Informed Search and Exploration

Subject: Artificial Intelligence

Faculty:Mrs. Sonika Shrivastava

Informed (Heuristic) Search Strategies

- ***Informed Search*** – a strategy that uses problem-specific knowledge beyond the definition of the problem itself

- Use ***heuristics*** to guide the search
  - **Heuristic**: estimation or "hunch" of how to search for a solution
- We define a heuristic function:

  h(n) = "estimate of the cost of the cheapest path from the starting node to the goal node"

Best-First Search

- – an algorithm in which a node is selected for expansion based on an evaluation function f(n)
  - Traditionally the node with the lowest evaluation function is selected
  - Not an accurate name…expanding the best node first would be a straight march to the goal.
  - Choose the node that *appears* to be the best

Informed (Heuristic) Search Strategies

- There is a whole family of Best-First Search algorithms with different evaluation functions
  - Each has a heuristic function h(n)

- h(n) = estimated cost of the cheapest path from node n to a goal node

- Example: in route planning the estimate of the cost of the cheapest path might be the straight line distance between two cities

*A Star algorithm: A**

- A Star algorithm is a best first graph search algorithm that finds a least cost path from a given initial node to one goal node.

A * Algorithm

- g(n) = cost from the initial state to the current state n
- h(n) = estimated cost of the cheapest path from node n to a goal node
- f(n) = evaluation function to select a node for expansion (usually the lowest cost node)
- f(n)=g(n)+h(n)

A * Algorithm

- *Given:* A graph of nodes, S is start, G is goal.
- *Aim of the Experiment:* To find out the path from S to G
- with the minimum cost.
- *Procedure:*
- 1.Create a *search graph* G,consisting solely of the start
- node S . Put S on a list called OPEN.
- 2 .Create a list called CLOSED that is initially empty.
-
- 3.LOOP:if OPEN is empty,exit with failure.

A * Algorithm

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 s in G.

6.Expand node n,generating the set,M,of its successors and install them as successors of n in G.

A * Algorithm

7 Establish a pointer to n from 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 of M that was already on OPEN or CLOSED,decide whether or not to redirect its pointer to n.For each member of M already on CLOSED,decide for each of its descendents in G whether or not to redirect its pointer.

8.Reorder the list OPEN,either according to some scheme or some heuristic merit.

9.Goto LOOP

Features of A*

Q1. What are the applications of A Star algorithm?

A . A Star algorithm is often used to search for the lowest cost path from the q_start to the q_goal location in a graph of cells/voronoi/visibility/quad tree.

Q2.Give some real life problems which uses the above algorithm.

A . The algorithm solves problems like *8-puzzle problem* and *missionaries & Cannibals problem.*

Q3. List some differences between the A Star algorithm and Dijkshtra's algorithm.

A . A Star is generally considered to be the best pathfinding algorithm.Also Djikshtra's algorithm is essentially the same as A *,except that there is no heuristic(H is always 0).Because it has no heuristic,it searches by expanding out equally in every direction.So it usually ends up exploring a much larger area before the target is found.This generally makes it slower than A *.

Properties of Heuristic Function

- Admissible heuristics
- Consistent heuristics
- Dominance of Heuristics
- Composite Heuristics

Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, 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: $h_{SLD}(n)$ (never overestimates the actual road distance)find optimal path

A* Search

- A* (A star) is the most widely known form of Best-First search
  - It evaluates nodes by combining g(n) and h(n)
  - f(n) = g(n) + h(n)
  - Where
    - 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

A* Search

- When h(n) = actual cost to goal
  - Only nodes in the correct path are expanded
  - Optimal solution is found
- When h(n) < actual cost to goal
  - Additional nodes are expanded
  - Optimal solution is found
- When h(n) > actual cost to goal
  - Optimal solution can be overlooked

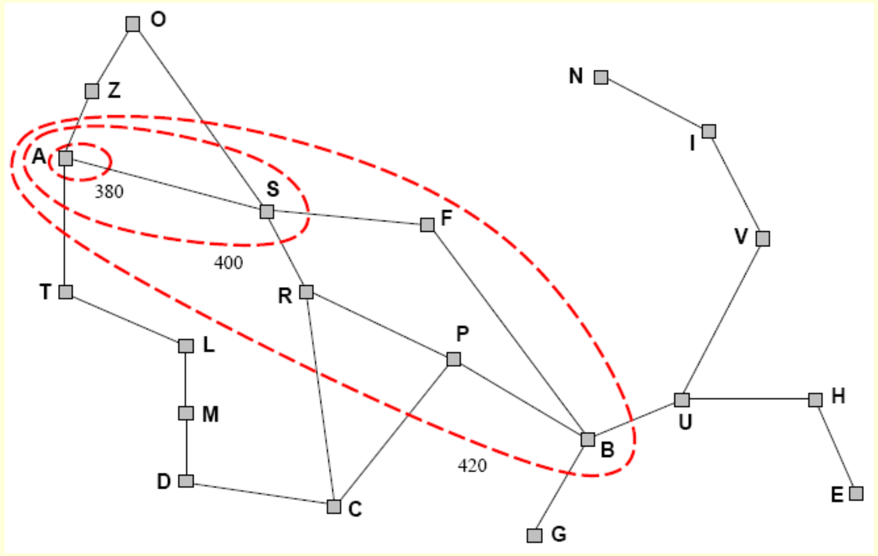
A* Search

- A* is optimal if it uses an ***admissible heuristic***
  - h(n) <= h*(n) the true cost from node n
  - if h(n) *never overestimates* the cost to reach the goal

- Example
  - h_{SLD} never overestimates the actual road distance

A* Search

- A* expands nodes in increasing f value
  - Gradually adds f-contours of nodes (like breadth-first search adding layers)
  - Contour i has all nodes $f=f_i$ where $f_i < f_{i+1}$



Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f $\leq f(G)$, i.e. step-cost > ε)
- Time/Space? Exponential: b^d

  except if: $|h(n) - h^*(n)| \leq O(\log h^*(n))$
- Optimal? Yes
- *Optimally Efficient*: Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)

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
- Best First search has $f(n)=h(n)$
- Uniform Cost search has $f(n)=g(n)$

Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, 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: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

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: it is guaranteed to expand less or equal nr of nodes.

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Relaxed problems

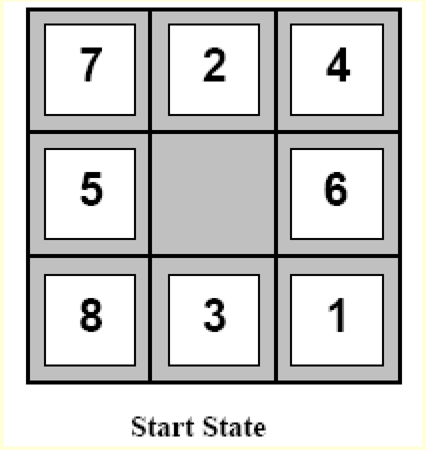
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution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

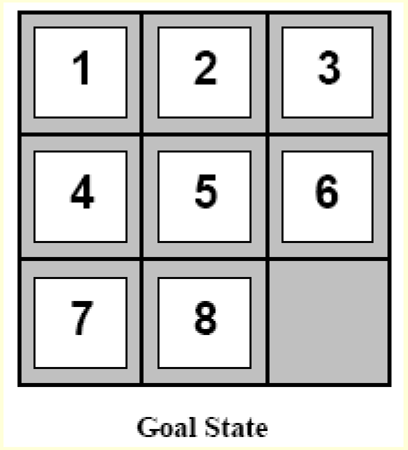
A* Search

- Complete
  - Yes, unless there are infinitely many nodes with f <= f(G)
- Time
  - Exponential in [relative error of h x length of soln]
  - The better the heuristic, the better the time
    - Best case h is perfect, O(d)
    - Worst case h = 0, $O(b^d)$ same as BFS
- Space
  - Keeps all nodes in memory and save in case of repetition
  - This is $O(b^d)$ or worse
  - A* usually runs out of space before it runs out of time
- Optimal
  - Yes, cannot expand f_{i+1} unless f_i is finished

Heuristic Functions

- Example: 8-Puzzle
  - Average solution cost for a random puzzle is 22 moves
  - Branching factor is about 3
    - Empty tile in the middle -> four moves
    - Empty tile on the edge -> three moves
    - Empty tile in corner -> two moves
  - 3^{22} is approx 3.1e10
    - Get rid of repeated states
    - 181440 distinct states



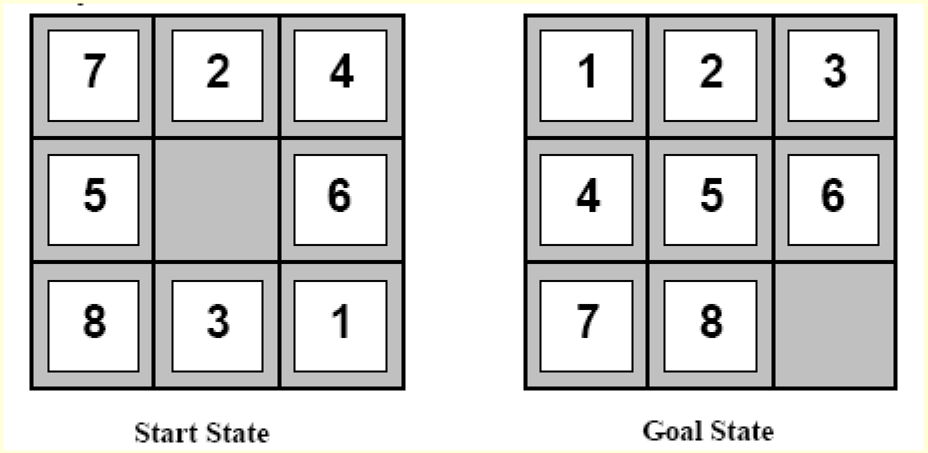


Heuristic Functions

- To use A* a heuristic function must be used that never overestimates the number of steps to the goal
- h1=the number of misplaced tiles
- h2=the sum of the Manhattan distances of the tiles from their goal positions

Heuristic Functions

- h1 = 7
- h2 = 4+0+3+3+1+0+2+1 = 14



Dominance

- If h2(n) > h1(n) for all n (both admissible) then h2(n) dominates h1(n) and is better for the search

- Take a look at figure 4.8!

Relaxed Problems

- A Relaxed Problem is a problem with fewer restrictions on the actions
  - The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- Key point: The optimal solution of a relaxed problem is no greater than the optimal solution of the real problem

Relaxed Problems

- Example: 8-puzzle
  - Consider only getting tiles 1, 2, 3, and 4 into place

  - If the rules are relaxed such that a tile can move anywhere then h1(n) gives the shortest solution
  - If the rules are relaxed such that a tile can move to any adjacent square then h2(n) gives the shortest solution

Relaxed Problems

- Store sub-problem solutions in a database
  - # patterns is much smaller than the search space
  - Generate database by working backwards from the solution
  - If multiple sub-problems apply, take the max
  - If multiple disjoint sub-problems apply, heuristics can be added

Learning Heuristics From Experience

- h(n) is an estimate cost of the solution beginning at state n
- How can an agent construct such a function?
- Experience!
  - Have the agent solve many instances of the problem and store the actual cost of h(n) at some state n
  - Learn from the features of a state that are relevant to the solution, rather than the state itself
    - Generate "many" states with a given feature and determine the average distance
    - Combine the information from multiple features
      - h(n) = c(1)*x1(n) + c(2)*x2(n) + … where x1, x2, … are features

Optimization Problems

- Instead of considering the whole state space, consider only the current state
- Limits necessary memory; paths not retained
- Amenable to large or continuous (infinite) state spaces where exhaustive search algorithms are not possible
- Local search algorithms can’t backtrack

Local Search Algorithms

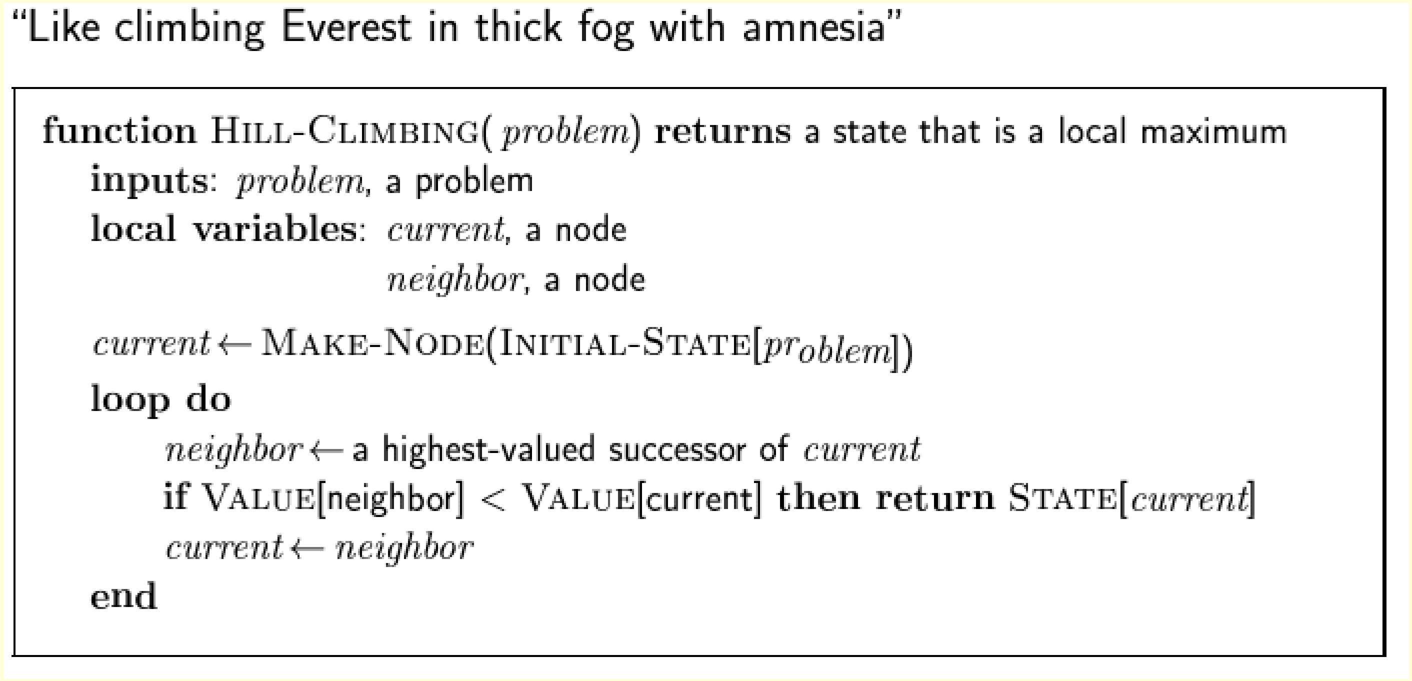
- They are useful for solving ***optimization problems***
  - Aim is to find a best state according to an ***objective function***

- Many optimization problems do not fit the standard search model outlined in chapter 3
  - E.g. There is no goal test or path cost in Darwinian evolution
- State space landscape

Hill Climbing

- Consider all possible successors as “one step” from the current state on the landscape.
- At each iteration, go to
  - The best successor (steepest ascent)
  - Any uphill move (first choice)
  - Any uphill move but steeper is more probable (stochastic)
- All variations get stuck at local maxima

Hill Climbing



"Like climbing Everest in thick fog with amnesia"

```
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
```

Hill Climbing

- Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence .
- Mathematical optimization problems implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-Travelling salesman problem where we need to minimize the distance traveled by salesman.
- ‘Heuristic search’ means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in reasonable time.
- A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

Features of Hill Climbing

1.Variant of generate and test algorithm : It is a variant of generate and test algorithm.
The generate and test algorithm is as follows :

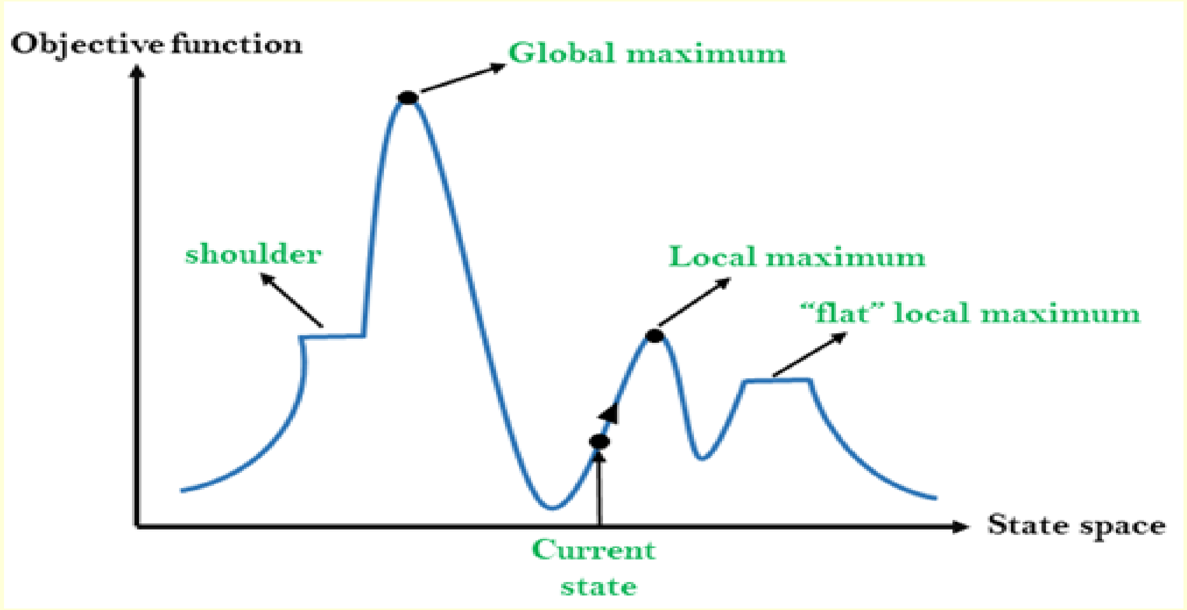
- ➢ Generate possible solutions.
- ➢ Test to see if this is the expected solution.
- ➢ If the solution has been found quit else go to step 1.

Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from the test procedur
Then this feedback is utilized by the generator in deciding the next move in search space.

2.Uses the Greedy approach : At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.

State Space diagram for Hill Climbing

- State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximize).
  X-axis : denotes the state space ie states or configuration our algorithm may reach.
  Y-axis : denotes the values of objective function corresponding to a particular state.
  The best solution will be that state space


Objective function
Global maximum
shoulder
Local maximum
"flat" local maximum
State space
Current state

Different regions in the State Space Diagram

- **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.
- **Global maximum :** It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
- **Plateua/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
- **Ridge :** It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
- **Current state :** The region of state space diagram where we are currently present during the search.
- **Shoulder :** It is a plateau that has an uphill edge.

Different regions in the State Space Diagram

- Local maxima = no uphill step
  - Algorithms on previous slide fail (not complete)
  - Allow "random restart" which is complete, but might take a very long time
- Plateau = all steps equal (flat or shoulder)
  - Must move to equal state to make progress, but no indication of the correct direction
- Ridge = narrow path of maxima, but might have to go down to go up (e.g. diagonal ridge in 4-direction space)

Types of Hill Climbing

- Simple Hill climbing
- Steepest-Ascent Hill climbing

Simple hill climbing

- Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:
- Less time consuming
- Less optimal solution and the solution is not guaranteed

Simple hill climbing

- Algorithm for Simple Hill Climbing:
- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
  - If it is goal state, then return success and quit.
  - Else if it is better than the current state then assign new state as a current state.
  - Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

steepest-Ascent

- The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

steepest-Ascent

- Algorithm for Steepest-Ascent hill climbing:
- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
  - Let SUCC be a state such that any successor of the current state will be better than it.
  - For each operator that applies to the current state:
    - Apply the new operator and generate a new state.
    - Evaluate the new state.
    - If it is goal state, then return it and quit, else compare it to the SUCC.
    - If it is better than SUCC, then set new state as SUCC.
    - If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

Problems and Solution in different regions in Hill climbing

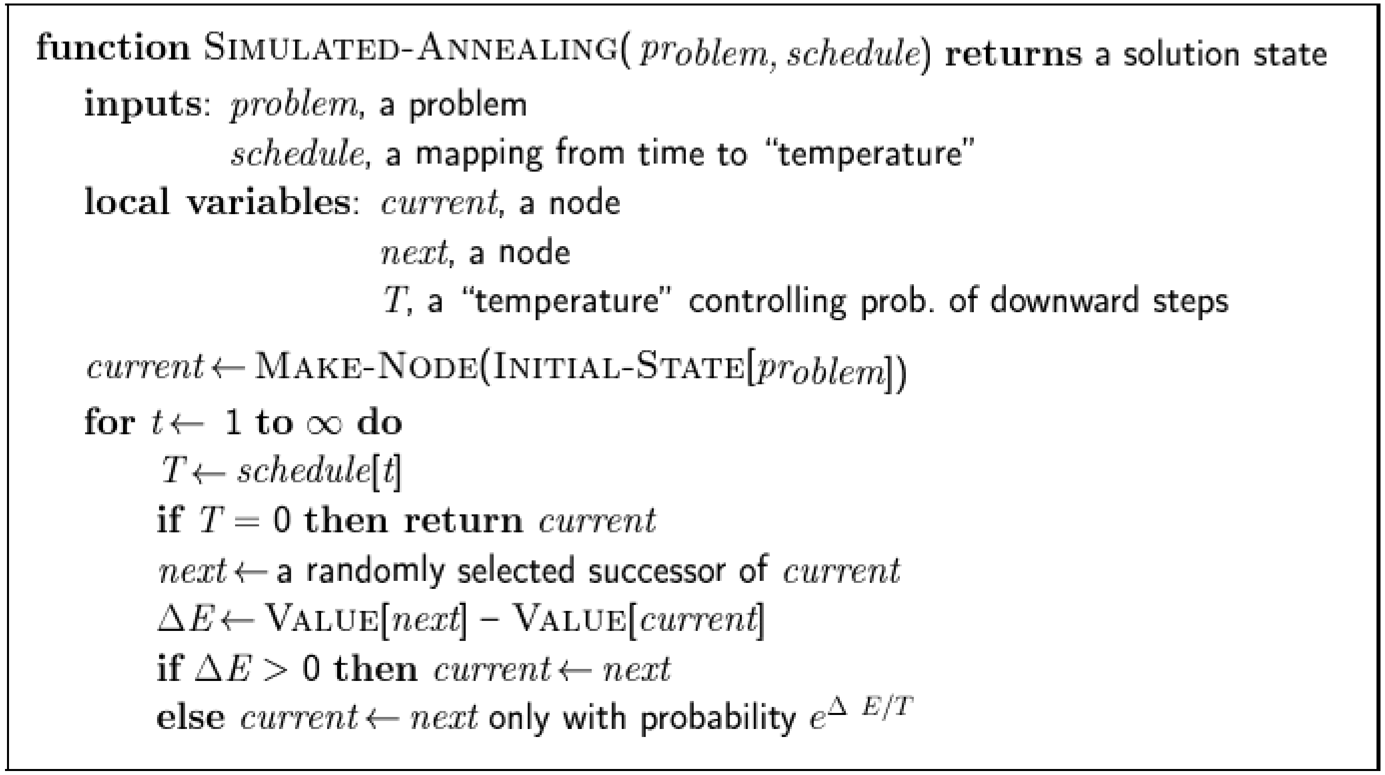
- Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :
- **Local maximum :** At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
  To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
- **Plateau :** On plateau all neighbors have same value . Hence, it is not possible to select the best direction.**To overcome plateaus :** Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region

- **Ridge :** Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state. **To overcome Ridge :** In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Simulated Annealing

- Idea: Escape local maxima by allowing some “bad” moves
  - But gradually decreasing their frequency
- Algorithm is randomized:
  - Take a step if random number is less than a value based on both the objective function and the Temperature
- When Temperature is high, chance of going toward a higher value of optimization function J(x) is greater
- Note higher dimension: “perturb parameter vector” vs. “look at next and previous value”

Simulated Annealing



```
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 $e^{\Delta E/T}$
```