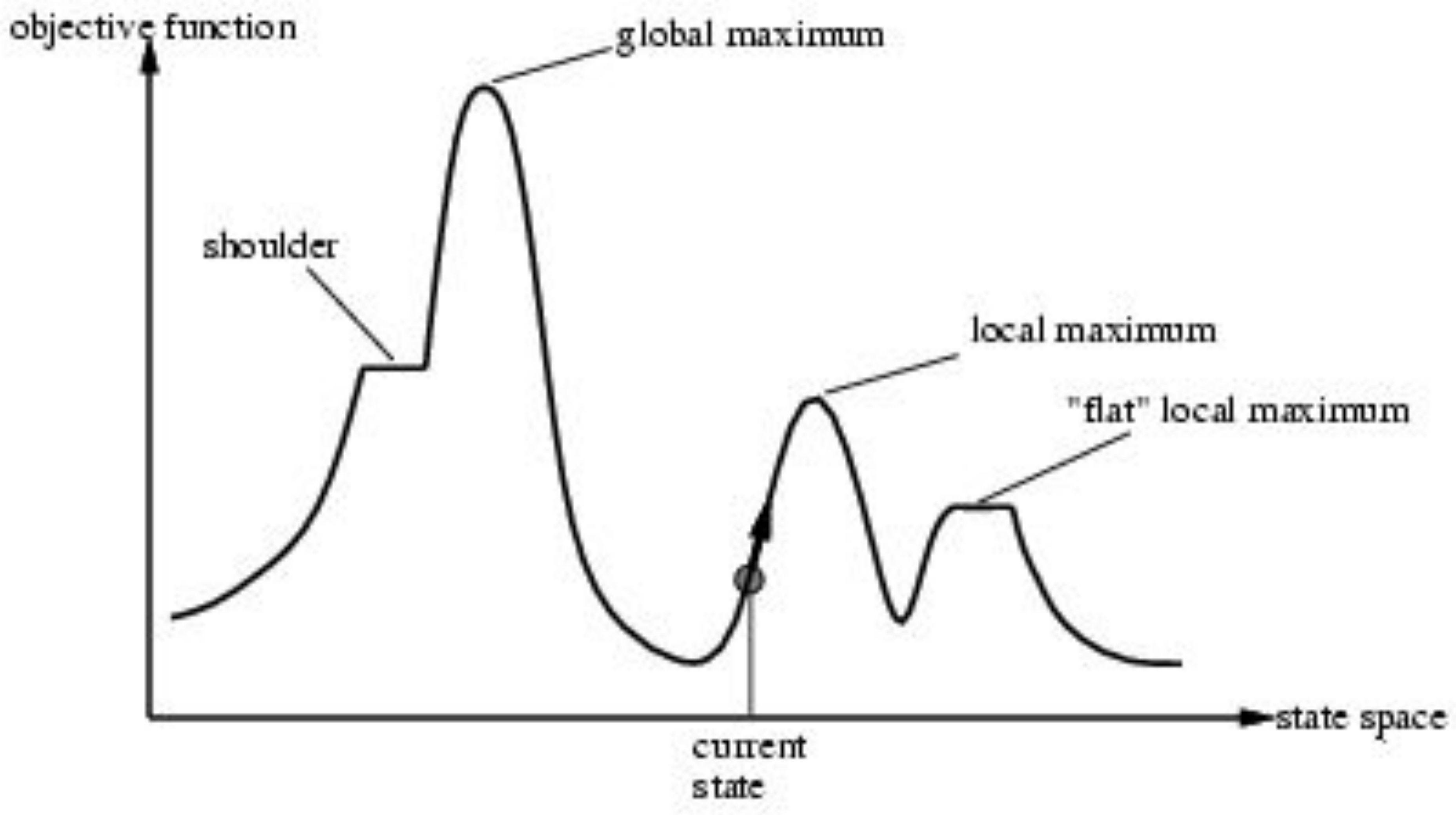


```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    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
```

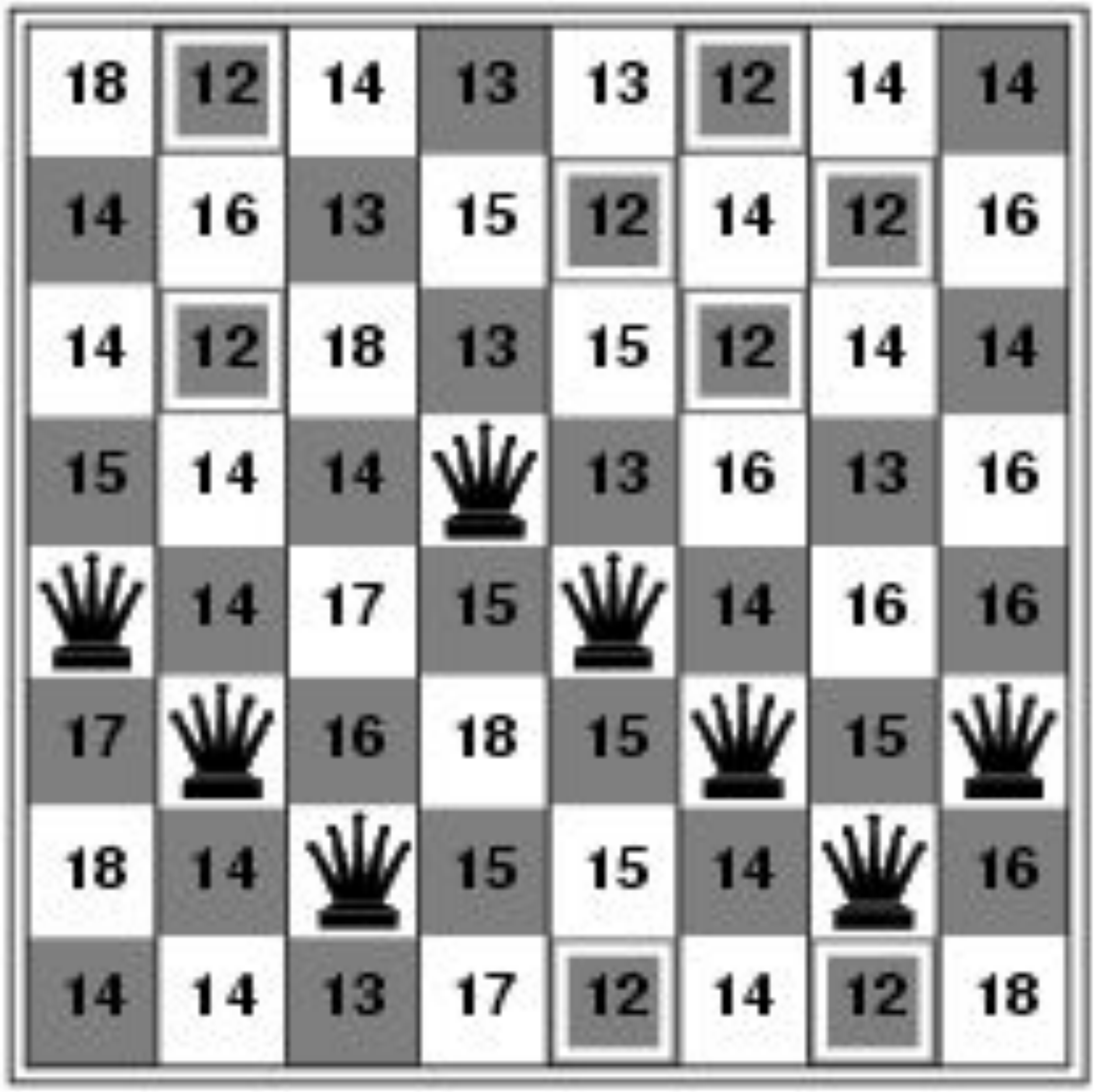
Hill-climbing

- Can get stuck in local maxima/minima

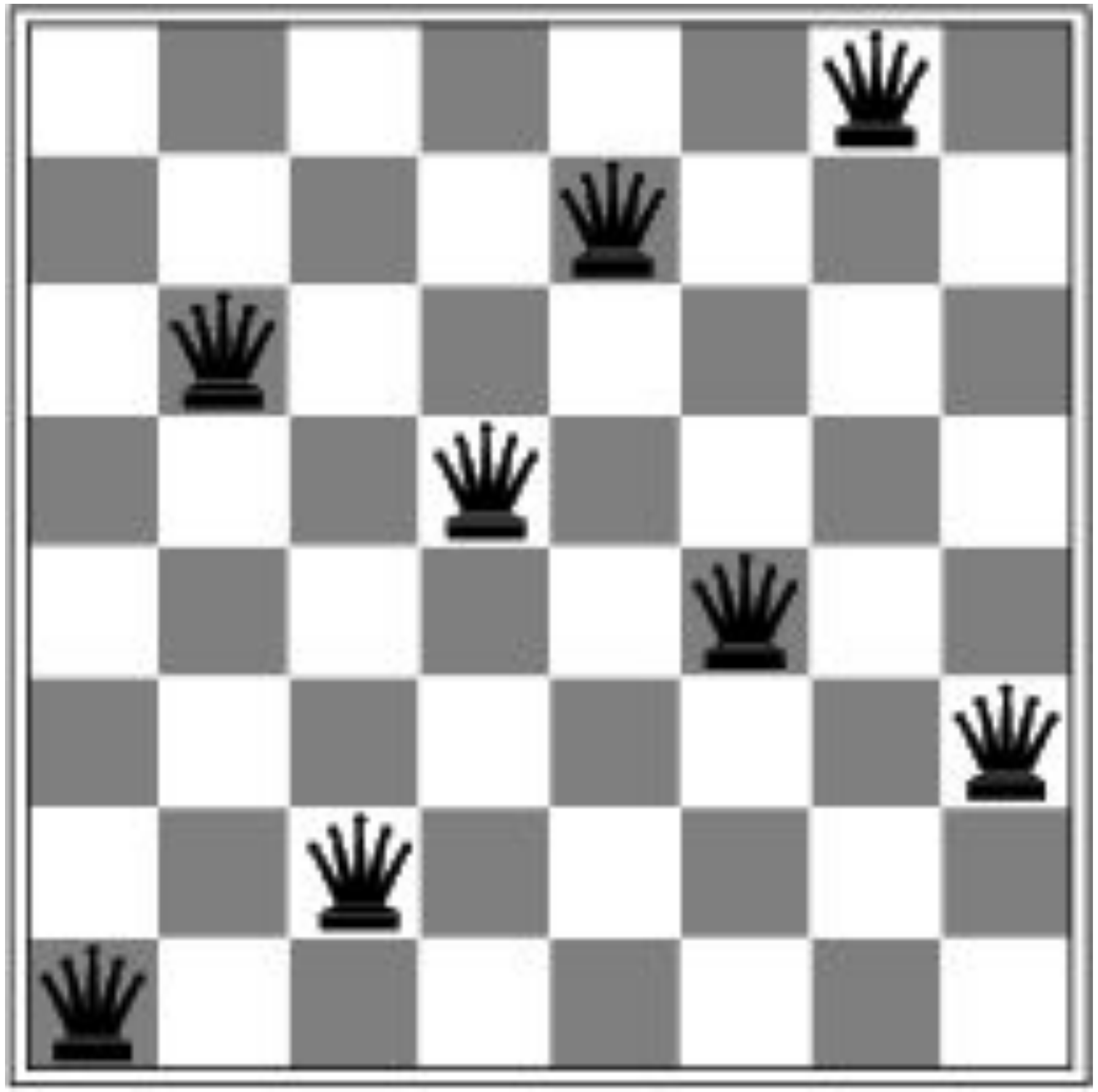


Hill-climbing 8-queens

- *h* = number of pairs of queens that are attacking each other, either directly or indirectly. Here, *h=17*.



Local minimum



Hill-climbing variations

- Simple hill climber: generates one successor, accepts it if better than the current solution

- “Permissive” hill climber: accepts the successor, accepts it if equal to or better than the current solution
-

  Steepest ascent hill climber: generates all successors, chooses the best one

In-class exercise

Optimization in general

- Optimal: no
- Complete: no
- Space: reasonable
- Time: who knows

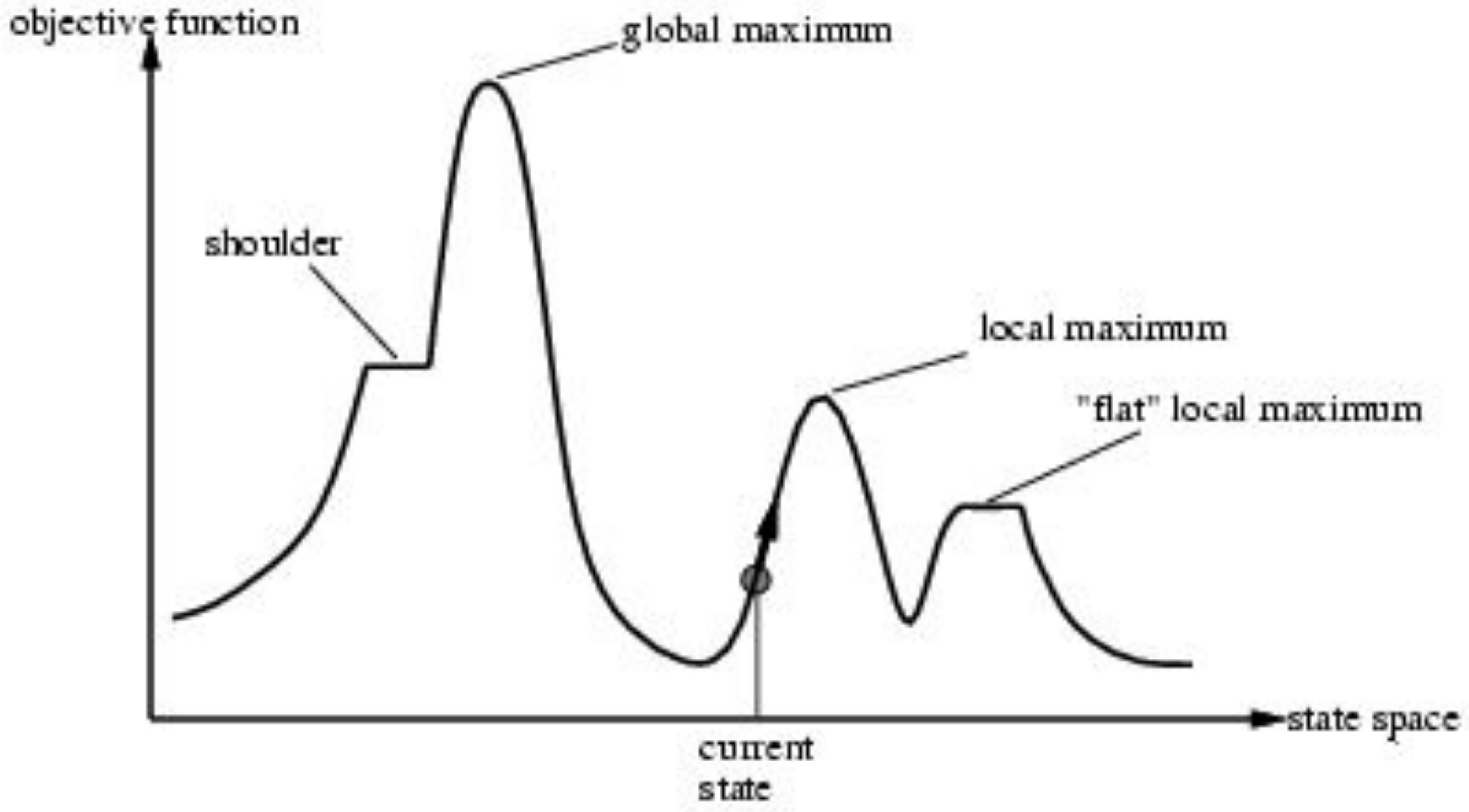
Other ideas...

- Exhaustive search: try all configurations, one after another
- Random search: generate random configurations

Random restarts

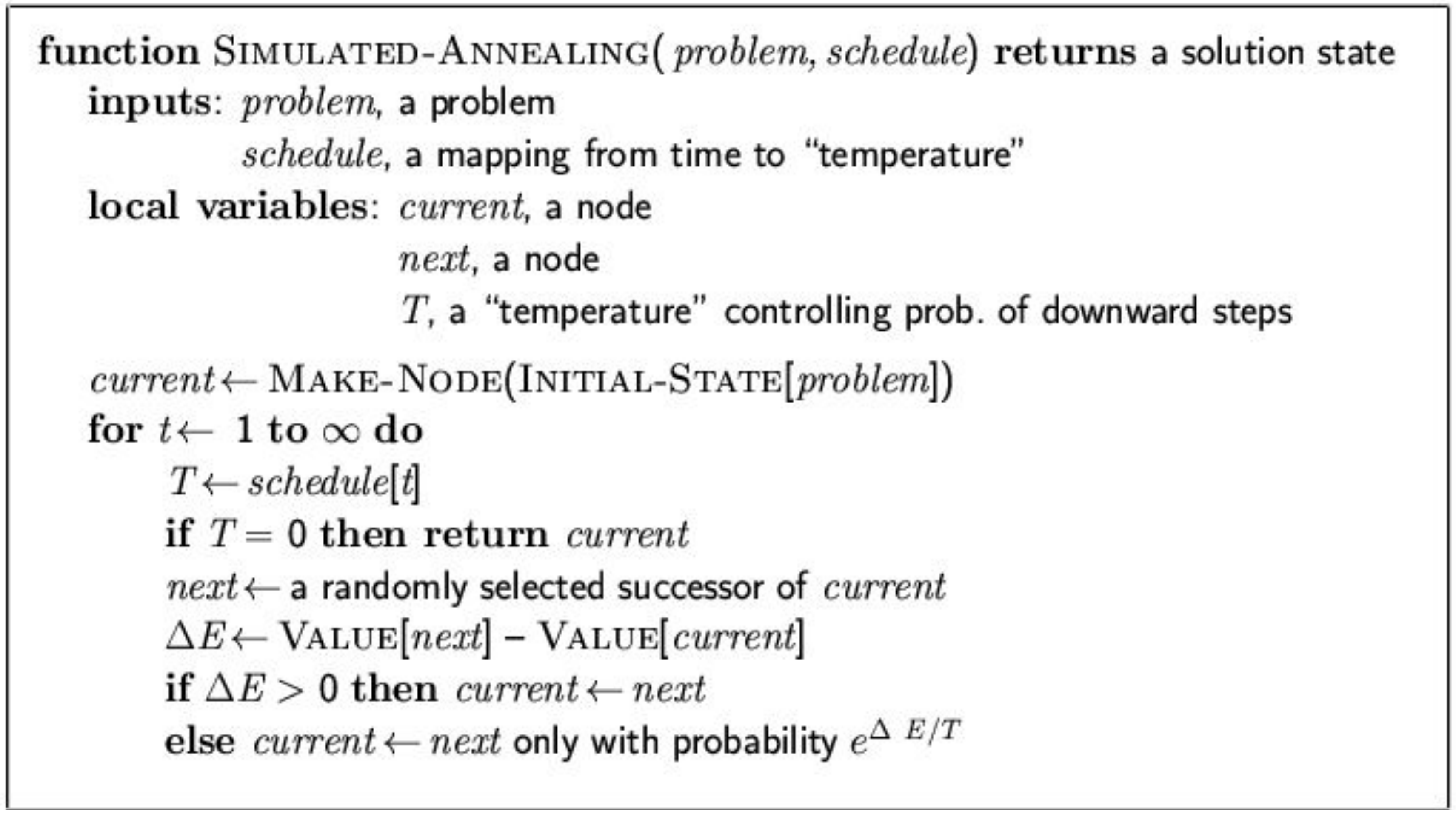
- When you stop making progress, start another hillclimber somewhere else
- Keep the best solution you found so far

Hill-climbing



Simulated annealing

- Do bad moves with decreasing probability



```
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
       $\Delta E$ ← VALUE[next] – VALUE[current]
       if $\Delta E > 0$ then current ← next
       else current ← next only with probability $e^{\Delta E/T}$
```

Simulated annealing

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

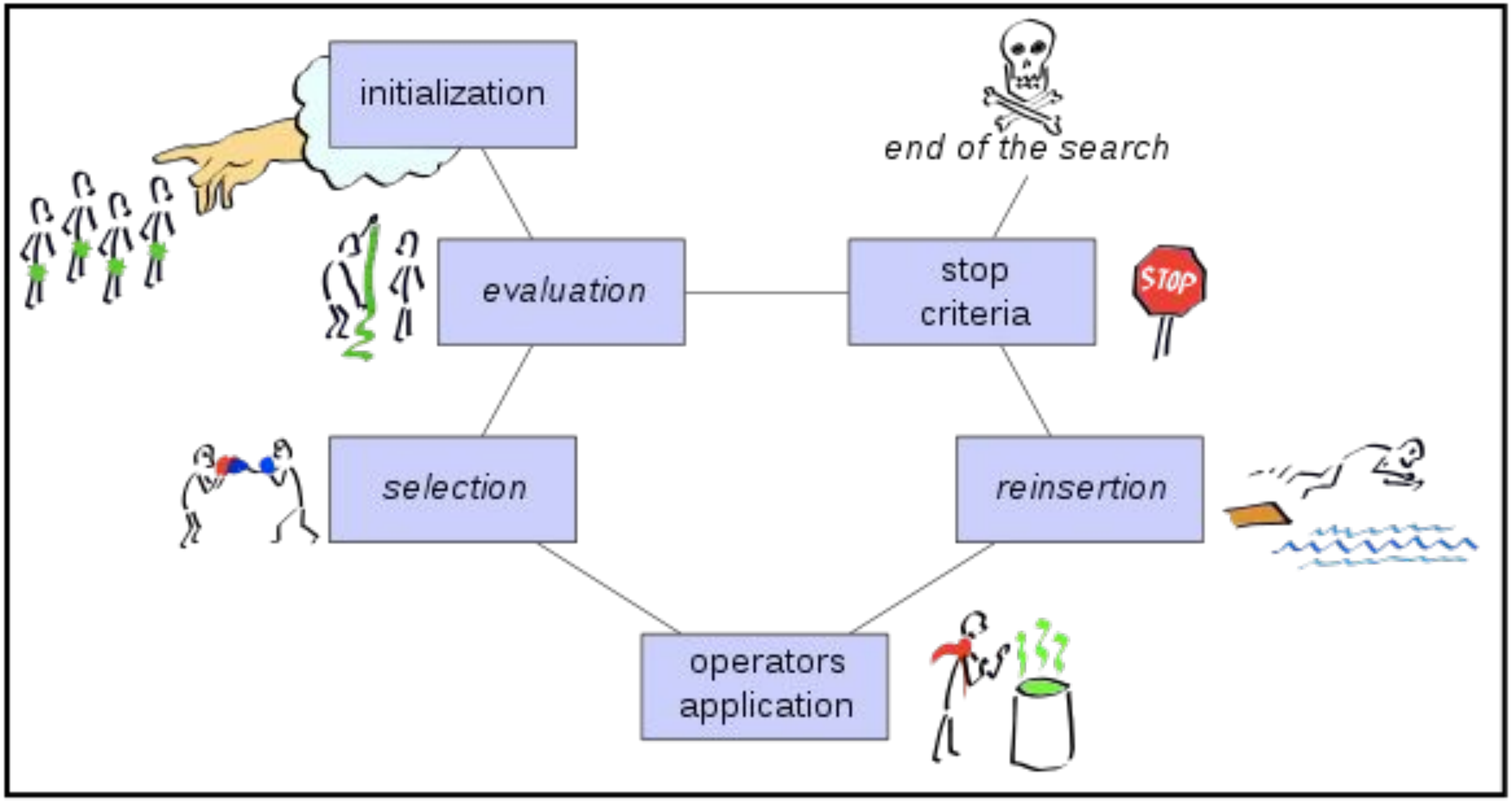
Limitations of local search

- Susceptible to local optima
- No sharing of information between parallel hill-climbing runs
- No possibility to learn from previous search experience

Evolutionary algorithms

- Stochastic global optimisation algorithms
- Inspired by Darwinian natural evolution
- Extremely domain-general, widely used in practice

Evolutionary computation



General schema of an Evolutionary Algorithm (EA)