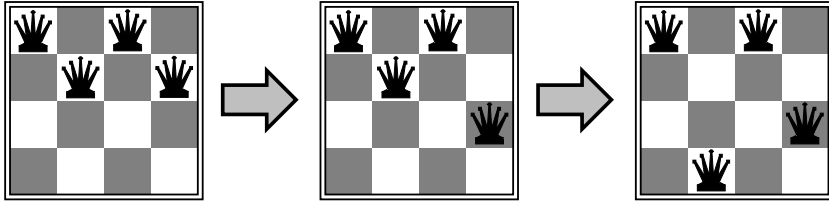


Review: Basic Concepts

Example: n -queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



- Case 1: Consider one (fixed) cell at a time
- Case 2: Consider one row at a time
- Case 3: Consider one queen at a time

Artificial Intelligence

Informed Search and Exploration

Readings: Chapter 4 of Russell & Norvig.

Review: Tree search

```

function TREE-SEARCH(problem, fringe)
  returns a solution, or failure
  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] applied
  to STATE(node) succeeds return node
  fringe ← INSERTALL(EXPAND(node,
  problem), fringe)
  
```

A strategy is defined by picking the *order of node expansion*

n -queens

- Case 1: Consider one (fixed) cell at a time
- Case 2: Consider one row at a time
- Case 3: Consider one queen at a time

	case1	case 2	case 3
Branching factor:	2	n	n^2
Maximal depth:	n^2	n	n
State space:	2^{n^2}	n^n	n^{2^n}

Informed Search Strategies

- Uninformed search strategies look for solutions by *systematically* generating new states and checking each of them against the goal.
- This approach is very inefficient in most cases.
- Most successor states are “obviously” a bad choice.
- Such strategies do not know that because they have minimal *problem-specific knowledge*.
- **Informed** search strategies exploit problem-specific knowledge as much as possible to drive the search.
- They are almost always *more efficient* than uninformed searches and often also *optimal*.

Uninformed Search Strategies

Strategies	Time	Space	Complete?
Breadth-first Search	$O(b^d)$	$O(b^d)$	Yes
Depth-first Search	$O(b^m)$	$O(bm)$	No
Depth-limited Search	$O(b^l)$	$O(bl)$	No
Iterative Deepening Search	$O(b^d)$	$O(bd)$	Yes
Uniform Cost Search	$O(b^d)$	$O(b^d)$	Yes

where b is the branching factor, d is the depth of the shallowest solution, m is the length of the longest path, l is the limit set by the user.

Informed Search Strategies

- f is typically an *imperfect measure* of the goodness of the node. The right choice of nodes is not always the one suggested by f .
- It is possible to build a perfect evaluation function, which will always suggest the right choice.
- How?
- Why don't we use perfect evaluation functions then?

Informed Search Strategies

Main Idea

- Use the knowledge of the problem domain to build an **evaluation function** f .
- For every node n in the search space, $f(n)$ quantifies the *desirability* of expanding n in order to reach the goal.
- Then use the desirability value of the nodes in the fringe to decide which node to expand next.

Best-First Search

- Idea: use an *evaluation function* for each node to estimate of “desirability”
- Strategy: Always expand most desirable unexpanded node
- Implementation: *fringe* is a priority queue sorted in decreasing order of desirability
- Special cases:
 - greedy search
 - A* search

Standard Assumptions on Search Spaces

- The cost of a node increases with the node’s depth.
- Transitions costs are non-negative and bounded below. That is, there is a $\delta > 0$ such that the cost of each transition is $\geq \delta$.
- Each node has only finitely-many successors.

Note: There *are* problems that do *not* satisfy one or more of these assumptions.

Best-first Search Strategies

- There is a *whole family* of best-first search strategies, each with a different evaluation function.
- Typically, strategies use estimates of the cost of reaching the goal and try to *minimize* it.
- Uniform Search also tries to minimize a cost measure.
- Is it a best-first search strategy?
- Not in spirit, because the evaluation function should incorporate a *cost estimate of going from the current state to the closest goal state*.

Implementing Best-first Search

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
inputs: problem, a problem
         Eval-Fn, an evaluation function

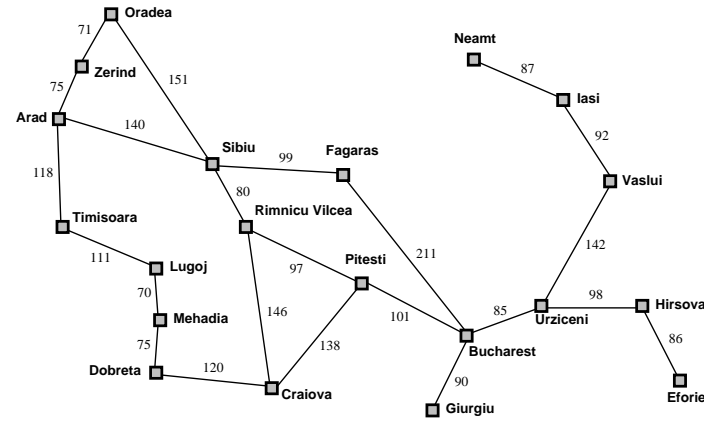
Queueing-Fn  $\leftarrow$  a function that orders nodes by EVAL-FN
return GENERAL-SEARCH(problem, Queueing-Fn)
```

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns a solution, or failure
nodes  $\leftarrow$  MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
loop do
  if nodes is empty then return failure
  node  $\leftarrow$  REMOVE-FRONT(nodes)
  if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
  nodes  $\leftarrow$  QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
end
```

Greedy Search

- Evaluation function $h(n)$ (heuristic) is an estimate of cost from n to the closest goal
E.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy search expands the node that *appears* to be closest to goal

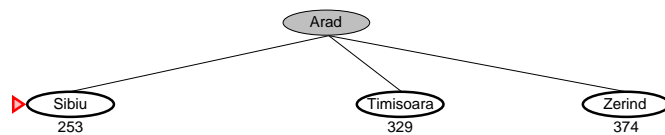
Romania with Step Costs in km



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

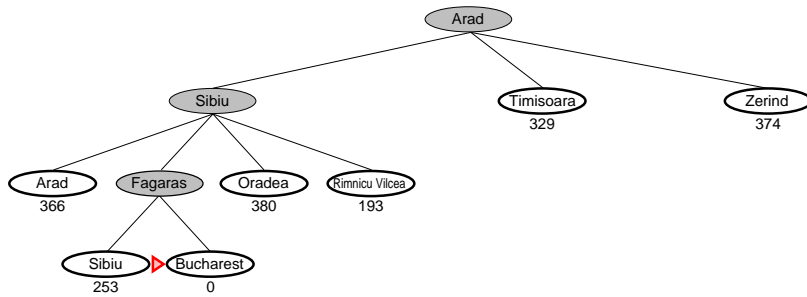
Greedy Search Example



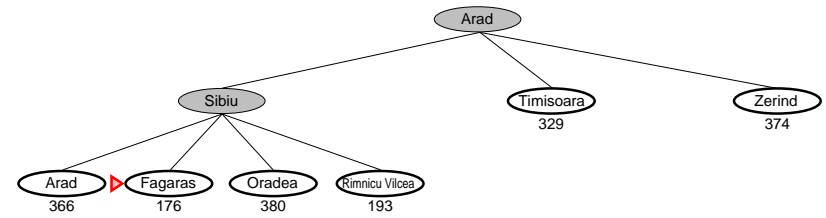
Greedy Search Example



Greedy search example



Greedy Search Example



Properties of greedy search

- Complete?? No—can get stuck in loops, e.g., with Oradea as goal, Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking
- Time??

Properties of Greedy Search

- Complete??

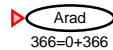
Properties of Greedy Search

- Complete?? No—can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking
- Time?? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?? $O(b^m)$ —keeps all nodes in memory
- Optimal??

Properties of Greedy Search

- Complete?? No—can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking
- Time?? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space??

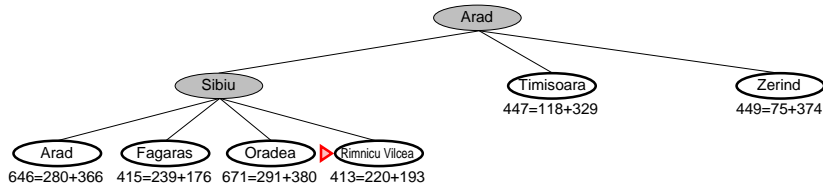
A* Search Example



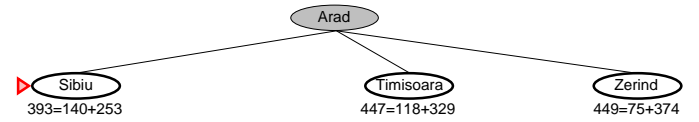
A* Search

- 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 to goal from n
 $f(n)$ = estimated total cost of path through n to goal
- A* search uses an *admissible* heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n .
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G .)
E.g., $h_{\text{SLD}}(n)$ never overestimates the actual road distance
- Theorem: A* search is optimal

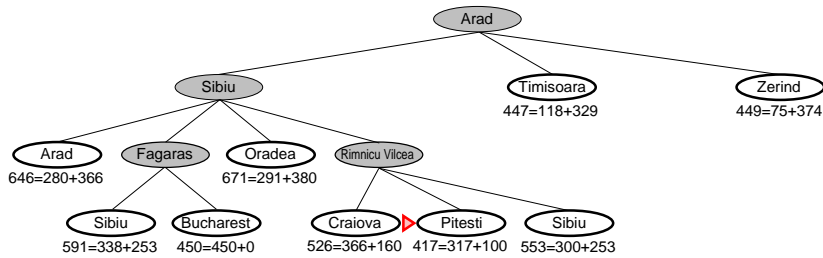
A* Search Example



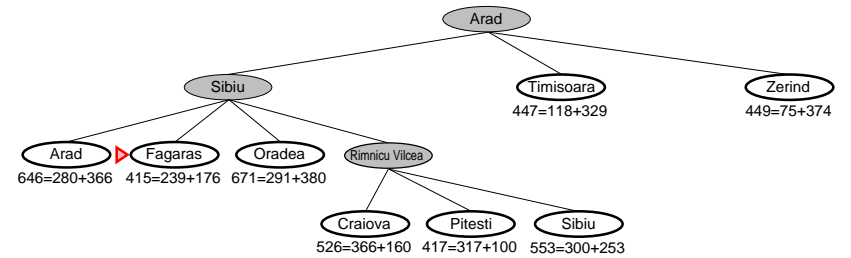
A* Search Example



A* Search Example

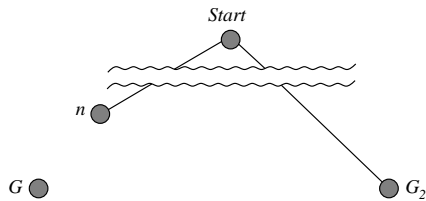


A* Search Example



Optimality of A* (standard proof)

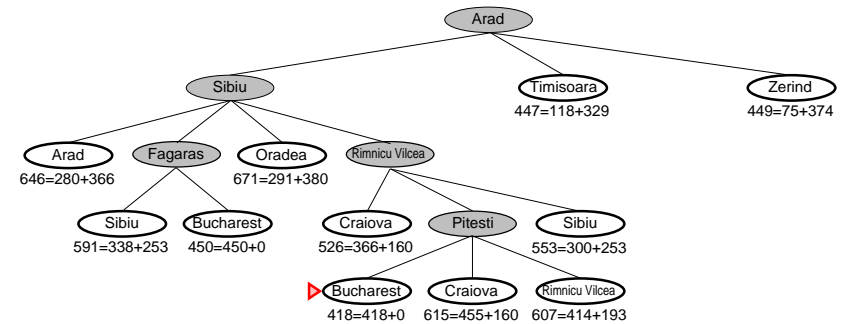
Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1 .



$$\begin{aligned}
 f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
 &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
 &\geq f(n) && \text{since } h \text{ is admissible}
 \end{aligned}$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

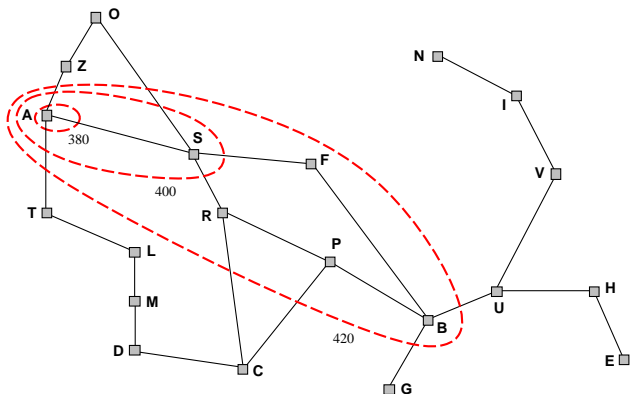
A* Search Example



Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing f value*
Gradually adds " f -contours" of nodes (cf. breadth-first adds layers)

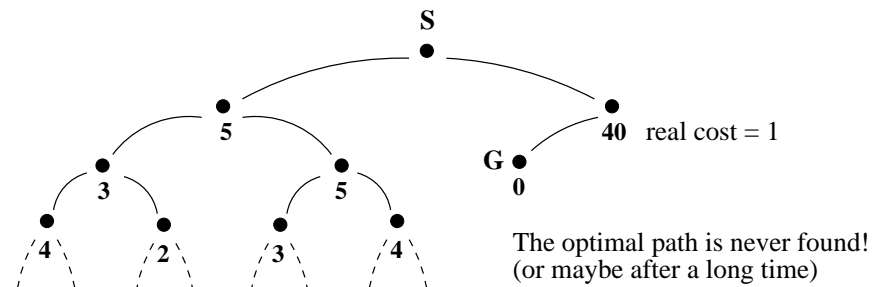
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



A* Search with Admissible Heuristic

If h is admissible, $f(n)$ never overestimates the actual cost of the best solution through n .

Overestimates are dangerous



Properties of A*

- Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time??

Properties of A*

- Complete??

Properties of A*

- Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time?? Exponential in [relative error in $h \times$ length of soln.]
- Space?? Keeps all nodes in memory
- Optimal??

Properties of A*

- Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time?? Exponential in [relative error in $h \times$ length of soln.]
- Space??

Proof of lemma: Consistency

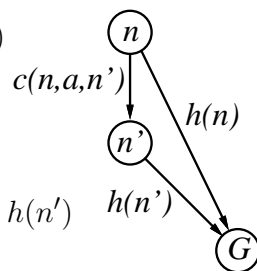
A heuristic is *consistent* if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

I.e., $f(n)$ is nondecreasing along any path.



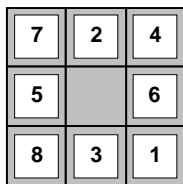
Properties of A*

- Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time?? Exponential in [relative error in $h \times$ length of soln.]
- Space?? Keeps all nodes in memory
- Optimal?? Yes—cannot expand f_{i+1} until f_i is finished
 A* expands all nodes with $f(n) < C^*$
 A* expands some nodes with $f(n) = C^*$
 A* expands no nodes with $f(n) > C^*$

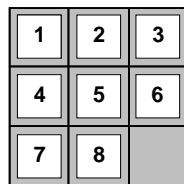
Admissible Heuristics

For the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance (i.e., number of squares from desired location of each tile)



Start State



Goal State

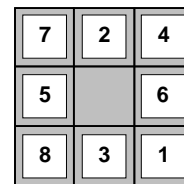
• $h_1(S) = ??$ 7

• $h_2(S) = ??$ 4+0+3+3+1+0+2+1 = 14

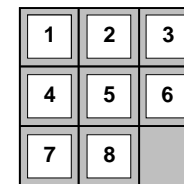
Admissible Heuristics

For the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance (i.e., number of squares from desired location of each tile)



Start State



Goal State

• $h_1(S) = ??$

• $h_2(S) = ??$

Optimality/Completeness of A* Search

If the problem is solvable, A* always finds an optimal solution when

- the standard assumptions are satisfied,
- the heuristic function is admissible.

A* is **optimally efficient** for any heuristic function h : No other optimal strategy expands fewer nodes than A*.

Dominance

- **Definition:** If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 **dominates** h_1 .
- For 8-puzzle, h_2 indeed dominates h_1 .
 - $h_1(n)$ = number of misplaced tiles
 - $h_2(n)$ = total Manhattan distance
- If h_2 dominates h_1 , then h_2 is better for search.
- For 8-puzzle, search costs:

$d = 14$ IDS = 3,473,941 nodes

A*(h_1) = 539 nodes

A*(h_2) = 113 nodes

$d = 24$ IDS \approx 54,000,000,000 nodes

A*(h_1) = 39,135 nodes

A*(h_2) = 1,641 nodes

Relaxed Problems

- Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the 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
- Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Complexity of A* Search

- **Worst-case time complexity:** still exponential ($O(b^d)$) unless the error in h is bounded by the logarithm of the actual path cost.
That is, unless

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

where $h^*(n)$ = actual cost from n to goal.

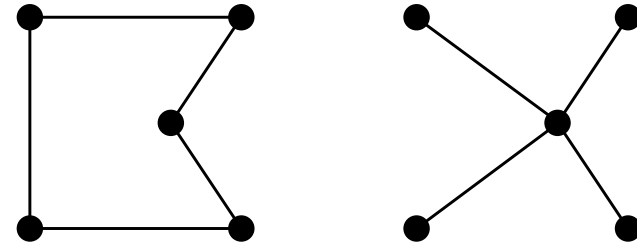
- **Worst-Case Space Complexity:** $O(b^m)$ as in greedy best-first.
- A* generally runs out of memory before running out of time. (Improvements: IDA*, SMA*).

Iterative Improvement Algorithms

- In many optimization problems, *path* is irrelevant; the goal state itself is the solution
- Then state space = set of “complete” configurations; find *optimal* configuration, e.g., TSP or, find configuration satisfying constraints, e.g., timetable
- In such cases, can use *iterative improvement* algorithms; keep a single “current” state, try to improve it
- Constant space, suitable for online as well as offline search

Relaxed Problems

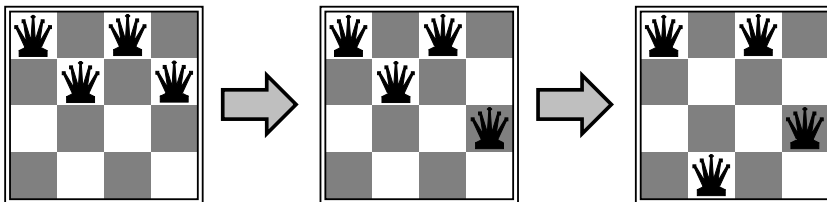
Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

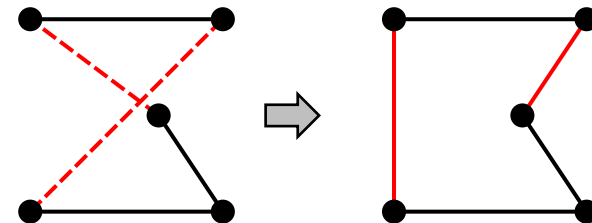
Local Search Example: n -queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts



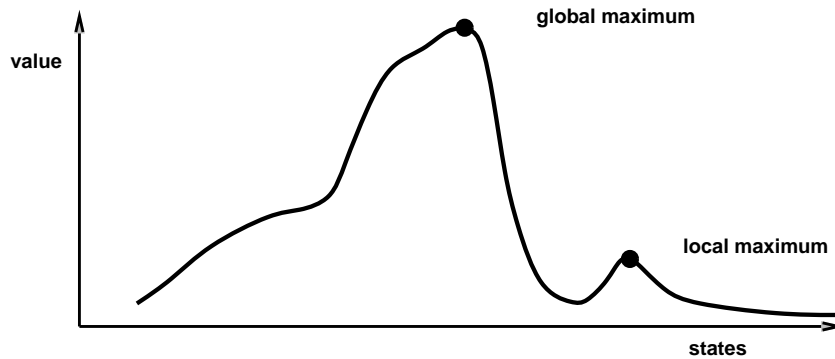
Local Search Example: TSP

- TSP: Travelling Salesperson Problem
- Start with any complete tour, perform pairwise exchanges



Hill-Climbing

Problem: depending on initial state, can get stuck on local maxima



In continuous spaces, problems w/ choosing step size, slow convergence

Hill-Climbing (or Gradient Descent)

“Like climbing Everest in thick fog with amnesia”

```
function Hill-Climbing(problem) return state
node: current, neighbor;
current := Make-Node(Initial-State(problem));
loop do
    neighbor := highest-value-successor(current)
    if (Value(neighbor) < Value(current))
        then return State(current)
        else current := neighbor
    end loop
end function
```

The returned state is a local maximum state.

Simulated Annealing Algorithm

```
function Simulated-Annealing(problem, schedule)
return state
node: current, neighbor; integer: t, T;
current := Make-Node(Initial-State(problem));
for t := 1 to MAX-ITERATION do
    T := schedule[t];
    if (T == 0) return State(current);
    neighbor := random-successor(current);
    if (Value(neighbor) < Value(current))
        then v := Value(neighbor) - Value(current);
            current := neighbor with Prob(exp(v/T))
        else current := neighbor;
    end if
end for
end function
```

Simulated Annealing

- Idea: escape local maxima by allowing some “bad” moves *but gradually decrease their size and frequency*
- Realize it by a random assignment like
 $current := neighbor \text{ with Prob}(\exp(v/T))$
where $v < 0$ and $|v|$ is the “size”; $T > 0$ is the “frequency”.
- Because $v < 0$ and $T > 0$, $0 < e^{\frac{v}{T}} < 1$.
- How to implement such a random assignment:
 - Generate a random real number x in $[0..1]$.
 - If $x \geq e^{\frac{v}{T}}$, then do the assignment; otherwise, do nothing.

Properties of Simulated Annealing

- The smaller $|v|$, the greater $e^{\frac{v}{T}}$; the greater T , the greater $e^{\frac{v}{T}}$.
- At fixed “temperature” T , state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough \Rightarrow always reach best state

- Is this necessarily an interesting guarantee??
- Devised by Metropolis et al., 1953, for physical process modelling
- Widely used in VLSI layout, airline scheduling, etc.