AI 2002
Artificial Intelligence

Dr. Hashim Yasin

Hill-climbing Search

Hill-climbing Search

- It is simply a **loop** that continuously moves in the direction of increasing value—that is, uphill.
- It terminates when it reaches a "**peak**" where no neighbor has a higher value.
  - *does not maintain a **search tree***
  - need only
    - *record the state* and
    - the value of the *objective function*.
- Hill climbing only looks towards the immediate neighbors of the current state.

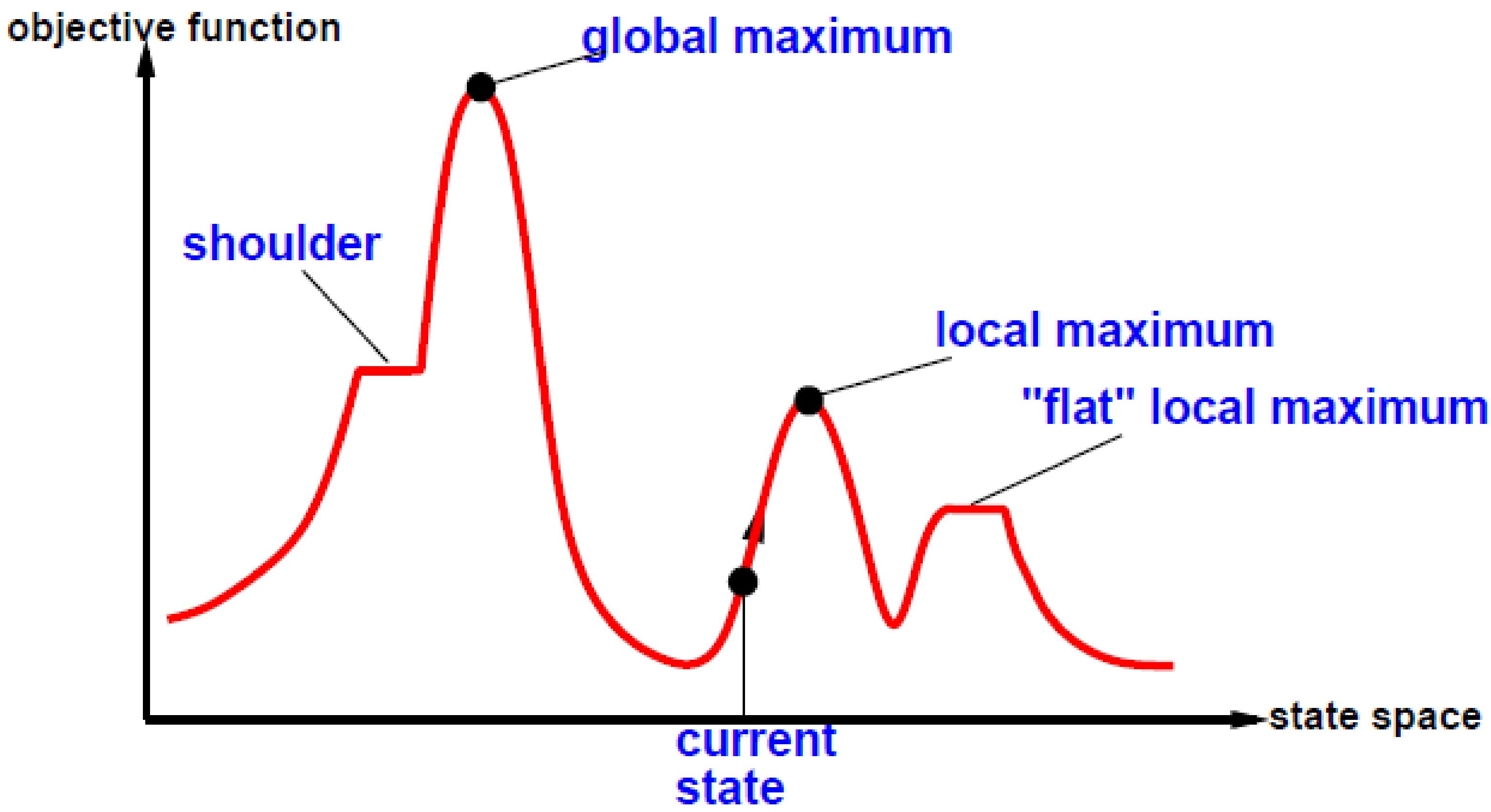
Hill-climbing Search

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of *current*
    if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
    current ← *neighbor*

- At *each step the current node is replaced by the best neighbor*; the neighbor with the highest VALUE,
- If a **heuristic cost estimate *h*** is used, we would find the neighbor with the ***lowest h***.

Hill-climbing Search



Hill-Climbing Search

With randomly generated 8-queens starting states, the steepest-ascent hill climbing:

- 14% of the time it solves the problem
- 86% of the time it get stuck at a local minimum
- However…
  - Takes only 4 steps on average when it succeeds
  - And 3 on average when it gets stuck
  - (for a state space with $\mathbf{8^8} \approx \mathbf{17}$ million states)

Hill-Climbing: Variants

Stochastic hill-climbing:

- Chooses *randomly* among *potential* successors
- Sometimes better than steepest ascent

First-choice hill-climbing:

- Generates successors randomly and picks *first*
- Good for many successors

Random restart hill-climbing:

- Restarts from randomly generated initial state when failed
- Roughly 7 iterations with 8-queens problem

Hill-Climbing Search

- **The success of hill-climbing search** depends very much
  - on the ***shape of the state-space landscape***

- If there are few local maxima and plateaux,
  - **random-restart hill climbing** will find a good solution very quickly.

Simulated Annealing

Simulated Annealing

Idea:

- escape local maxima by allowing some **"bad"** moves but gradually *decrease the size and frequency of the bad moves*,

- In thermodynamics, the probability to go from a state with energy E_1 to a state of energy E_2 is given by:

$$p = e^{\frac{(E_2 - E_1)}{kT}} = e^{\frac{-(E_1 - E_2)}{kT}}$$

Simulated Annealing

$$p = e^{\frac{-(E_1 - E_2)}{kT}}$$

Where,

- e is Euler’s number
- T is a “temperature" controlling the probability of downward steps
- k is Boltzmann’s constant
  - (relating energy and temperature; with appropriate choice of units, it will be equal to 1).

Simulated Annealing

$$p = e^{\frac{(E_2 - E_1)}{kT}} = e^{\frac{-(E_1 - E_2)}{kT}}$$

- The idea is that probability decreases exponentially with $E_2 - E_1$ increasing,
- The probability gets lower as temperature decreases
- If the ***schedule*** lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1.

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

  $current \leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

Similar to hill climbing, but a random move instead of best move

case of improvement, make the move

Otherwise, choose the move with probability that decreases exponentially with the "badness" of the move.

Simulated Annealing…Example

- Consider there are 3 moves available, with changes in the objective function of

$$\Delta E_1 = -0.1, \Delta E_2 = 0.5, \Delta E_3 = -3$$

- Suppose $T = 1$
- Pick a move randomly:
  - if ΔE_2 is picked, move there.
  - if ΔE_1 or ΔE_3 are picked, probability of move $= e^{\frac{\Delta E}{T}}$
    - move 1: prob1 = $e^{-0.1}$ = 0.9,
      - i.e., 90% of the time we will accept this move
    - move 3: prob3 = e^{-3}= 0.0497
      - i.e., 5% of the time we will accept this move

Simulated Annealing

T = “temperature” parameter

- **If T is high** => the probability of “locally bad” move is higher
- **If T is low** => the probability of “locally bad” move is lower
- typically, T is decreased as the algorithm runs longer
  - i.e., there is a “temperature schedule”

Simulated Annealing

<u>**Convergence:**</u>

- With <u>exponential schedule</u>, will provably converge to global optimum
  - If <u>*T* decreases slowly</u> enough, then simulated annealing search will find a global optimum with probability approaching 1.
- Few more precise convergence rate
  - Recent work on **rapidly mixing Markov chains**.
  - Surprisingly, deep foundations.

Simulated Annealing

- method proposed in 1983 by IBM researchers for solving VLSI layout problems.
  - theoretically will always **find the global optimum** (the best solution)
- Useful for some problems, but can be very **slow**
  - slowness comes about because T must be decreased very gradually to retain optimality
- In practice *how to decide the rate* at which to decrease T? (this is a practical problem with this method)

Local Beam Search

Local Beam Search

Idea:

- **Keep track of k states rather than just one**

- Start with k randomly generated states
- At each iteration, all the successors of all k states are generated
- If any one is a goal state, stop;
- else select the k **best successors** from the complete list and repeat.

Local Beam Search

```
function BEAM-SEARCH(problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

A local beam search with *k* states might seem to be nothing more than running ***k* random restarts in parallel** instead of in sequence.

Local Beam Search

Local beam search with $k = 1$

- We would randomly generate **1 start state**
- At each step we would generate all the successors, and retain the **1 best state**
- Equivalent to **HILL-CLIMBING**

Local Beam Search

Local beam search with $k = \infty$

- **1 initial state** and no limit of the number of states retained
- We start at initial state and **generate ALL successor** states (no limit how many)
- If one of those is a goal, we stop
- Otherwise, we generate all successors of those states (2 steps from the initial state), and continue

- Equivalent to **BREADTH-FIRST SEARCH** except that *each layer is generated all at once*.

Reading Material

- **Artificial Intelligence,** A Modern Approach

  Stuart J. Russell and Peter Norvig

  - **Chapter 4.**