Artificial Intelligence

Dr. Qaiser Abbas
Department of Computer Science & IT
University of Sargodha

Beyond Classical Search

- *Simplifying assumptions of the previous Study, and thereby getting closer to the real world.*
  - **Outline**:
    - Local search
    - Hill Climbing
    - Simulated annealing
    - Genetic algorithms (will continue in the next topic)

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

- Previously, state spaces were searched and recorded from start to end until a path to a solution is found, but here, the path to the goal is irrelevant.
  - For example, in 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

- If the path to the goal does not matter, then we are considering a different class of algorithms known as **Local search** algorithms.
  - It considers only current and its neighboring nodes.
  - Paths are not retained.
  - Two key advantages:
    (1) Use very little memory
    (2) Can find reasonable solutions in large or infinite state spaces.

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

- To understand local search, **state-space** landscape is given in figure.
  - If elevation corresponds to cost, then the aim is to find the lowest valley—a **global minimum**;
  - if elevation corresponds to an objective function, then the aim is to find the highest peak—a **global maximum**.
  - Local search algorithm are known to be **Complete** and **Optimal**.
  - A **plateau** is a flat area: can be a flat local maximum or a **shoulder** (from which progress is possible).

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

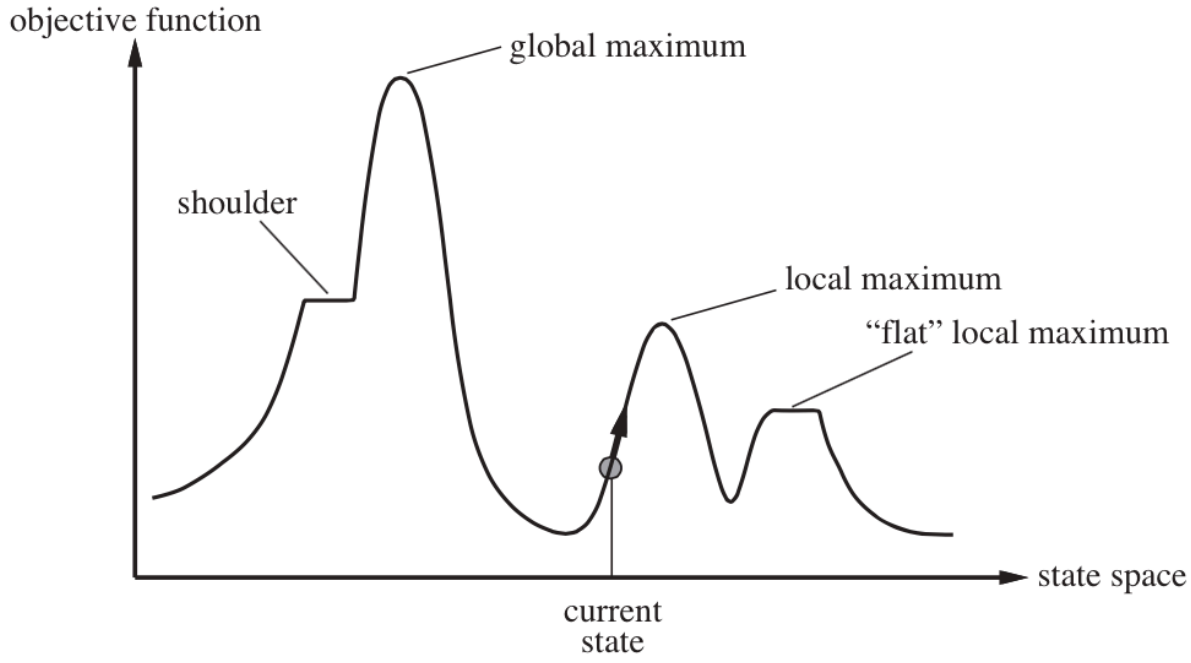


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Hill Climbing Algorithm

- Loop continues in the direction of increasing value—that is, uphill and terminates when it reaches a “peak” where no neighbor has a higher value.
- Record the current and its immediate neighboring node with the value of the objective function only.
- **Problems:**
  - Hill climbing is good for finding a **Local Maximum** but It does not guarantee the best possible solution (**Global Maximum**).
  - A hill-climbing search might get lost on the plateau.

Hill Climbing Algorithm

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
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

Variants of Hill Climbing Search

- **Stochastic hill climbing:**
  - Difference is only random selection of neighboring nodes w.r.t basic algorithm
- **Coordinate descent/ascent:**
  - Search in-turn along each coordinate, changing one variable at a time.
- **Random-restart hill climbing:**
  - It is also known as **Shotgun hill climbing**.
  - It iteratively does hill-climbing, each time with a random initial condition .
  - Best is kept if a new run of hill climbing produces a better than the stored state.
  - Get rid of local maxima

Simulated annealing

- A hill-climbing algorithm that *never* moves toward states with lower value is guaranteed to be incomplete, because it can get stuck on a local maximum.

- In contrast, a purely random walk—that is, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient.

- To try to combine hill climbing with a random walk in some way that yields both efficiency and completeness, **Simulated annealing** (heat and cool) is such an algorithm.

Simulated annealing

- Imagine the task of getting a ping-pong ball into the deepest crevice (narrow opening) in a bumpy surface. If we just let the ball roll, it will come to rest at a local minimum.

- If we shake the surface, we can bounce the ball out of the local minimum. The trick is to shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from the global minimum.

- Simulated-annealing solution is to start by shaking hard (i.e., at a high cost) and then gradually reduce the intensity of the shaking (i.e., lower the cost).

Simulated annealing

- **Simulated annealing (SA)** is a probabilistic technique for approximating the global optimum of a given function. It is often used when the search space is discrete.

- For problems where finding the precise global optimum is less important than finding an acceptable global optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as brute-force search or gradient descent.

Simulated annealing algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

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

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time.

Simulated annealing

- Instead of picking the *best* move, however, it picks a *random* move. If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1.

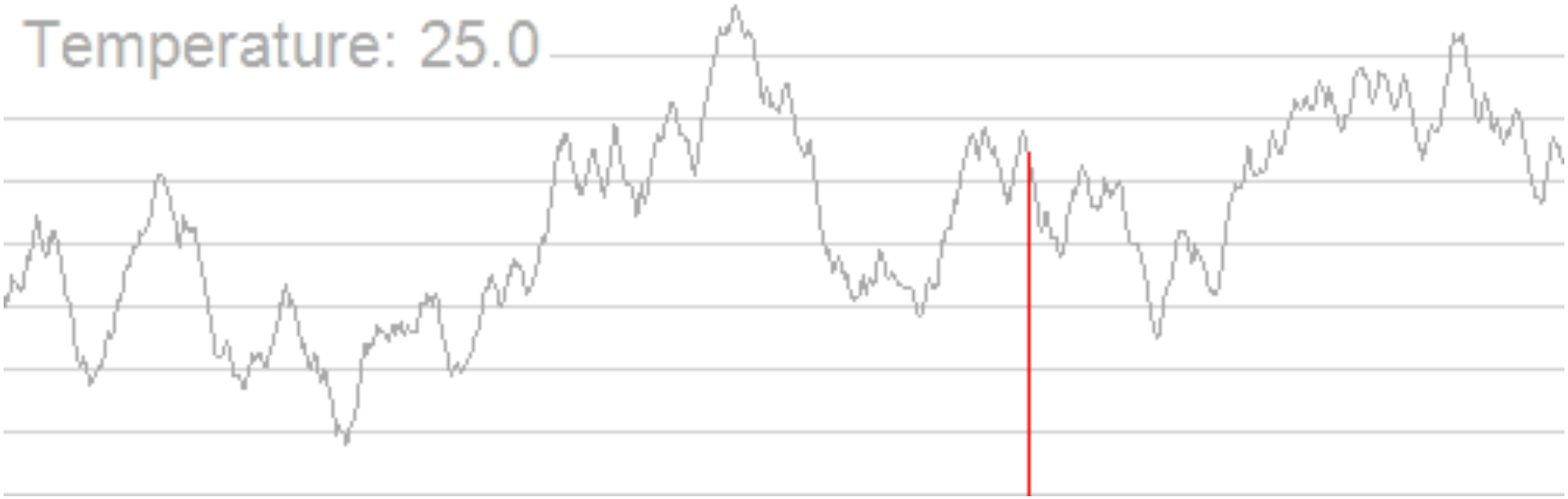
- The probability also decreases as the "temperature" T goes down: "bad" moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases.
  - $e^{-25/100} = 0.778$, $e^{-25/25} = 0.36$, $e^{-25/20} = 0.28$, $e^{-25/10} = 0.08$

- If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1.

Simulated annealing

- **The acceptance probability function**
  - The acceptance probability function takes in the old cost, new cost, and current temperature and spits out a number between 0 and 1, which is a sort of recommendation on whether or not to jump to the new solution.
  - For example:
    - 1.0: definitely switch (the new solution is better)
    - 0.0: definitely stay put (the new solution is infinitely worse)
    - 0.5: the odds are 50-50
  - Once the acceptance probability is calculated, it's compared to a randomly-generated number between 0 and 1. If the acceptance probability is larger than the random number, you're switching!

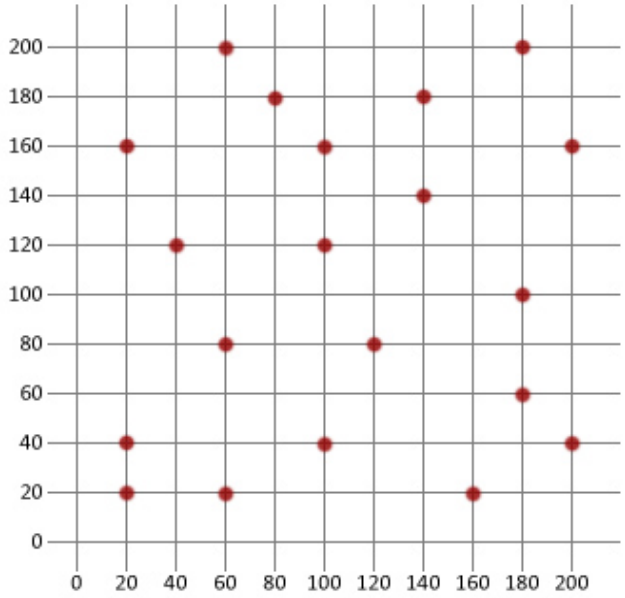
Simulated annealing

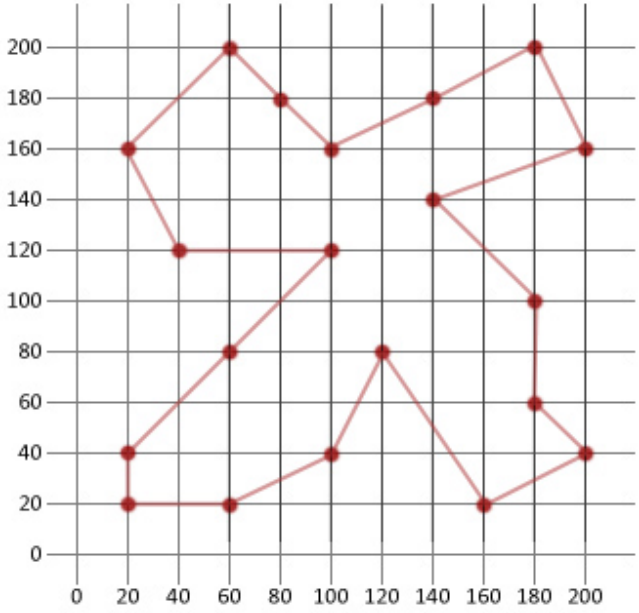


GIF Taken from Wikipedia

Lab Project 4

- Go to http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6
- Run and understand the Simulated Annealing in case of Travelling Salesman problem. Also augment the code with these maps as input and output.
- After hands on experience, submit the report along with the source code and screenshots.





Local beam search

- Read it yourself (an easy one)