Lecture 8-9
Artificial Intelligence

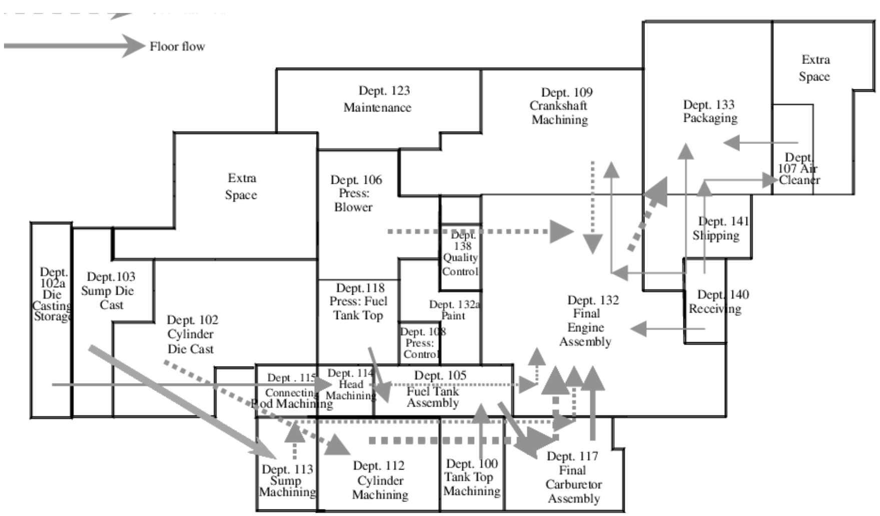
Khola Naseem
khola.naseem@uet.edu.pk

Local search

- Previous lecture: path to goal is solution to problem
  - systematic exploration of search space. O(b^d)
- The search algorithms that we have seen so far are designed to explore search spaces systematically.
- When a goal is found, the path to that goal also constitutes a solution to the problem
- path to goal is solution to problem
- E.g The solution to the traveling in Romania problem is a sequence of cities to get to Bucharest

Local search

- In many problems, however, the path to the goal is irrelevant. For example, in the 8-queens problem what matters is the final configuration of queens, not the order in which they are added.
- The goal itself is the solution.
- The same general property holds for many important applications such as integrated-circuit design, factory-floor layout, telecommunications network optimization etc.



Credit: Khola Naseem

Local search

- we need algorithms that are suitable for problems in which all that matters is the solution state, not the path cost to reach it.
- investigates online search, in which the agent is faced with a state space that is initially unknown and must be explored.
- The state space is set up as a set of “complete” configurations, the optimal configuration is one of them
- In such cases, we can use local search algorithms
  - Keeps a single "current" state, and then shift states, but don’t keep track of paths.
  - Use very limited memory
  - Find reasonable solutions in large state spaces.

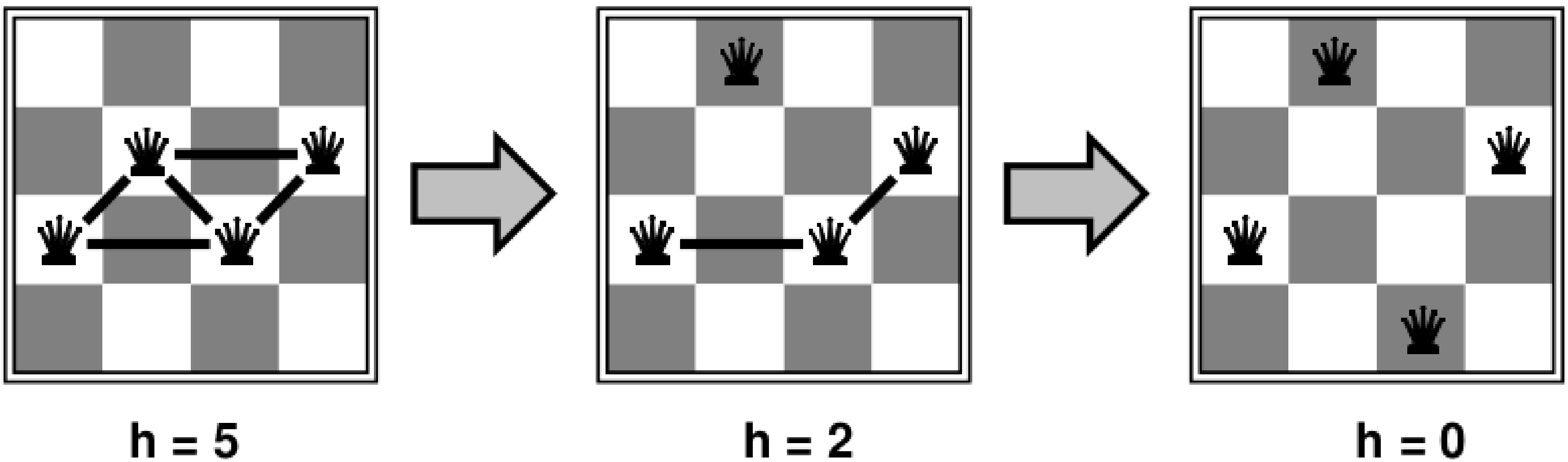
Local Search Methods

- Applicable when seeking Goal State & don't care how to get there. E.g.,
  - N-queens,
  - finding shortest/cheapest round trips

    (Travel Salesman Problem, Vehicle Routing Problem)
  - finding models of propositional formulae (SAT solvers)
  - VLSI layout, planning, scheduling, time-tabling, . . .
  - resource allocation
  - protein structure prediction
  - genome sequence assembly

Example:

- Goal: Put n queens on an n × n board with no two queens on the same row, column, or diagonal
- Neighbor: move one queen to another row
- Search: go from one neighbor to the next…



Initial state … Improve it … using local transformations

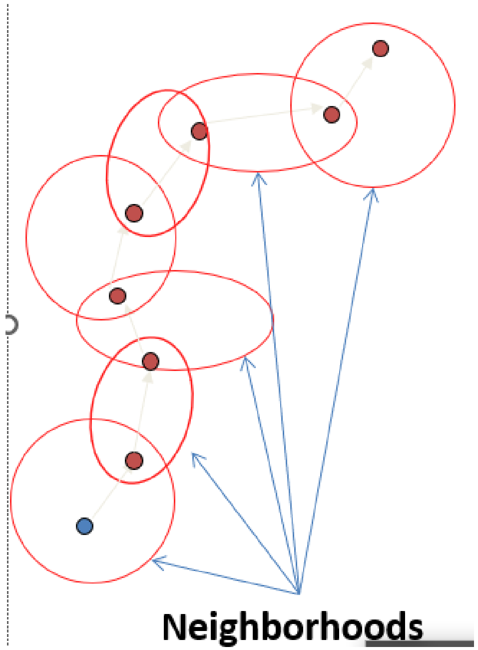
Almost always solves n-queens problems instantaneously for very large n, e.g., n = 1 million

Credit: Khola Naseem

Local Search Methods

- Key idea (surprisingly simple):

  1. Select (random) initial state (generate an initial guess)
  2. Make local modification to improve current state
     1. Evaluate current state and move to other states
  3. Repeat Step 2 until goal state found (or out of time)



Credit: Khola Naseem

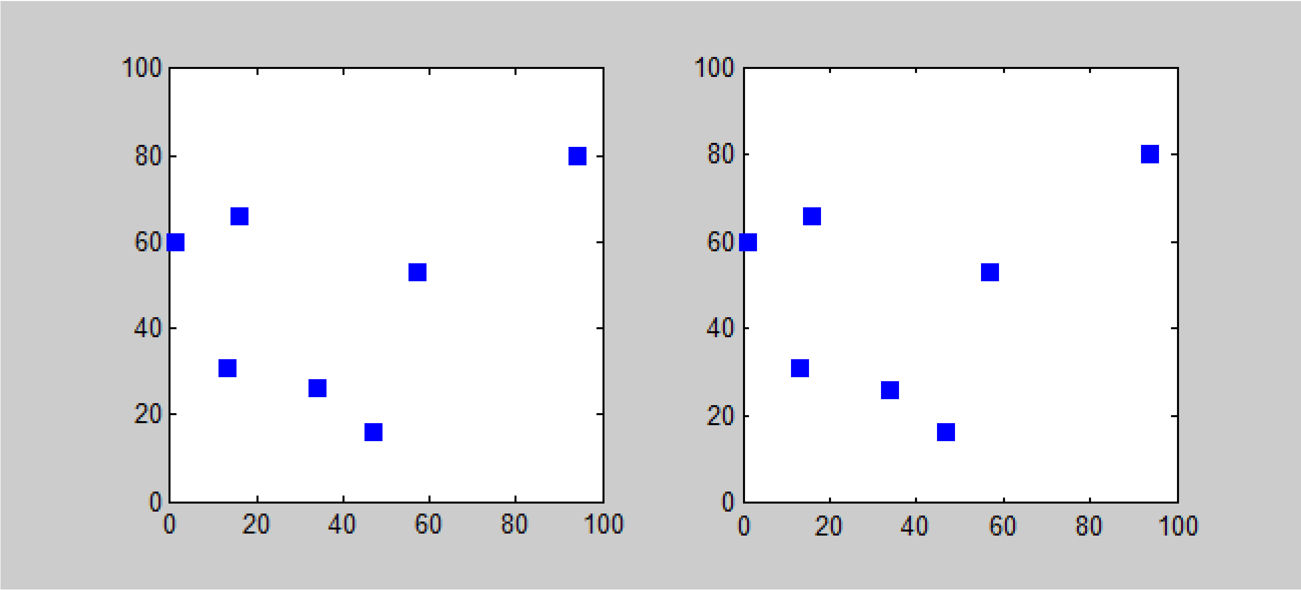
Example: Travelling Salesman Problem

- Find the shortest Tour traversing all cities once.

Credit: Khola Naseem

Example: Travelling Salesman Problem

- Find the shortest Tour traversing all cities once.



- 7 possible states

Example: Travelling Salesman Problem

- A Solution: Exhaustive Search
  - (Generate and Test) !!
- The number of all tours is about (n-1)!/2
- If n = 36 the number is about:

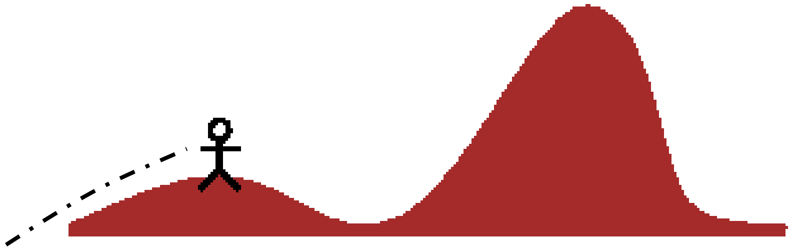
  566573983193072464833325668761600000000
- Not Viable Approach !!
- For 20 cities 77146 years

Credit: Khola Naseem

Local Search Algorithms

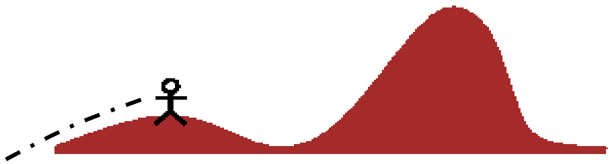
Hill Climbing (local search ,greedy, no backtracking)

- "Like climbing Everest in thick fog with amnesia“
- Hill climbing search algorithm (a.k.a greedy local search) uses a loop that continually moves in the direction of increasing values (that is uphill).
- It terminates when it reaches a peak where no neighbour has a higher value
- The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function



Local Search Algorithms

Hill Climbing algorithm



- "Like climbing Everest in thick fog with amnesia“

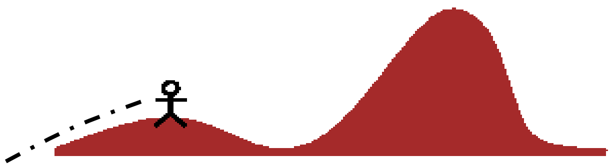
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

    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.

Local Search Algorithms

Hill Climbing algorithm

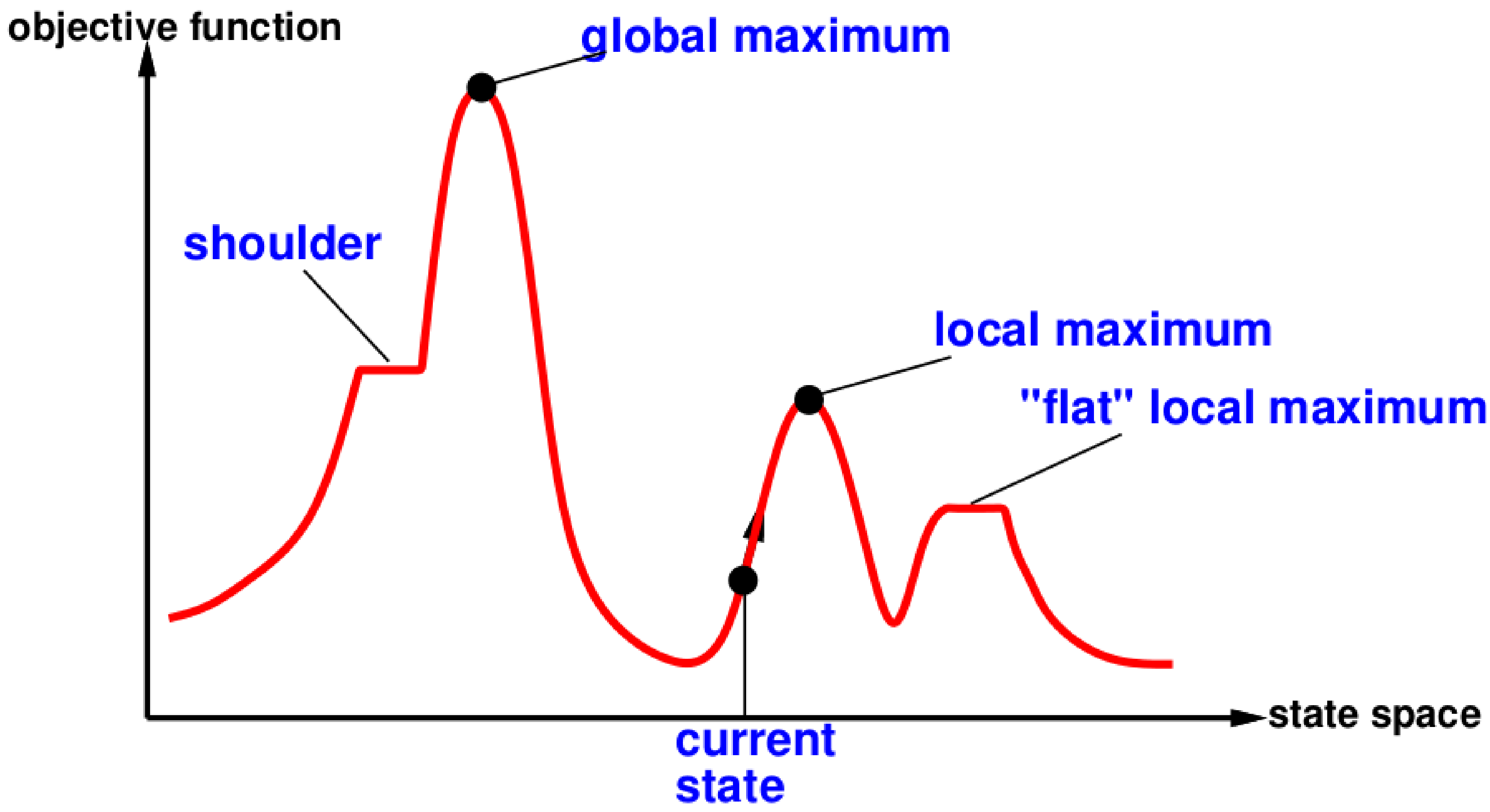


1. Pick a random point in the search space
2. Consider all the neighbours of the current state
3. Choose the neighbour with the best quality and move to that state
4. Repeat 2 to 4 until all the neighboring states are of lower quality
5. Return the current state as the solution state.

Local Search Algorithms

Hill Climbing algorithm

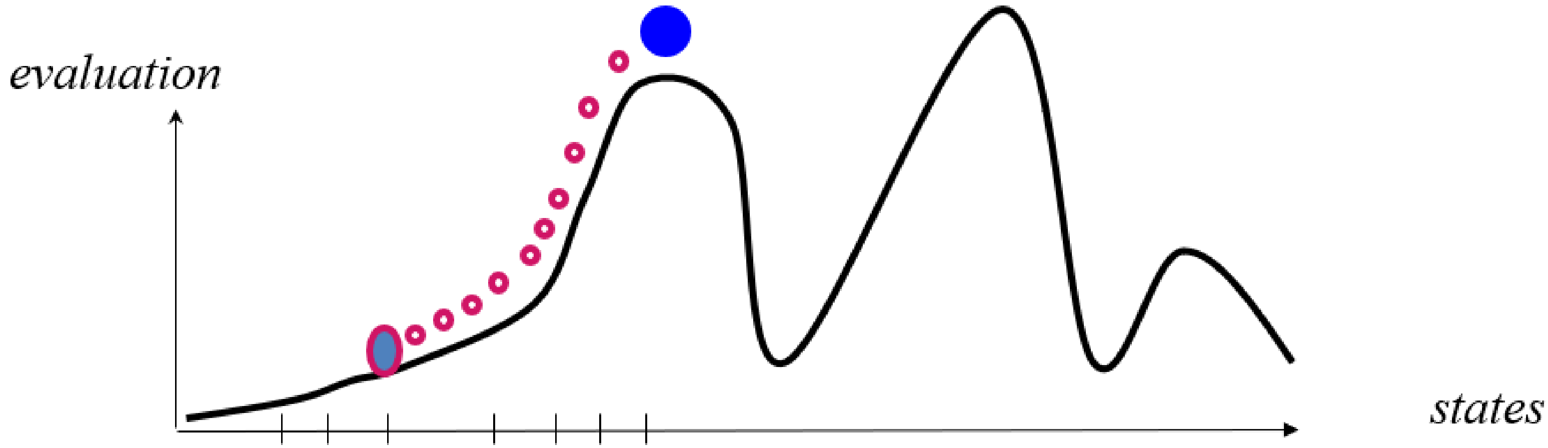
- A state space landscape is a graph of states associated with their costs



Local Search Algorithms

Hill Climbing algorithm Drawbacks

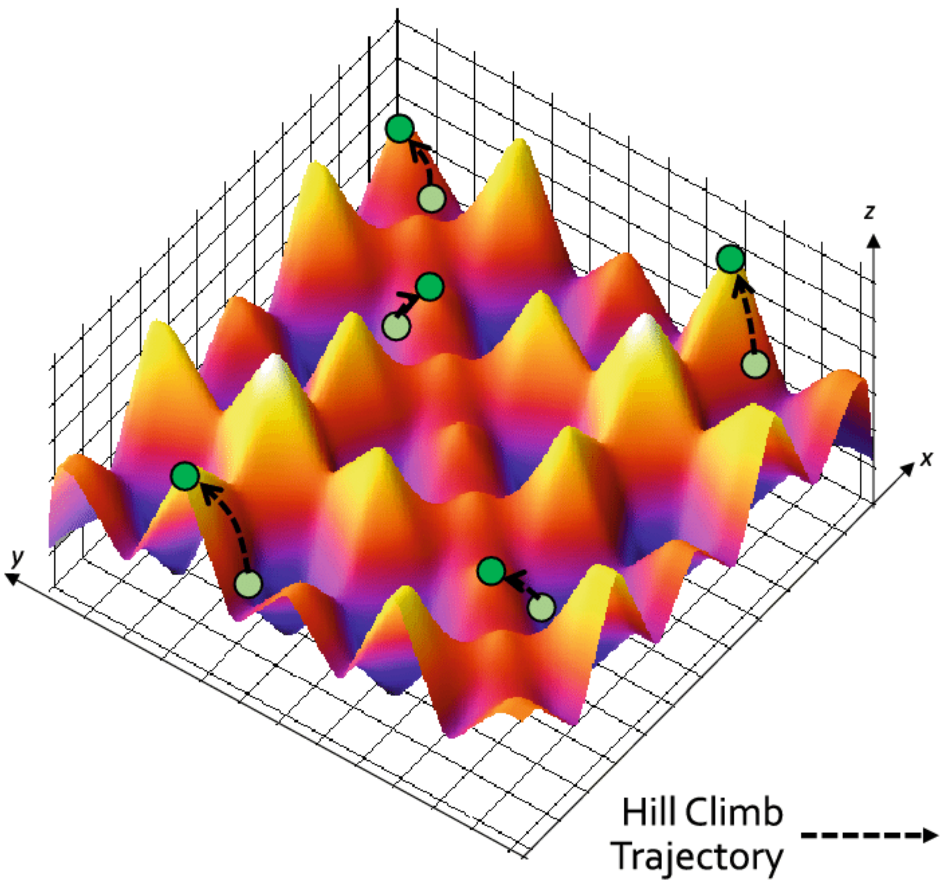
- **Local maxima:** a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.



Local Search Algorithms

Hill Climbing algorithm Drawbacks

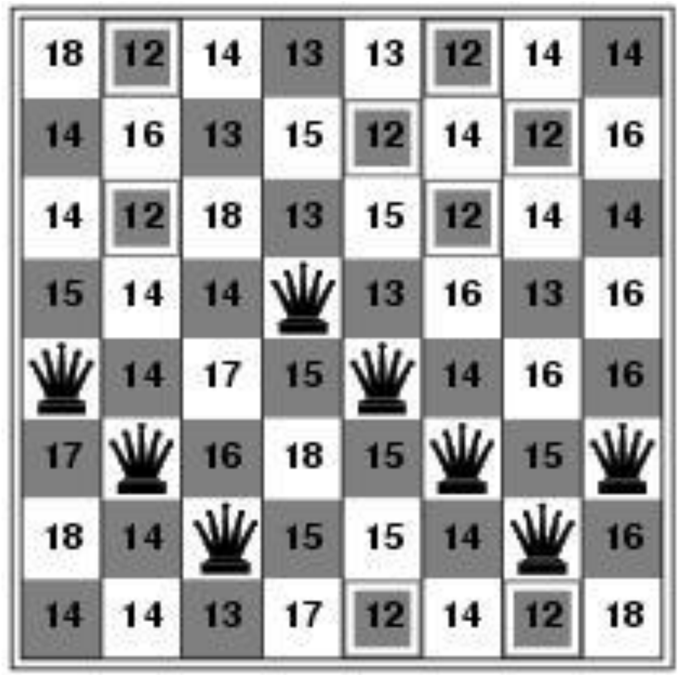
- Local maxima: gets worst in higher dimension



Local Search Algorithms

Hill Climbing algorithm Drawbacks

- **Local maxima:** Hill-climbing search: 8-queens problem



Need to convert to an optimization problem
h = number of pairs of queens that are attacking each other
h = *17* for the above state

Local Search Algorithms

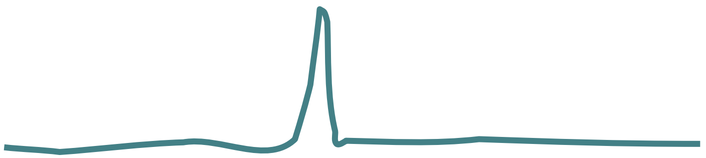
Hill Climbing algorithm Drawbacks

- **Local maxima:** Hill-climbing search: 8-queens problem
  - Randomly generated 8-queens starting states…
    - 14% the time it solves the problem
    - 86% of the time it get stuck at a local minimum
  - (for a state space with 8^8 =~17 million states)

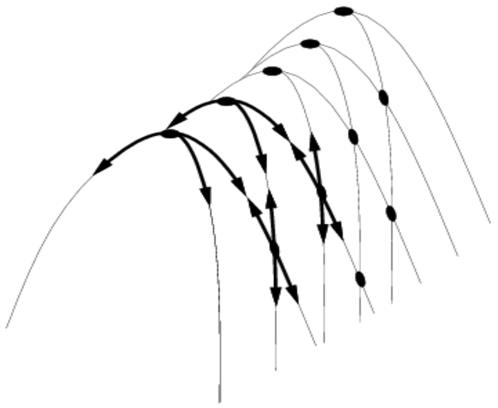
Local Search Algorithms

Hill Climbing algorithm Drawbacks

- Plateaus



- Diagonal ridges

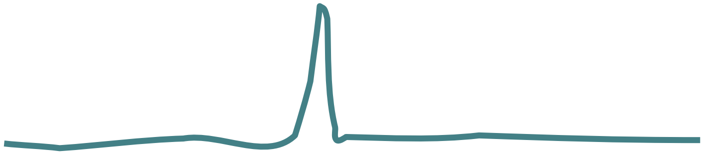


Credit: Khola Naseem

Local Search Algorithms

Hill Climbing algorithm Drawbacks

- Plateaus
  - Its not it's a good option to keep moving
  - must take care of infinite loop
    - One common solution is to put a limit on the number of consecutive sideways moves allowed. For example, we could allow up to, say, 100 consecutive sideways moves in the 8-queens problem. This raises the percentage of problem instances solved by hill climbing from 14% to 94%



Local Search Algorithms

Many variants of hill climbing have been invented.

- **FIRST-CHOICE HILL CLIMBING:**
  - hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g., thousands) of successors.
- **Random-restart hill climbing**
  - Random-restart hill climbing adopts the well-known adage, “If at first you don’t succeed, try, try again.” It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

Local Search Algorithms

Many variants of hill climbing have been invented.

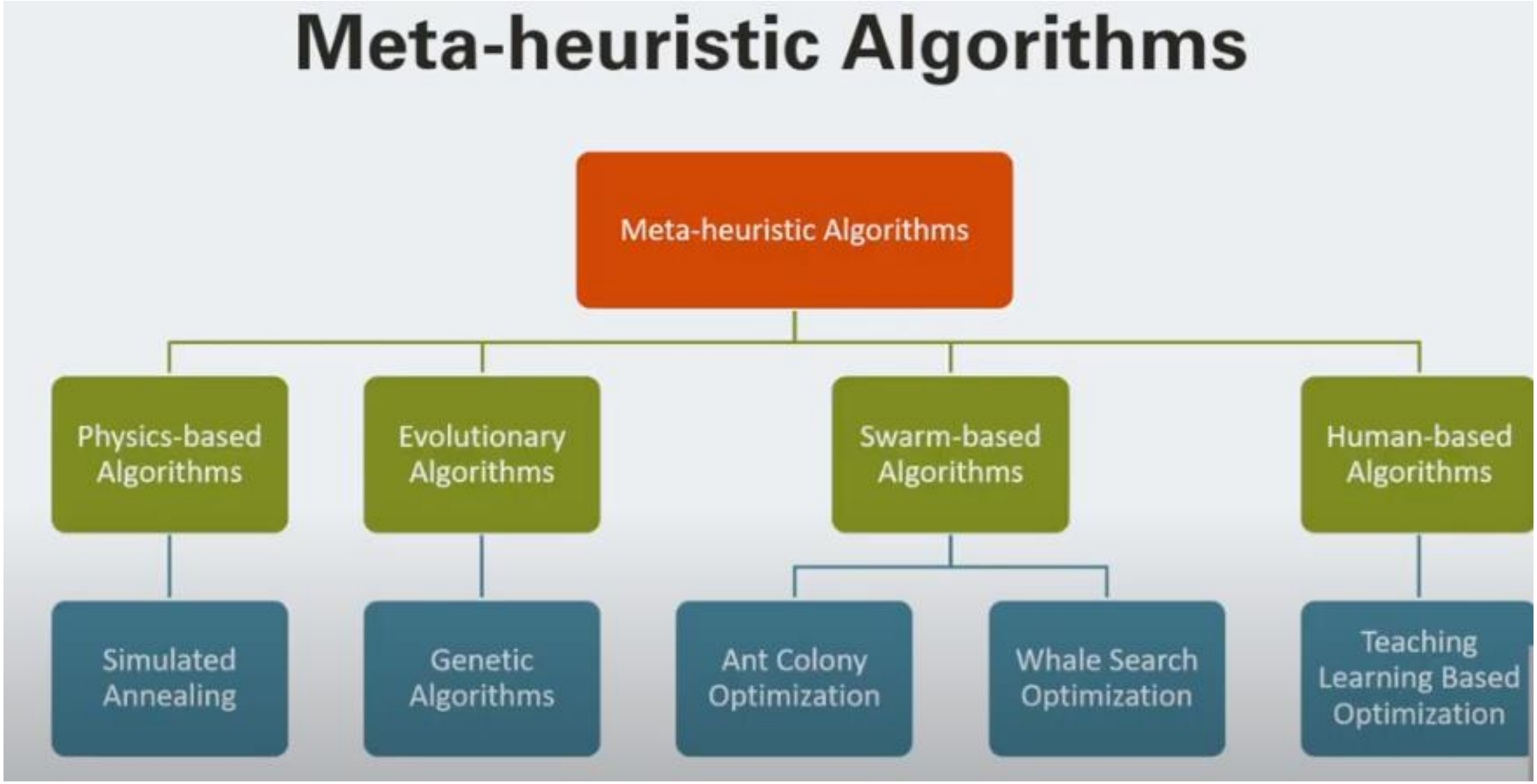
- **Random-restart hill climbing**
- The success of hill climbing depends very much on the shape of the state-space landscape: if there are few local maxima and plateaux, random-restart hill climbing will find a good solution very quickly. On the other hand, many real problems have a landscape that
- Are complex NP-hard problems typically have an exponential number of local maxima to get stuck on. Despite this, a reasonably good local maximum can often be found after a small number of restarts.

Meta-Heuristic

- Metaheuristics are a type of algorithm that are used to find approximate solutions to optimization problems. They are often used when the exact solution is too computationally expensive to find.
- Metaheuristics work by iteratively improving a solution until it is good enough to be considered the final answer.
- There are many different types of metaheuristics, each with their own strengths and weaknesses. Some of the more popular metaheuristics include simulated annealing, genetic algorithms

Meta-Heuristic

- Metaheuristics

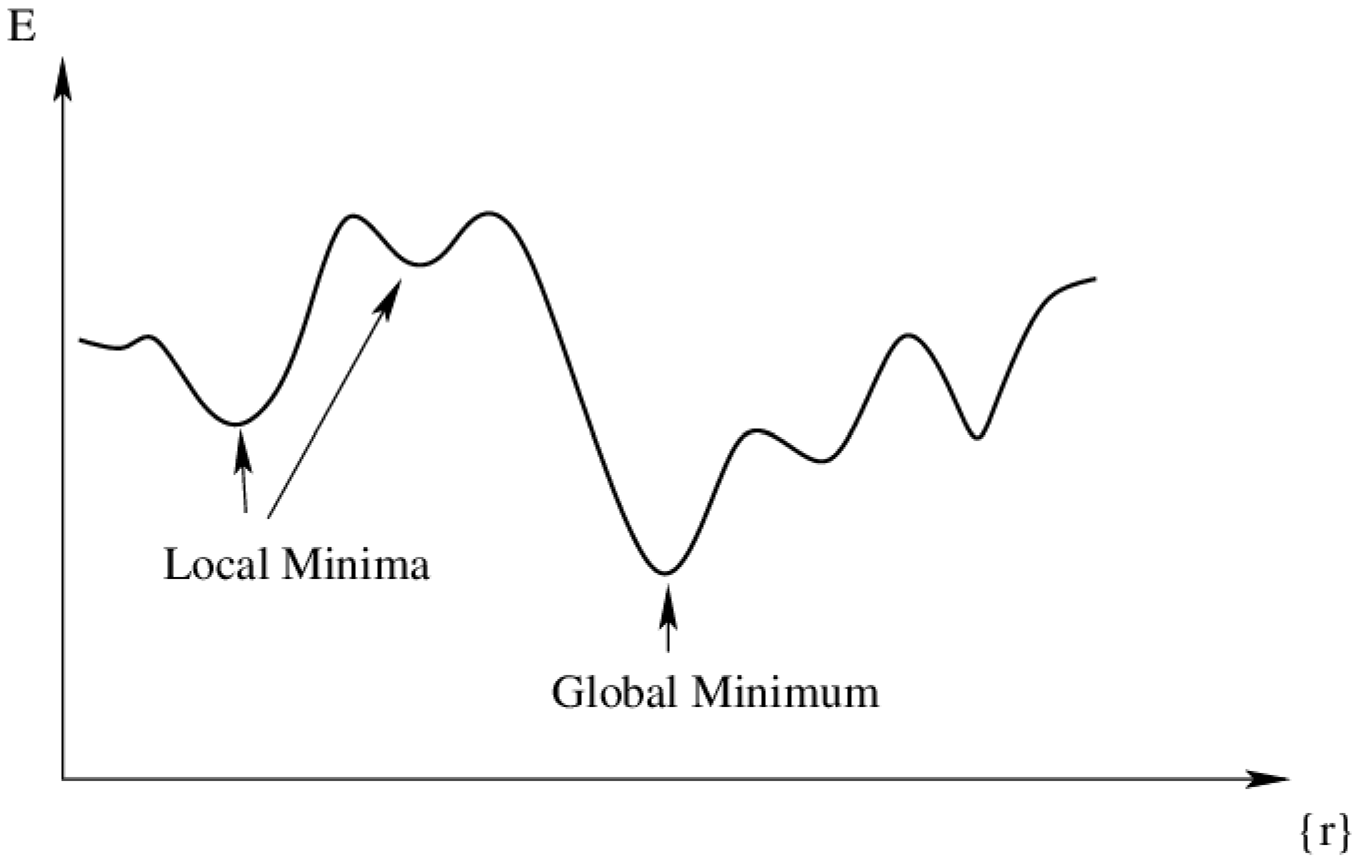


Credit: Khola Naseem

Meta-Heuristic

- Metaheuristics work by iteratively improving a solution to a problem. They start with an initial solution, then use a set of rules or heuristics to modify the solution.
- The goal is to find a solution that is better than the current one. The process is repeated until a satisfactory solution is found.
- Metaheuristics are often used for problems that are NP-hard, meaning that they are difficult to solve. However, metaheuristics can often find good solutions to these problems in a reasonable amount of time.

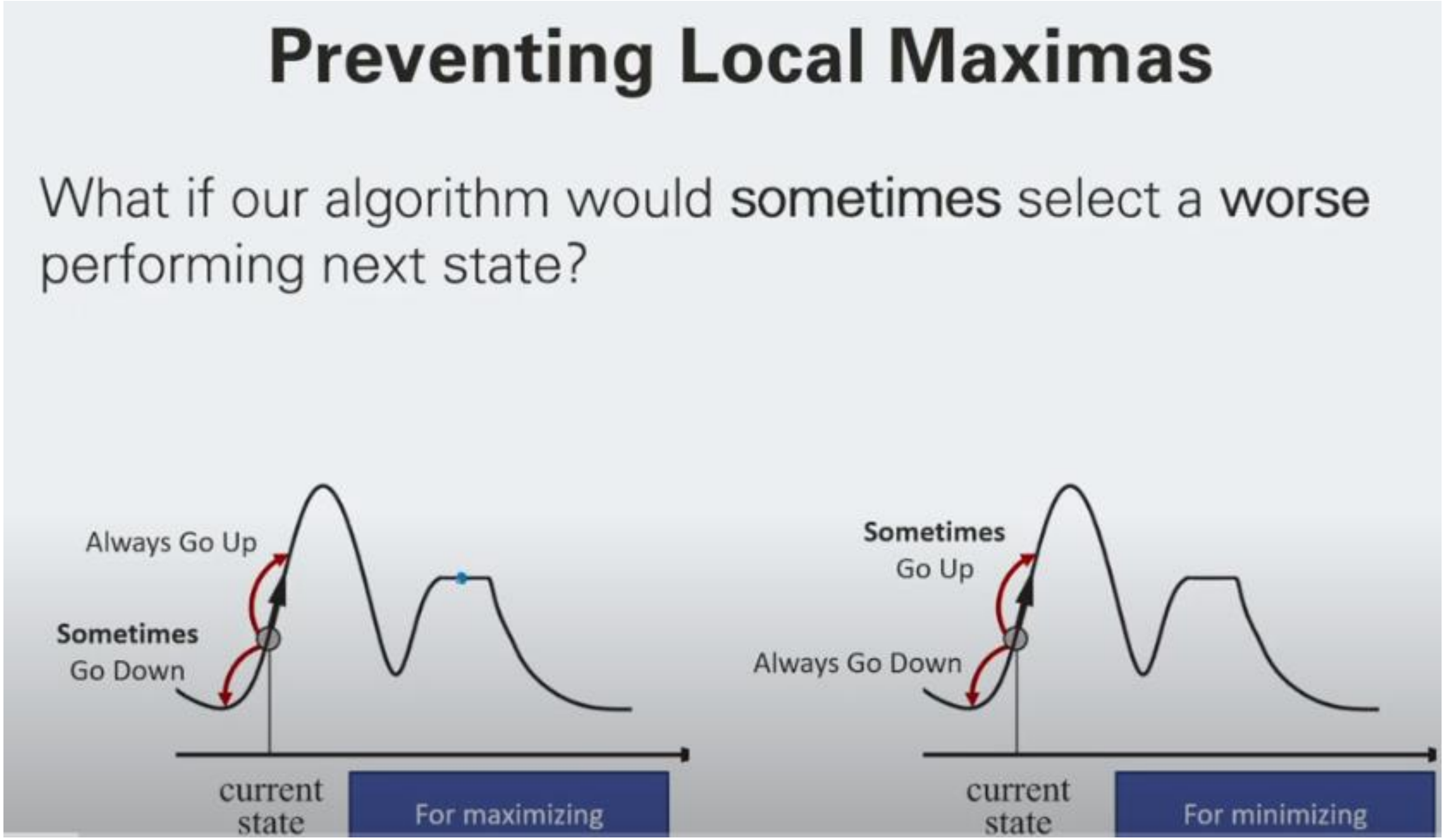
Optimization problem



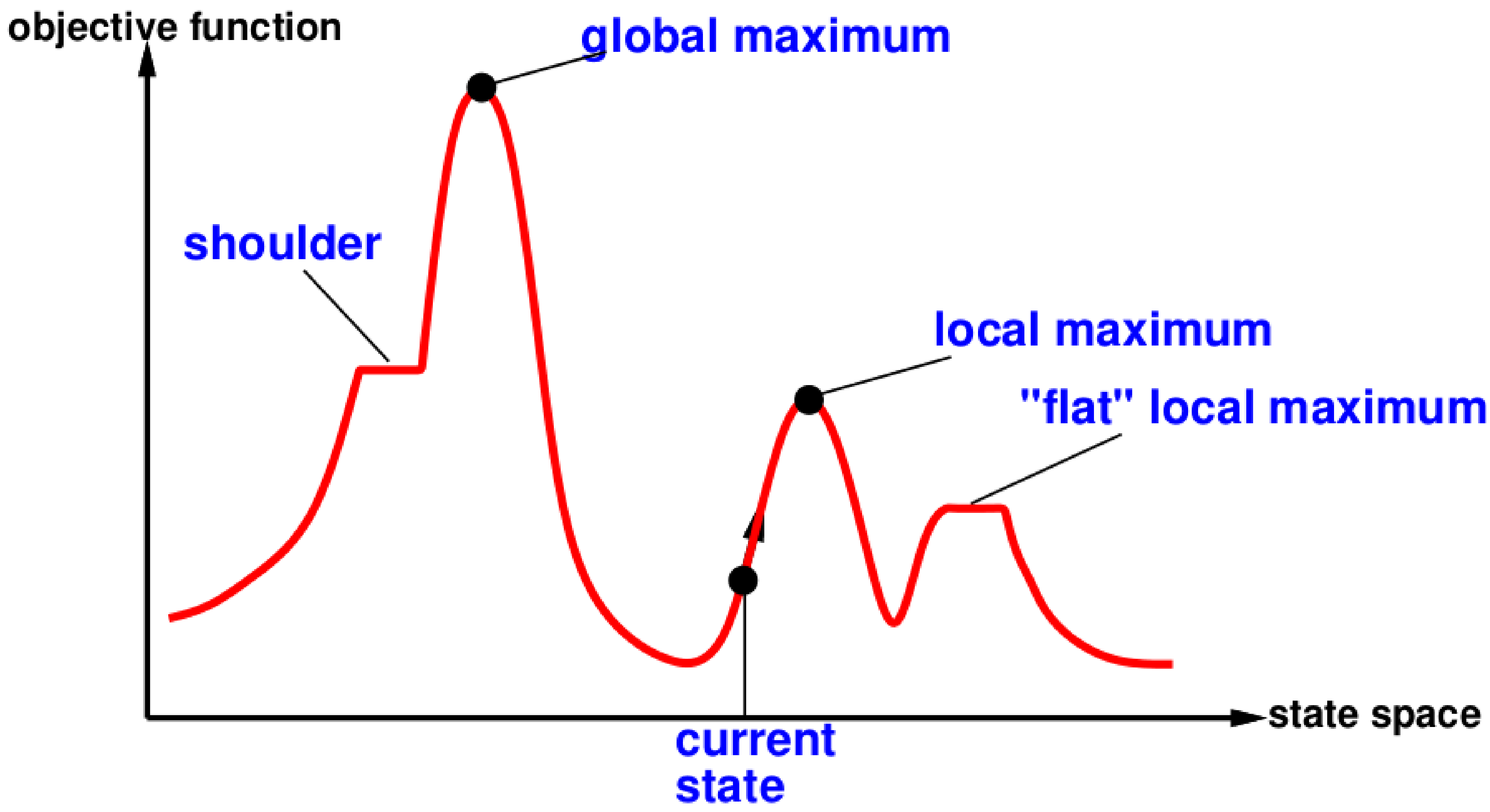
Simulated Annealing

- Issue in Hill climbing:

- A hill-climbing algorithm that never makes "downhill"



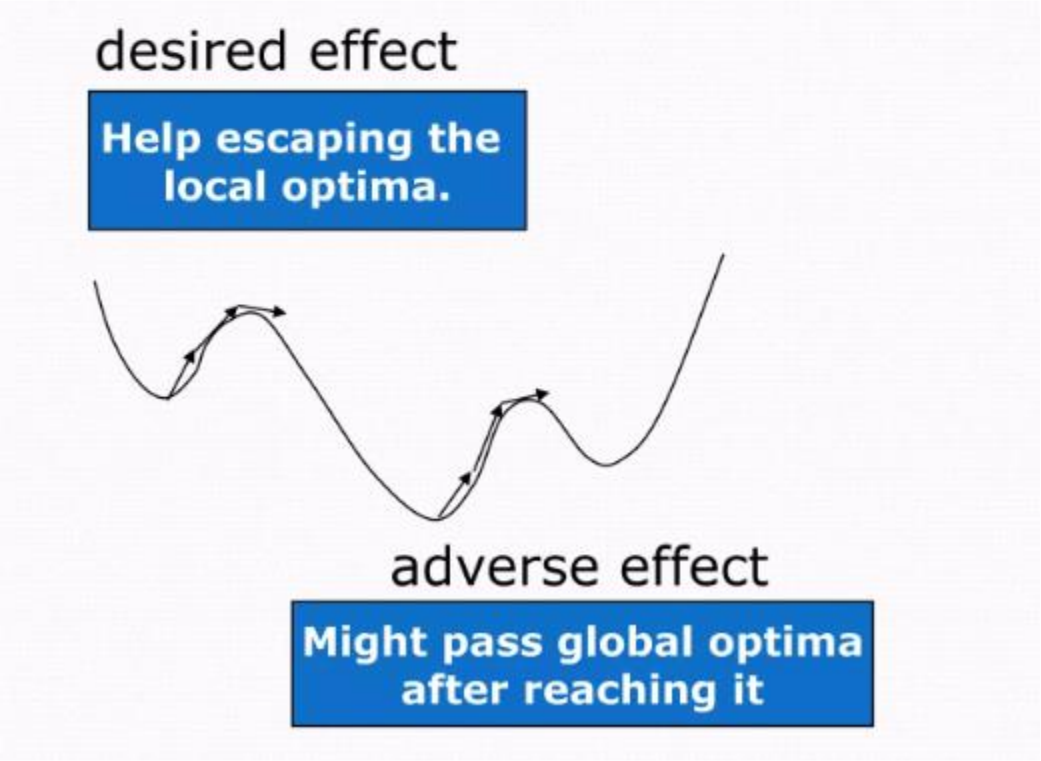
Simulated Annealing



Credit: Khola Naseem

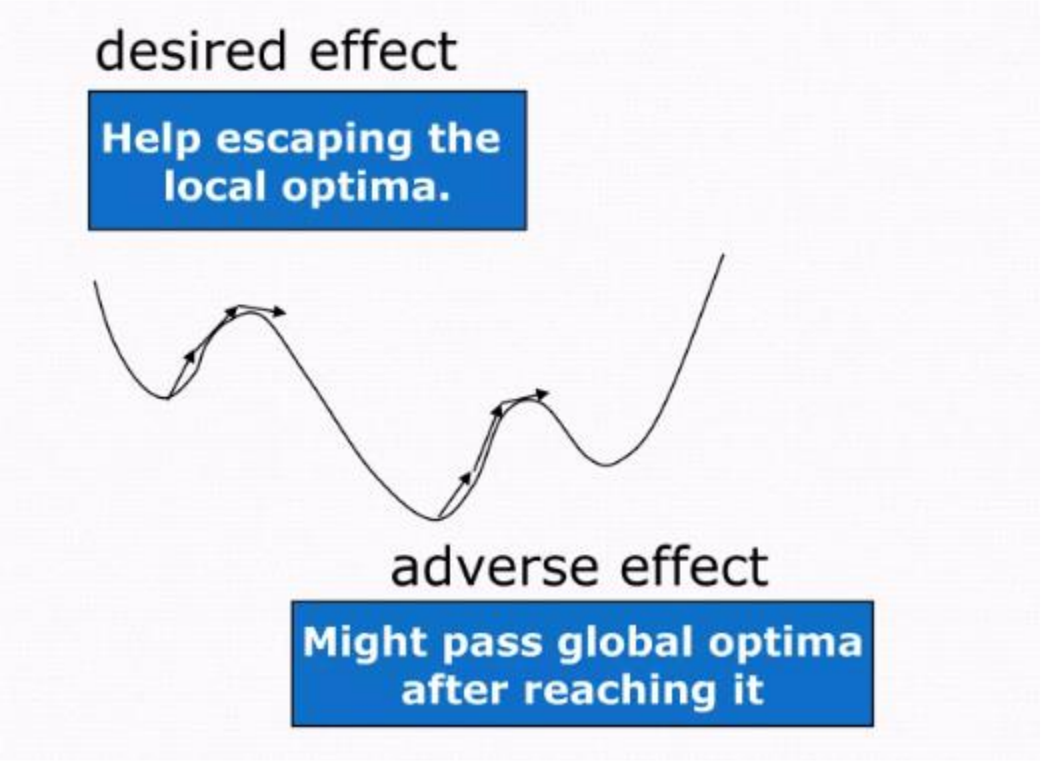
Simulated Annealing

- X1,x2 f(x1)>f(x2)
- f(x2)>f(x1) but problem is minimization



Simulated Annealing

- X1,x2 f(x1)>f(x2)
- f(x2)>f(x1)   but problem is minimization



Simulated Annealing

Physical Annealing	Simulated Annealing
Metal	Optimization Problem
Energy State	Cost Function
Temperature	Control Parameter
Crystalline Structure	The optimal Solution
Global optima solution can be achieved as long as the cooling process is slow enough	

Simulated Annealing

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* ← *next*
        else *current* ← *next* only with probability $e^{\Delta E/T}$

Simulated Annealing

- For maximization:

Simulated Annealing

Let **E** denotes the objective function value (also called **energy**)

If $\Delta E = E_{next} - E_{current} > 0$; probability of accepting a state with a **better object function** is always **1**

If $\Delta E = E_{next} - E_{current} < 0$; probability of accepting a state with a **worse object function** is

$$P(\text{Accept Next}) = e^{\Delta E/T}$$

T = temperature at time step

For **maximizing**; slight change for **minimizing**

Simulated Annealing

- :

```
func simulated_annealing(state):
    for t = 1 to ∞ do
        T ← schedule(t)
        if T = 0 then
              return state
        candidate ← random_neighbor(state)
        E = eval(candidate) - eval(state)
        if E > 0 then
            state ← candidate
        else
            prob ← probability(E, T)
            if random() < prob  then
               state ← candidate
```

Application example:

- :

In placement there are multiple complex objective.

The cost function is difficult to balance and requires testing.

An example of cost function that balances area efficiency vs. performance.

$$Cost=c_1 area+c_2 delay+c_3 power+c_4 crosstalk$$

Where the c_i weights heavily depend on the application and requirements of the project