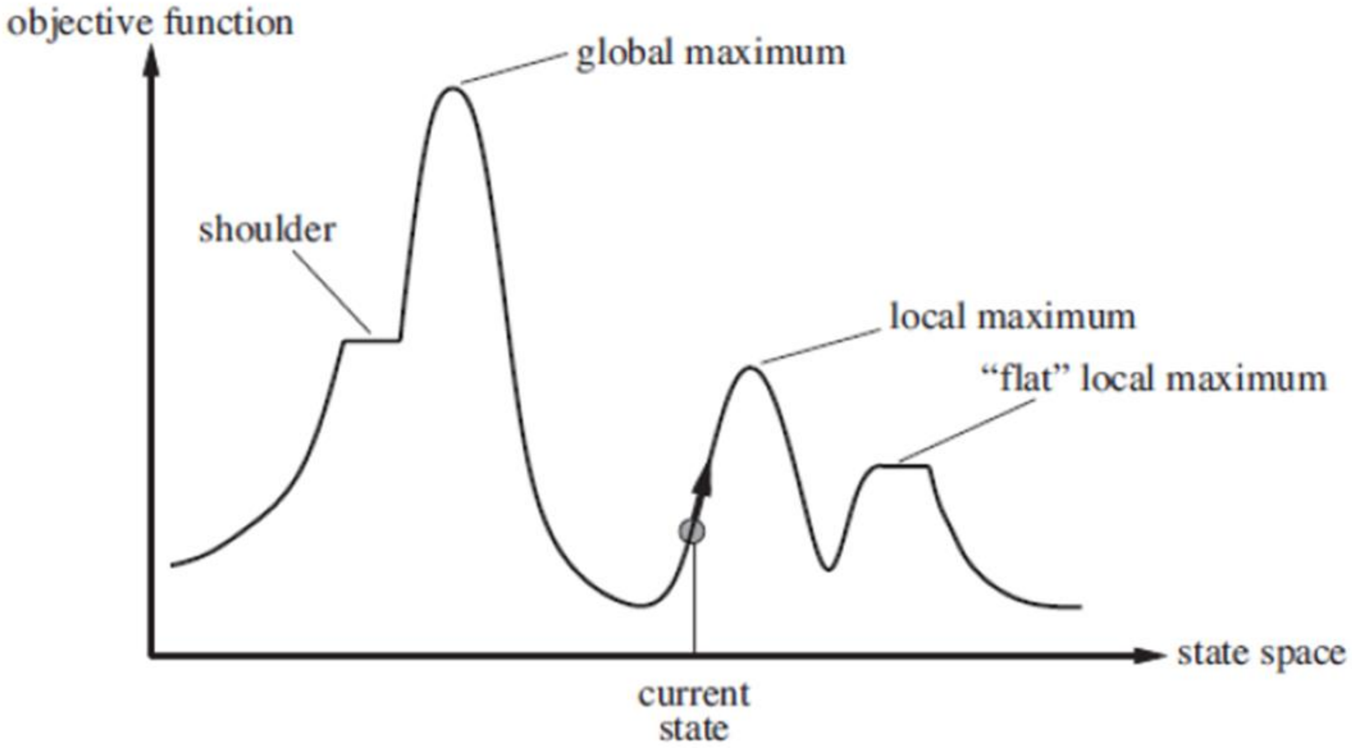
Module 6: Optimisation (Hill Climbing, Simulated Annealing and GA)

Source for the slides:

https://www.xpowerpoint.com/hill-climbing-search-main--PPT.html#

Local vs Global Search Algorithms

- Hill Climbing
- SA
- S.T.

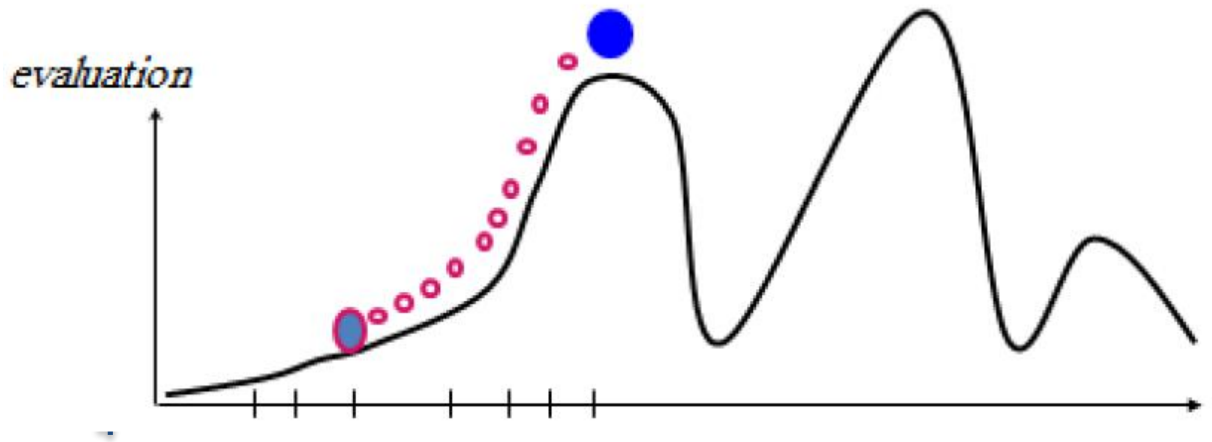


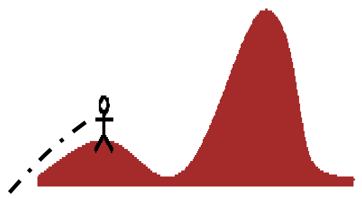
- GA

Hill-Climbing Search

Hill-Climbing Search

- **Main Idea**: Keep a single current node and move to a neighboring state to improve it.
- Uses a loop that continuously moves in the direction of increasing value (**uphill**):
- Choose the best successor, choose randomly if there is more than one.
- Terminate when a peak reached where no neighbor has a higher value.
- It also called **greedy local search**, steepest ascent/descent.



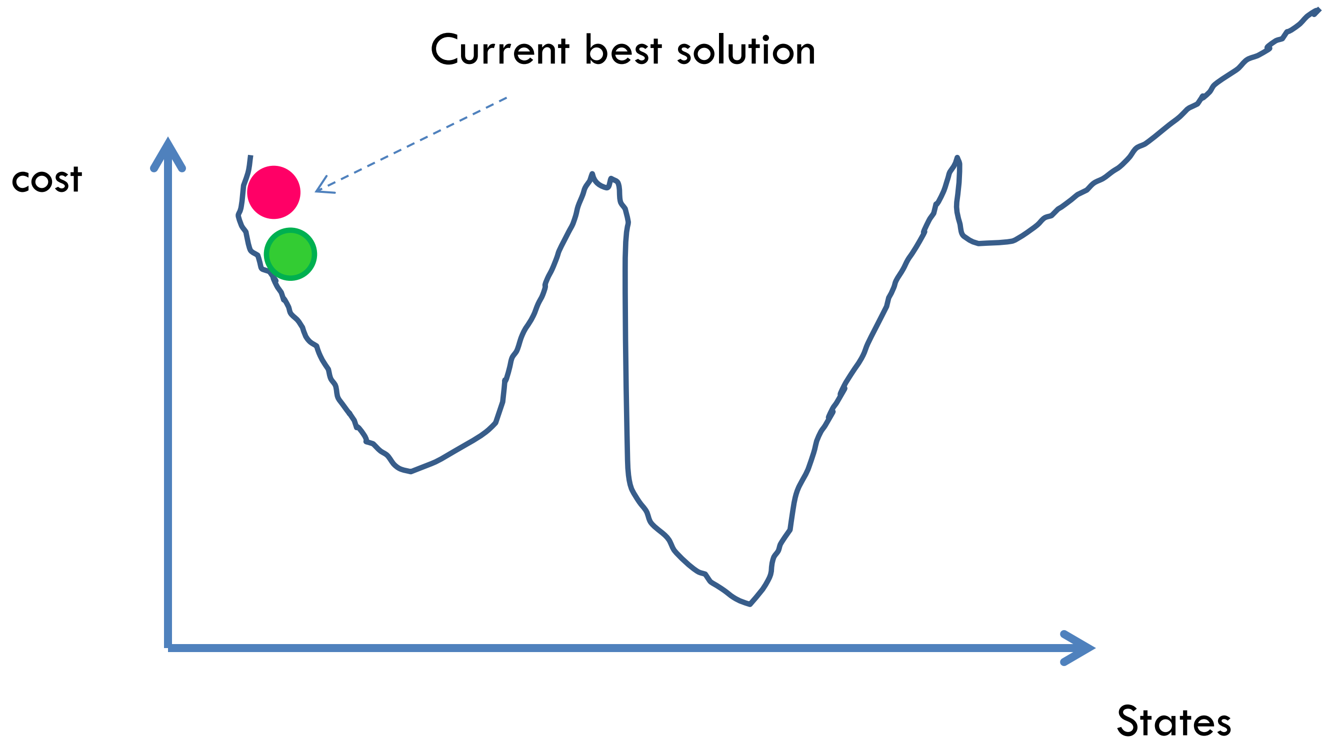


Hill-Climbing Search

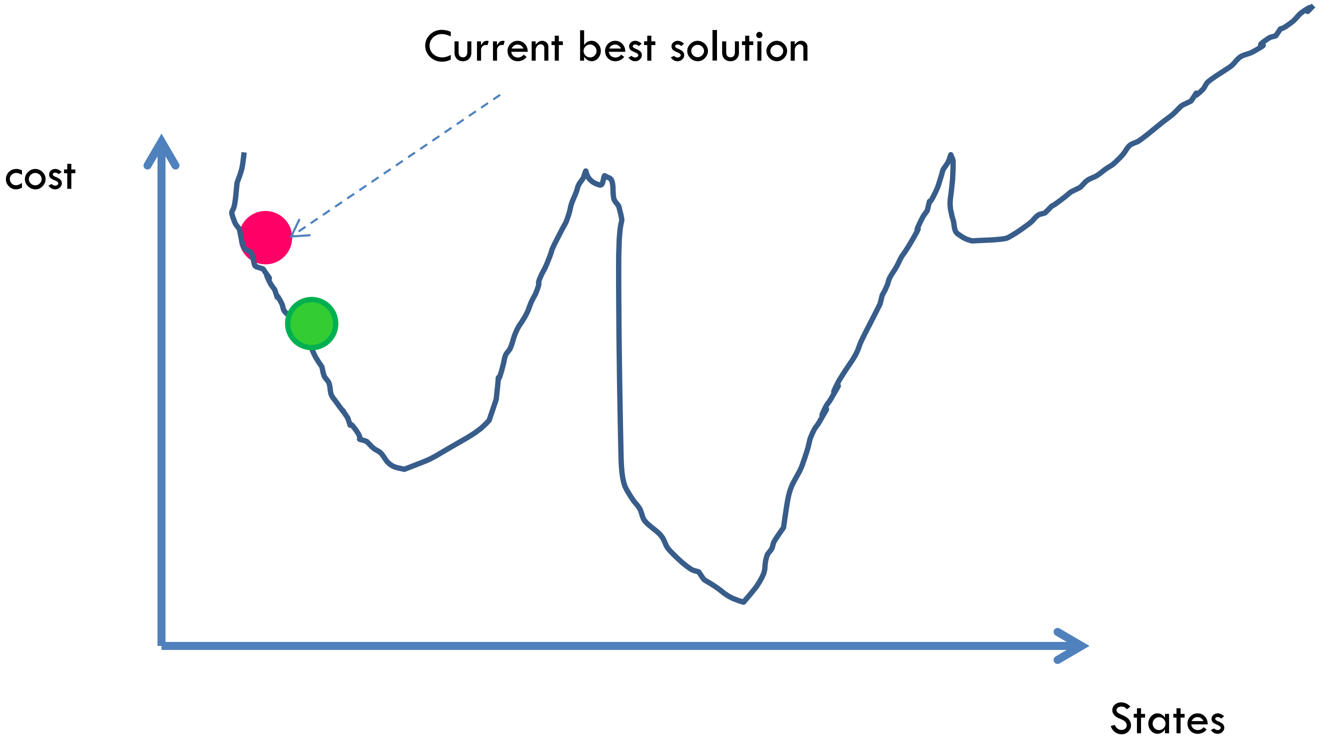
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   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
```

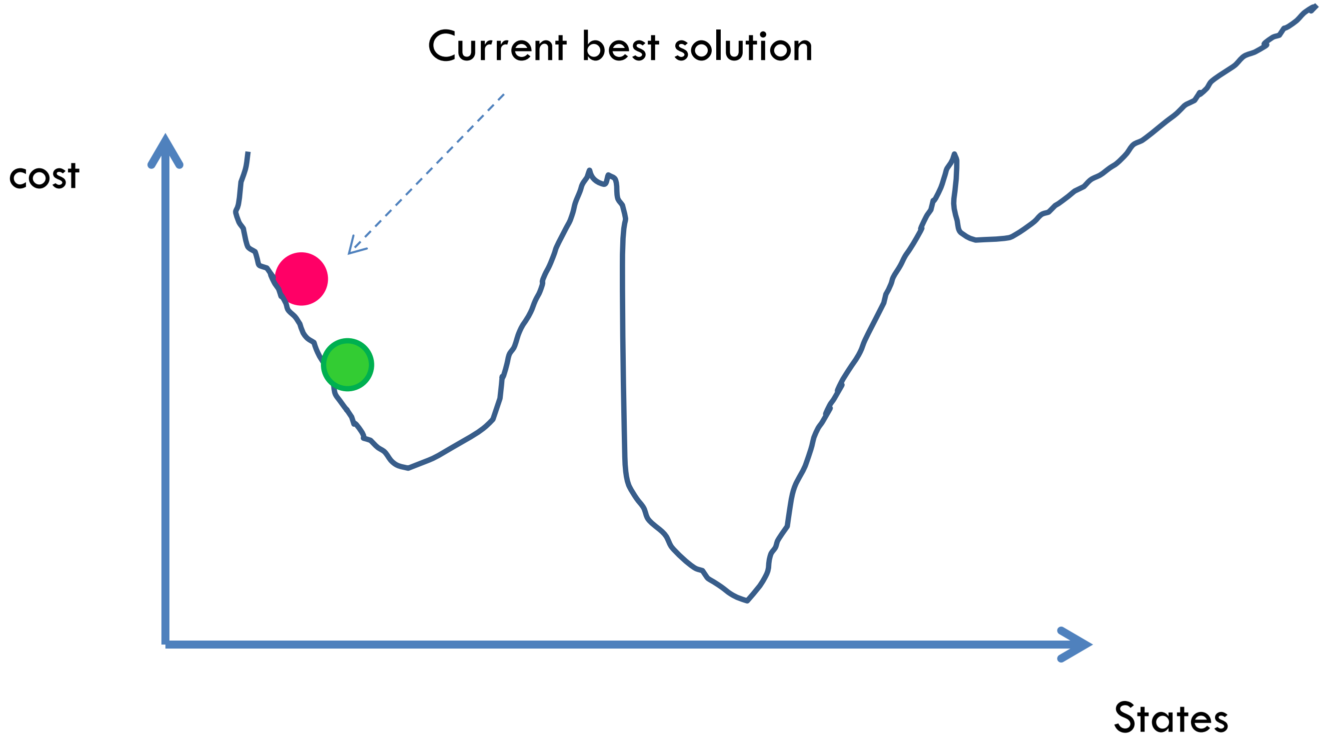
Hill-Climbing in Action …



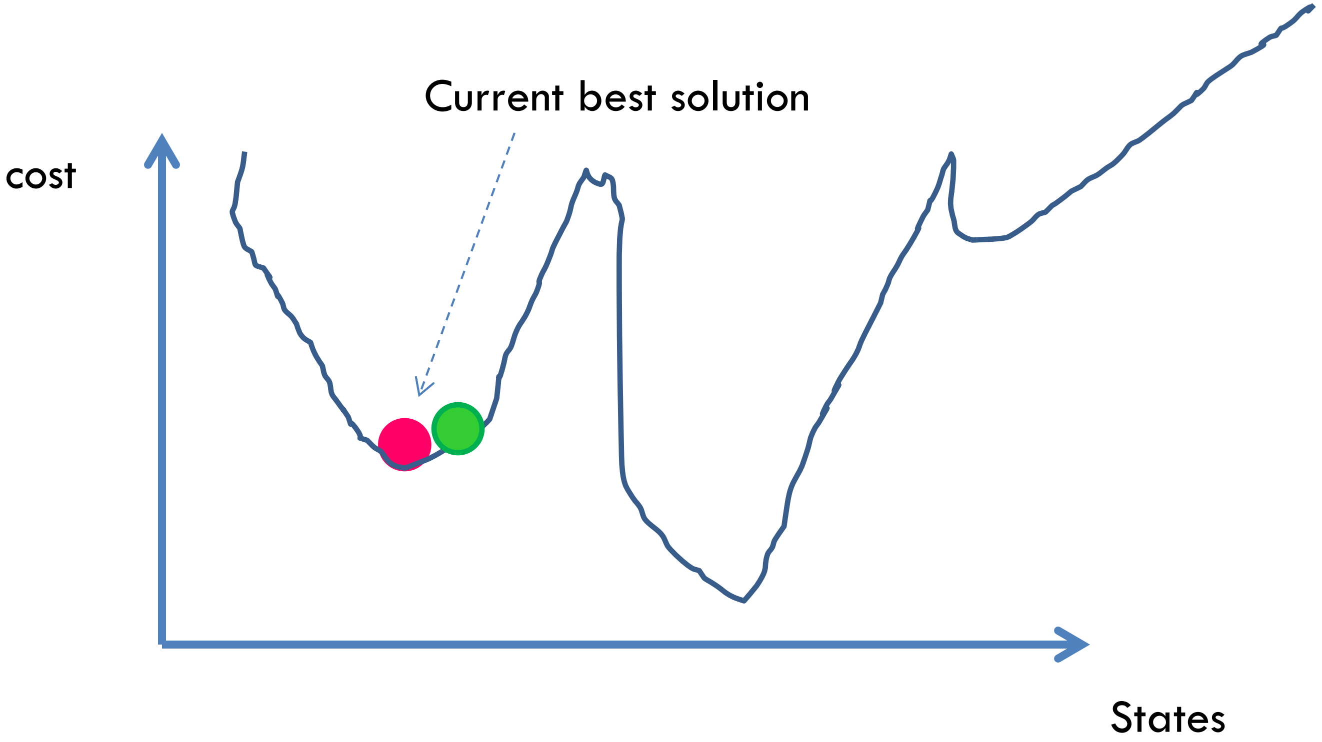
Hill-Climbing in Action …



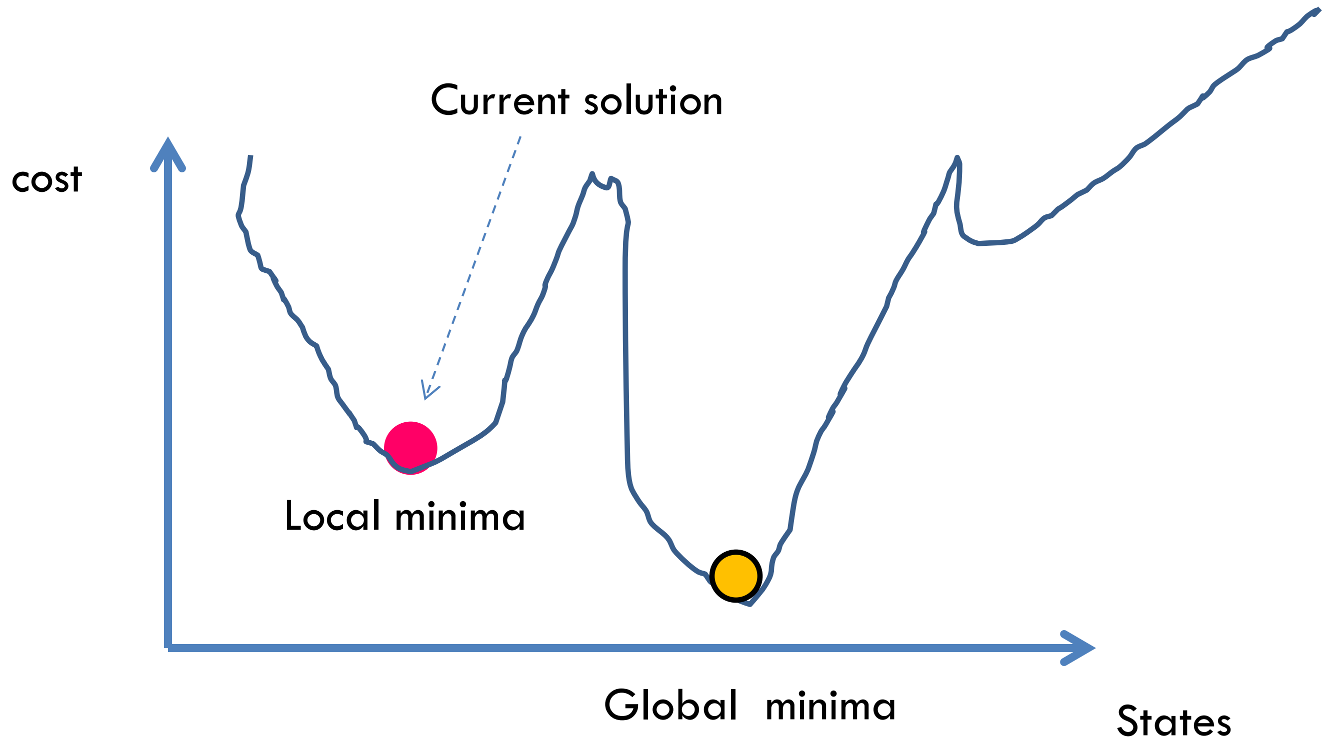
Hill-Climbing in Action …



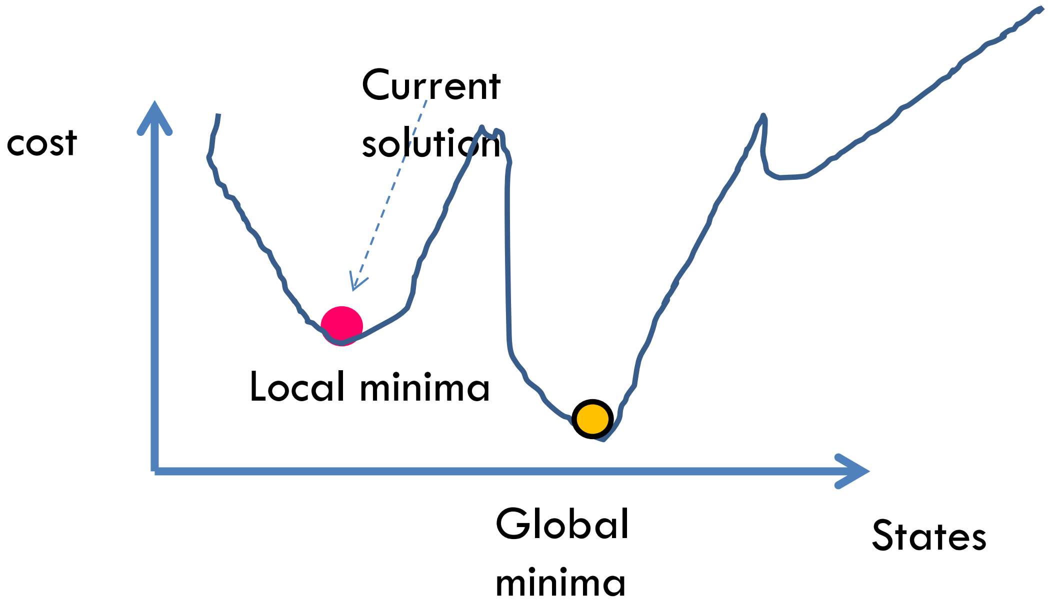
Hill-Climbing in Action …



Hill-Climbing in Action …



Hill-Climbing in Action …

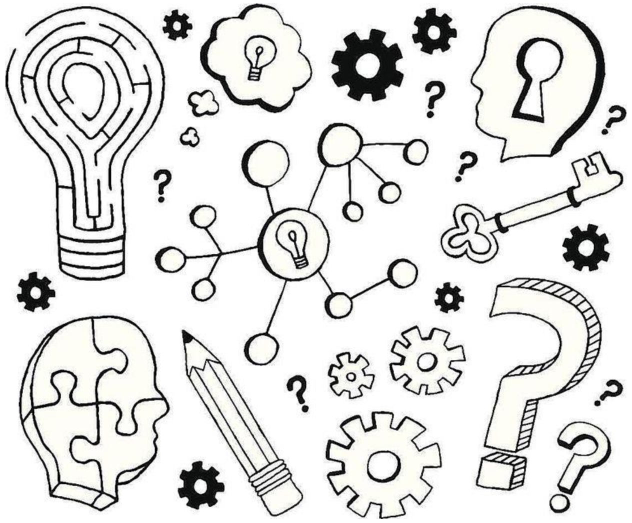


Drawback: Depending on initial state, it can get stuck in local maxima/minimum or flat local maximum and not find the solution.

Cure: Random restart.

Activity (Reflection, 20’)

Reflect on what you have studied in **Hill-Climbing Search.**



Simulated Annealing Search

The Problem

- Most minimization strategies find the *nearest* local minimum
- Standard strategy
  - Generate trial point based on current estimates
  - Evaluate function at proposed location
  - Accept new value if it improves solution

The Solution

- We need a strategy to find other minima
- This means, we must sometimes select new points that do not improve solution
- How?

Annealing

- One manner in which crystals are formed
- Gradual cooling of liquid …
    - ✓ At high temperatures, molecules move freely
    - ✓ At low temperatures, molecules are "stuck"
    - ✓ If cooling is slow

Low energy, organized crystal lattice formed

Simulated annealing Search

- **Main Idea**: escape local maxima by allowing some "bad" moves but gradually **decrease their frequency**.

- Instead of picking the **best** move, it picks a **random** move..

Simulated annealing Search

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 $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

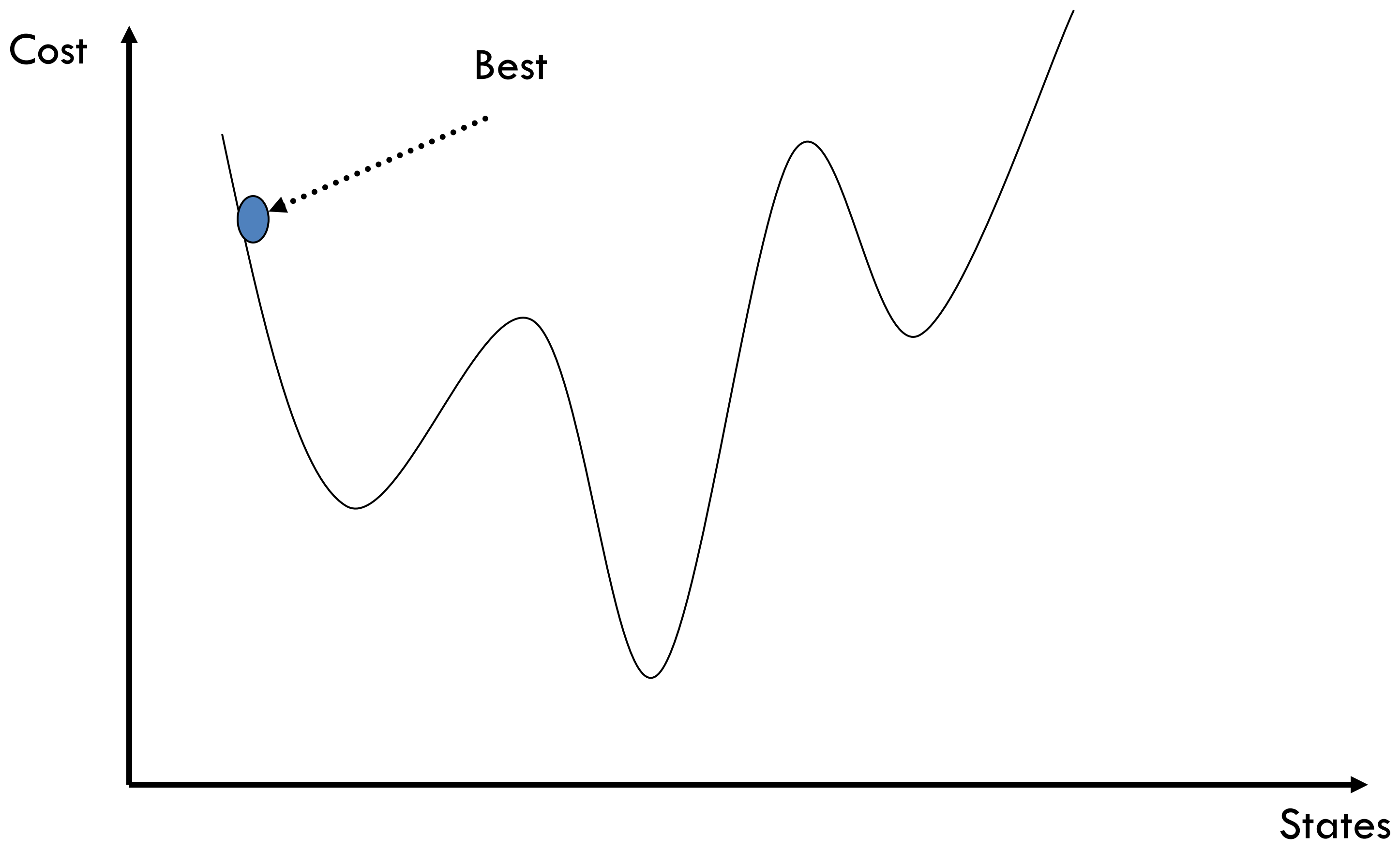
Similar to hill climbing, but a random move instead of best move.

Case of improvement, make the move.

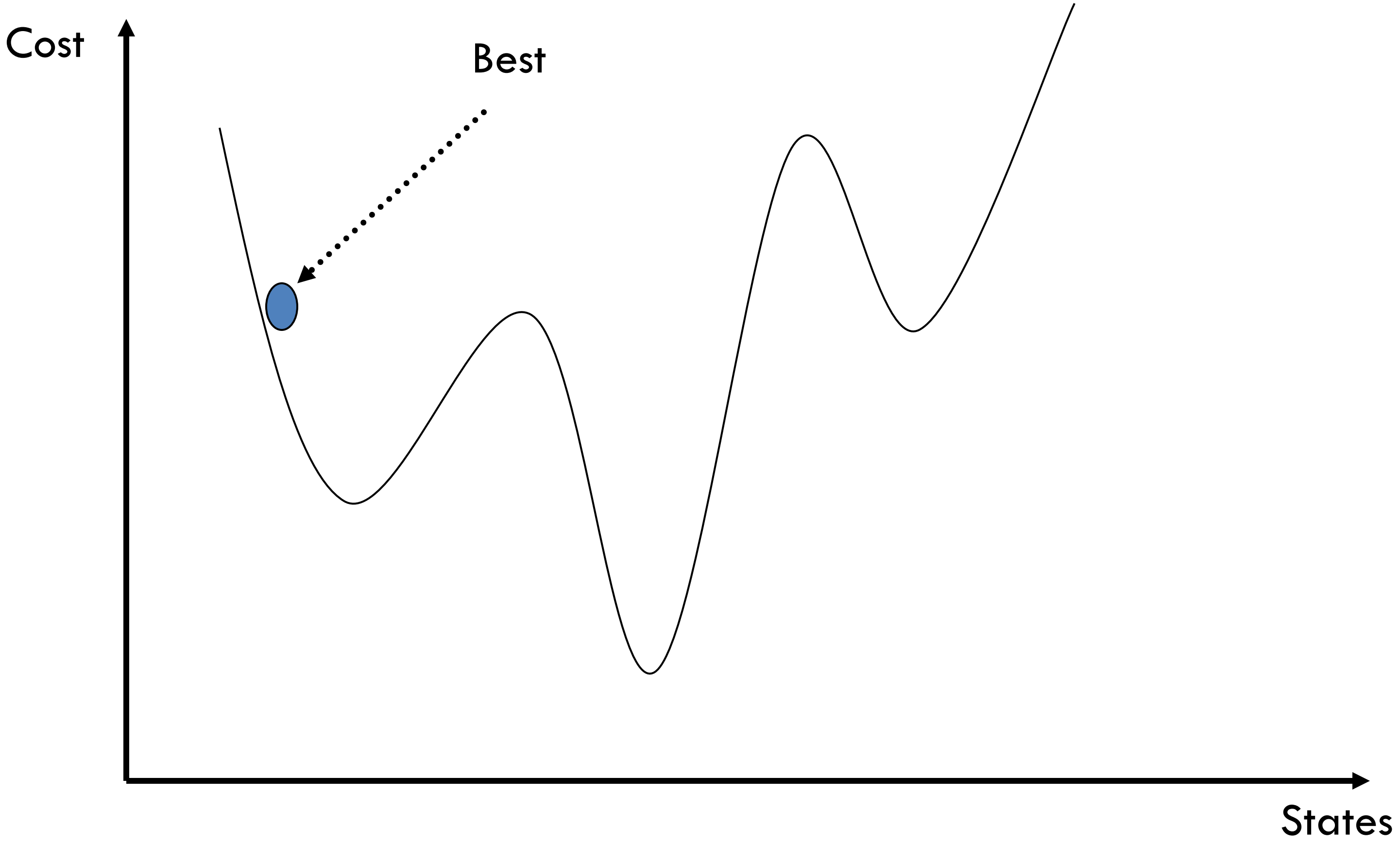
Otherwise, choose the move with probability that decreases exponentially with the "badness" of the move.

- say the change in objective function is δ
- if δ is positive, then move to that state
- **otherwise**:
  - move to this state with probability proportional to δ
  - thus: worse moves (very large negative δ) are executed less often

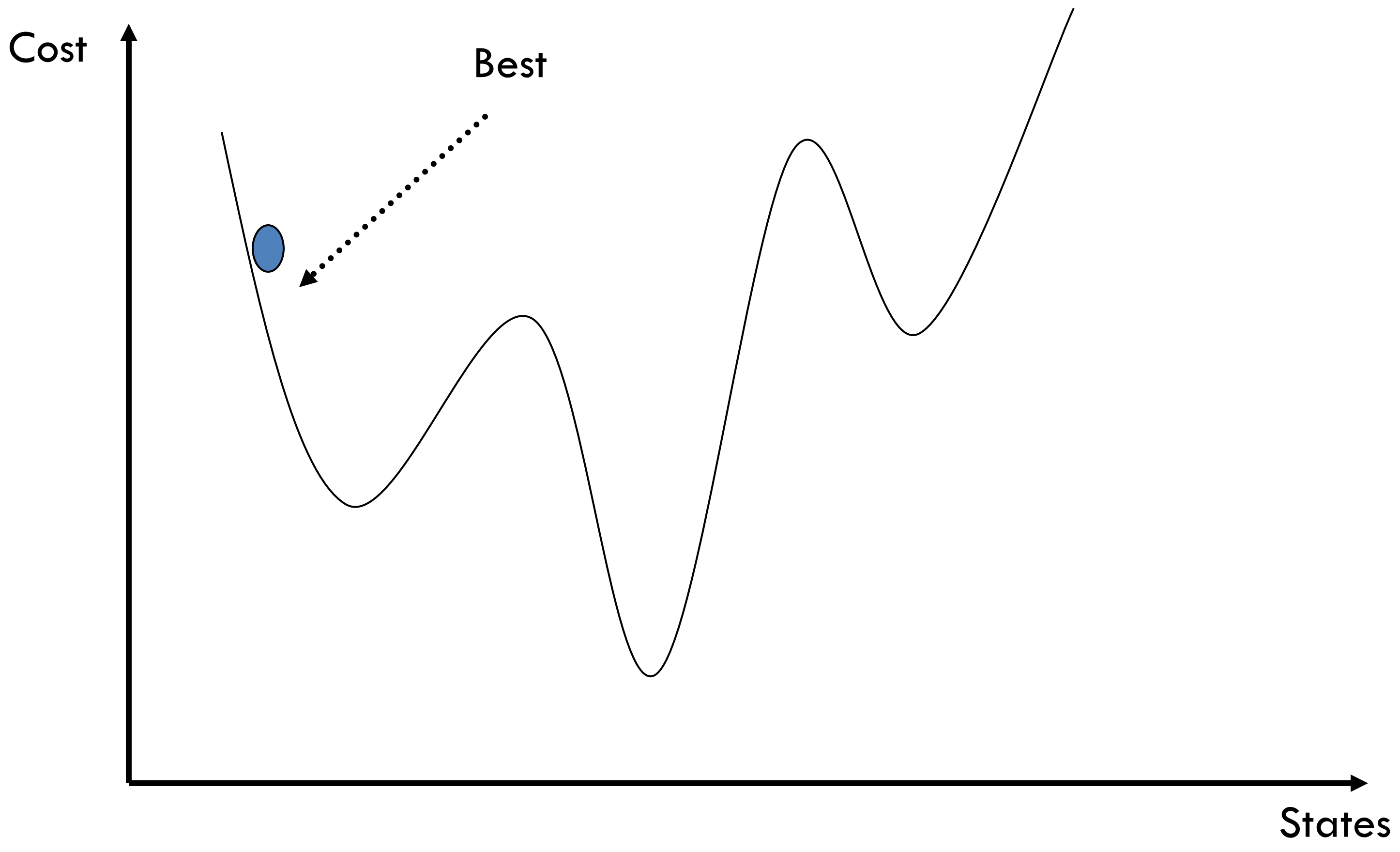
Simulated Annealing



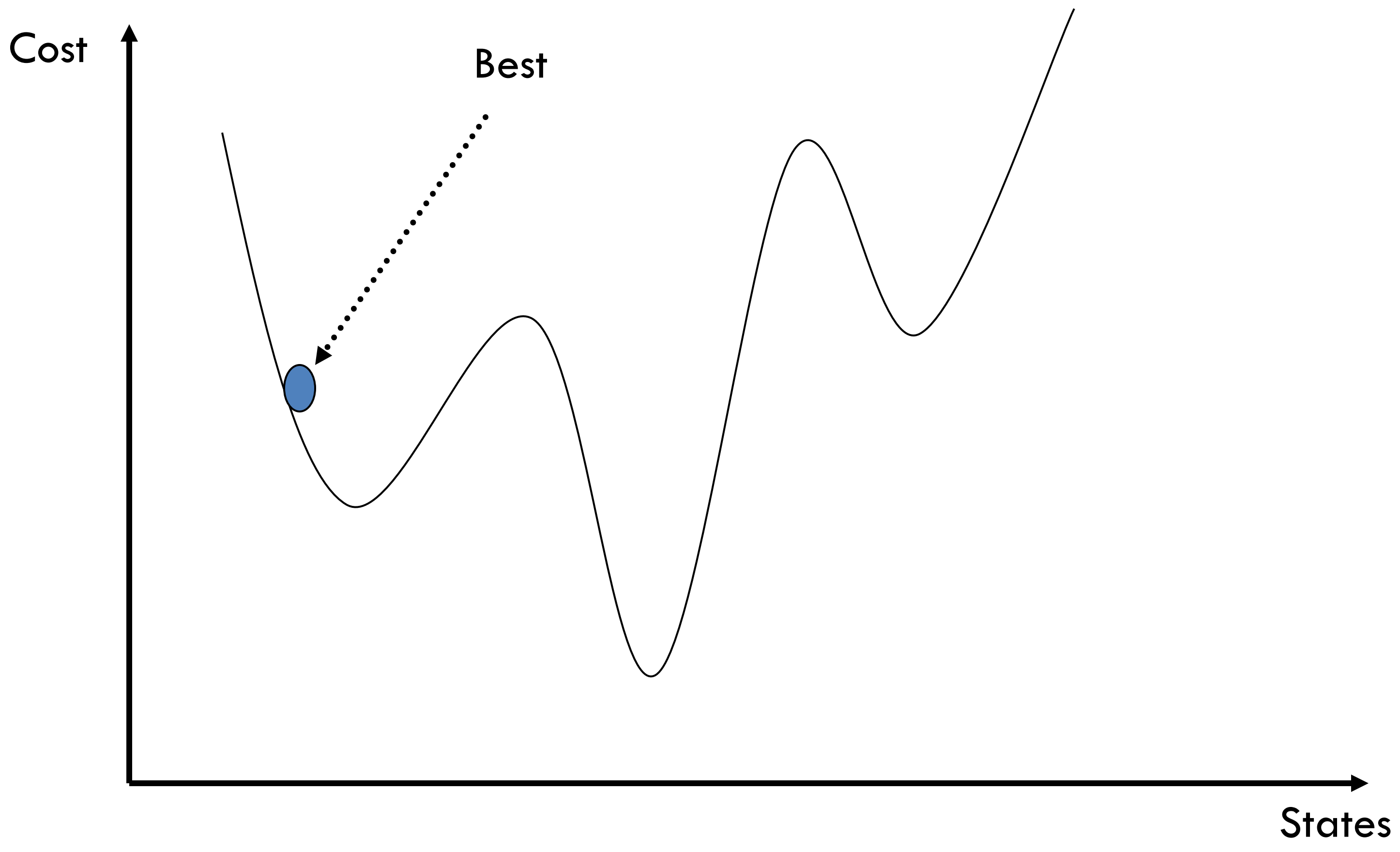
Simulated Annealing



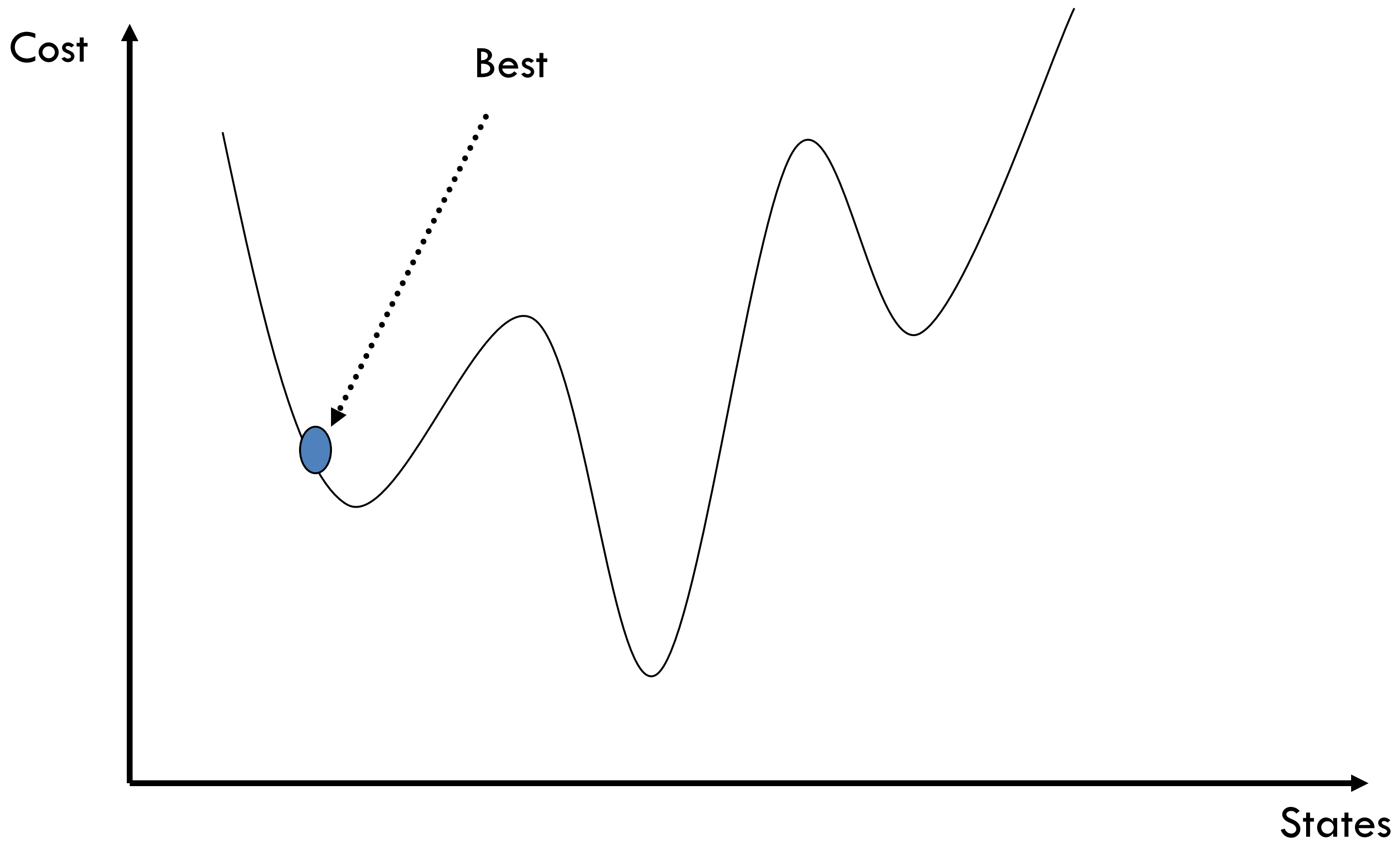
Simulated Annealing



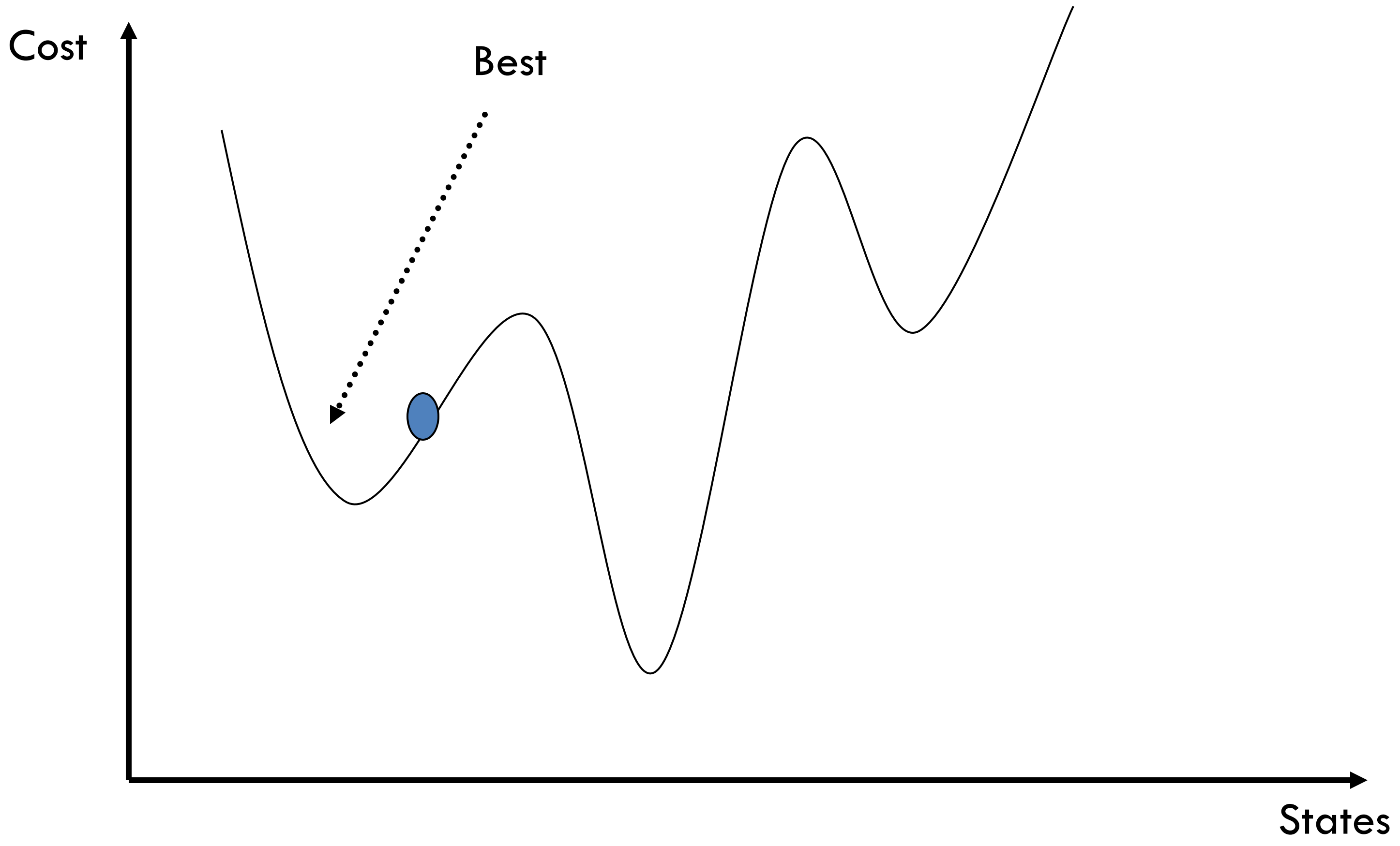
Simulated Annealing



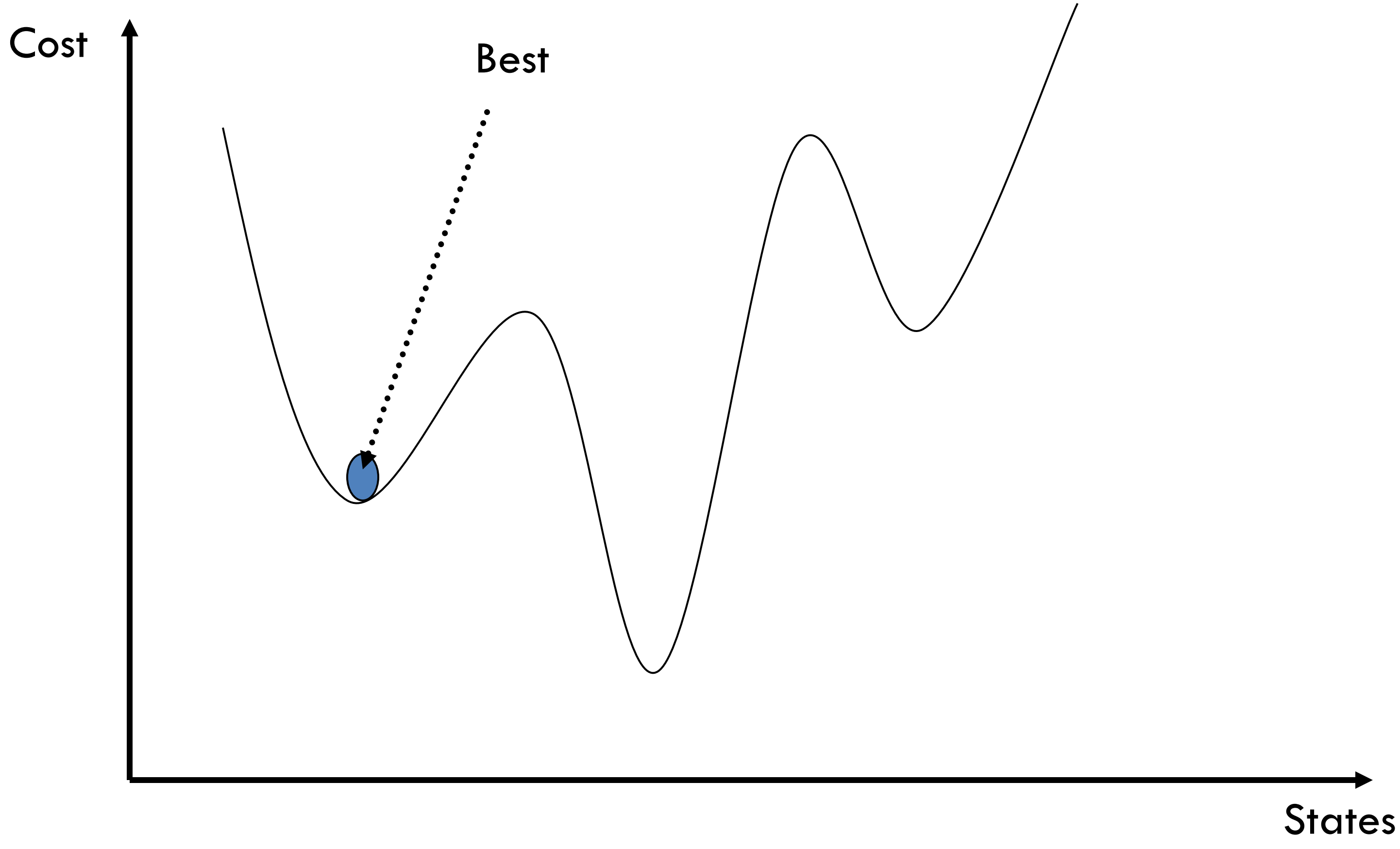
Simulated Annealing



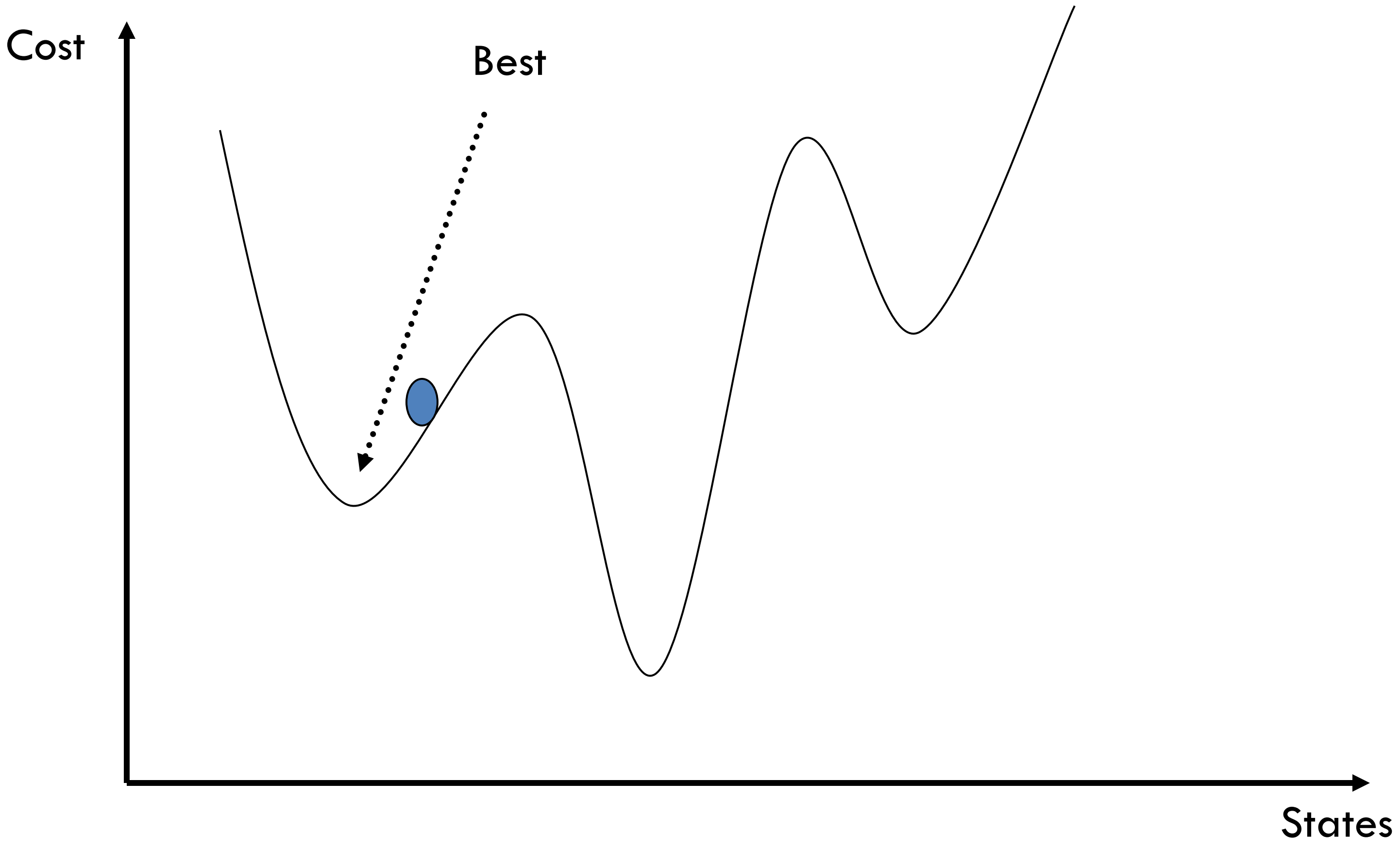
Simulated Annealing



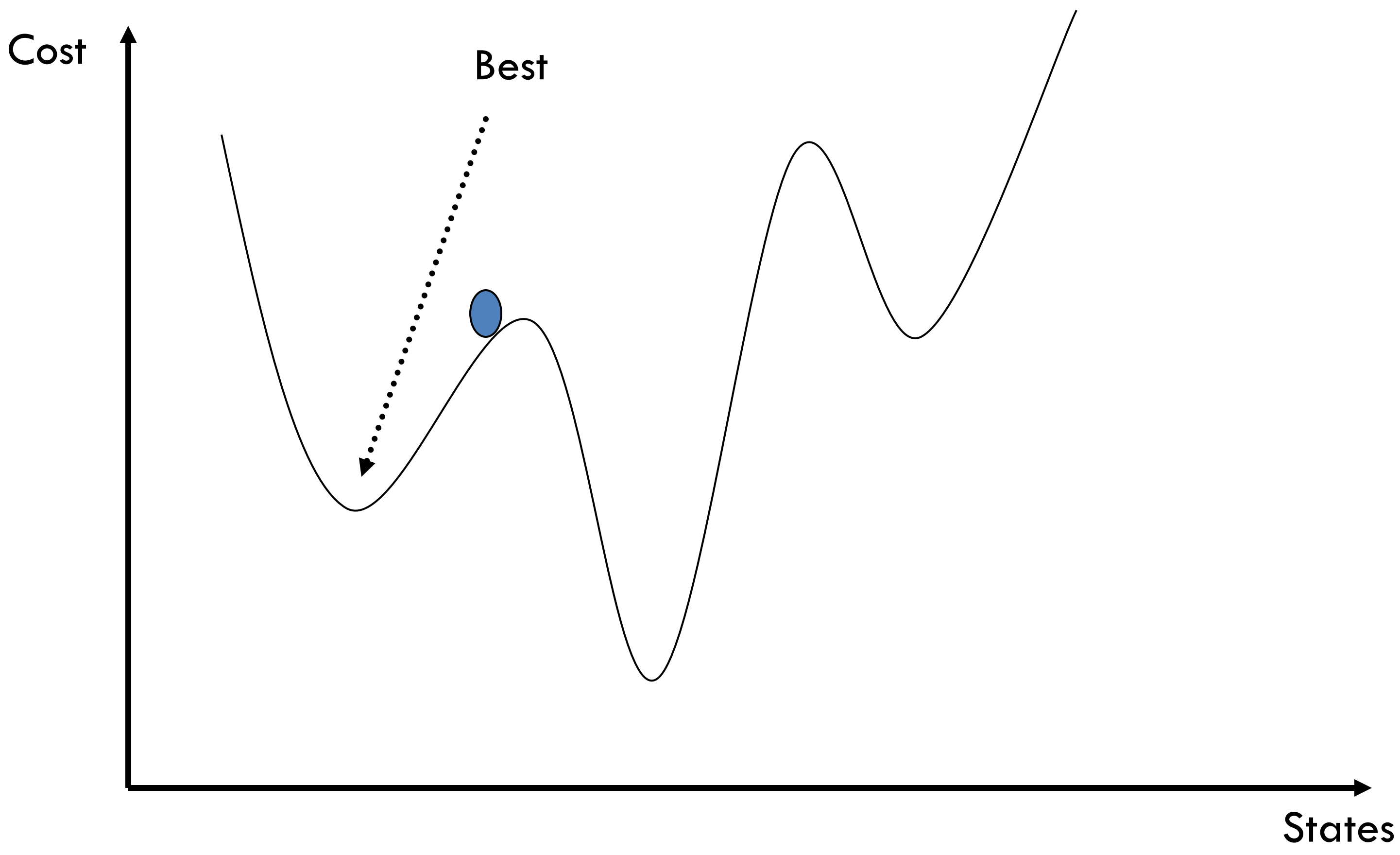
Simulated Annealing



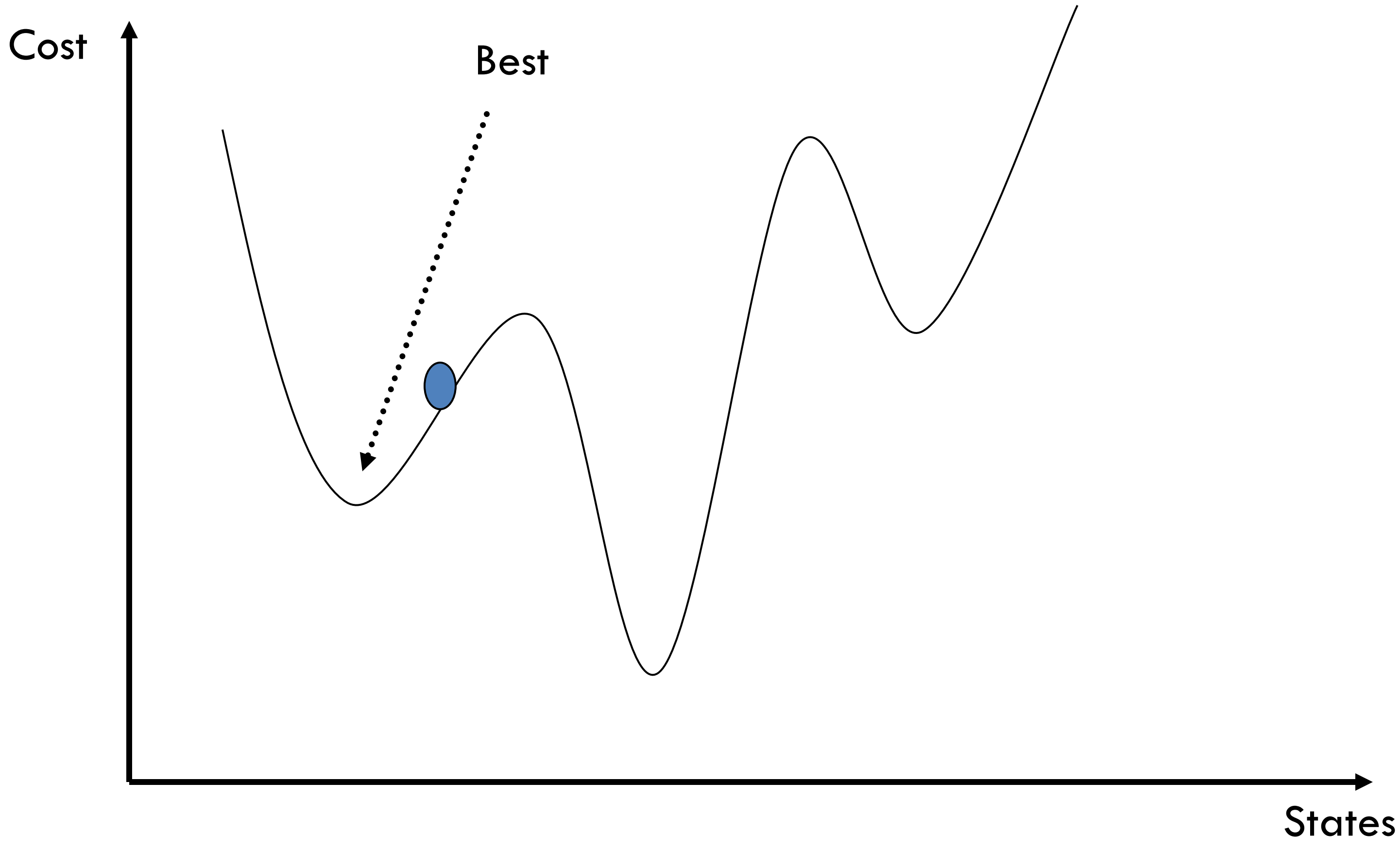
Simulated Annealing



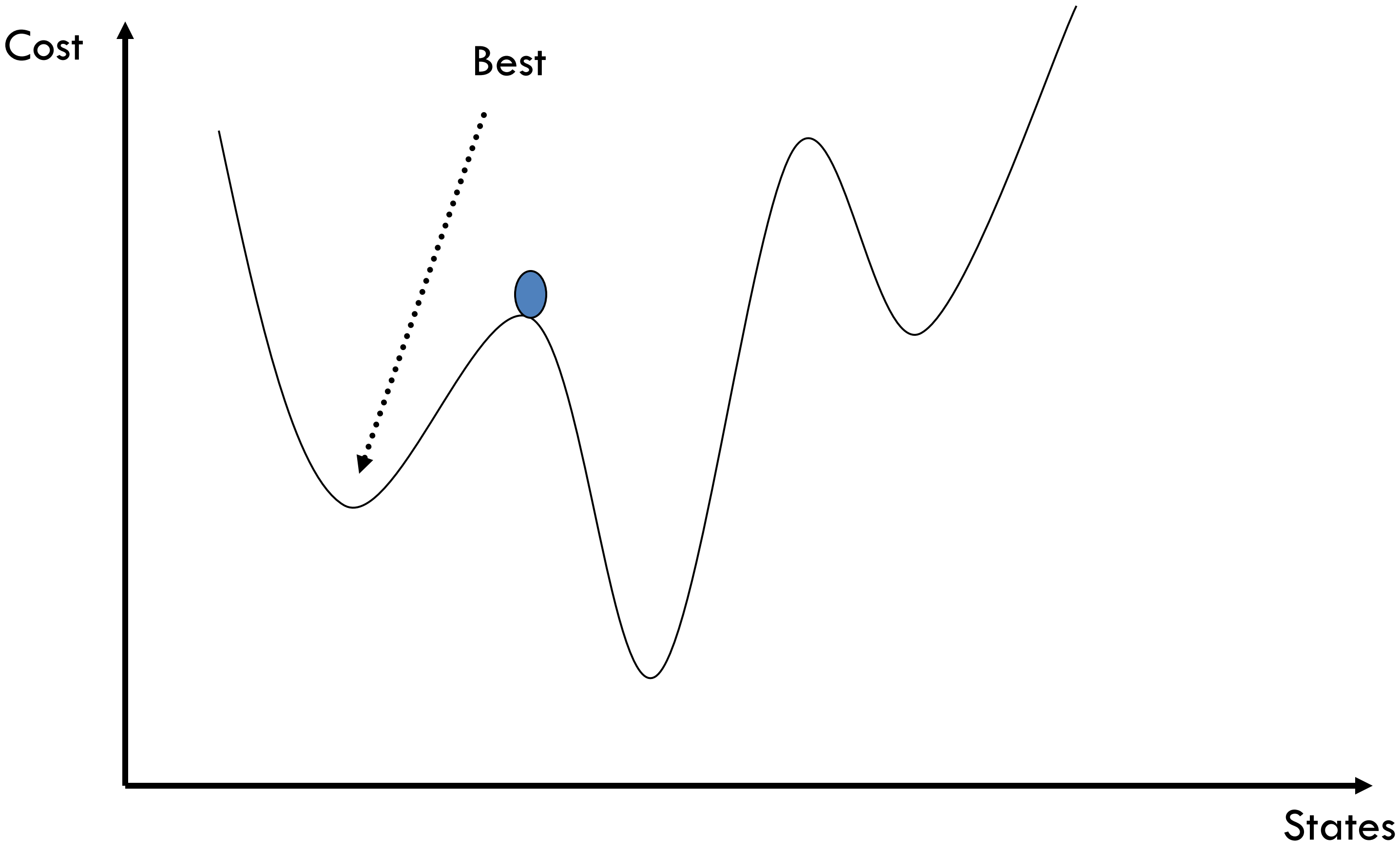
Simulated Annealing



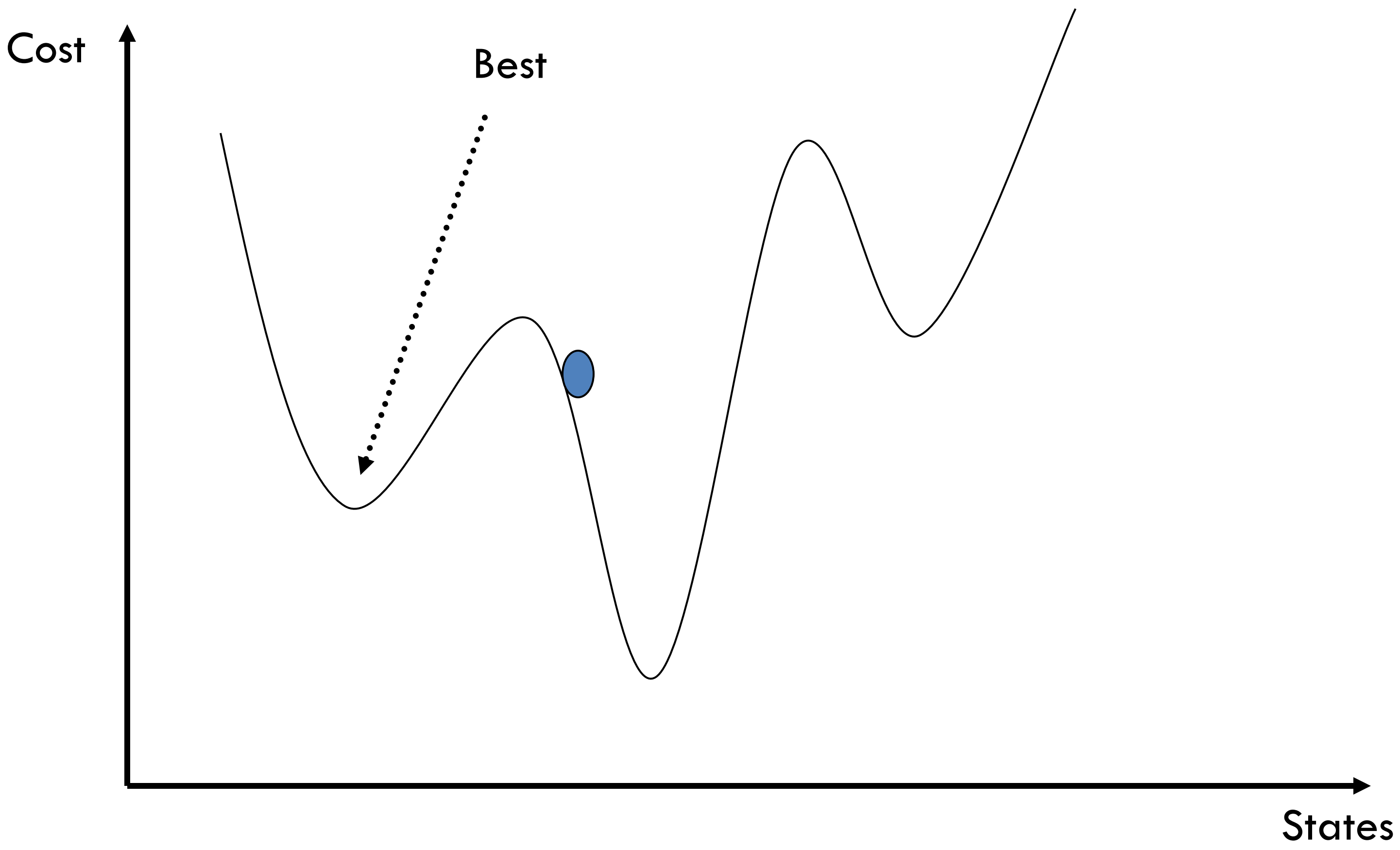
Simulated Annealing



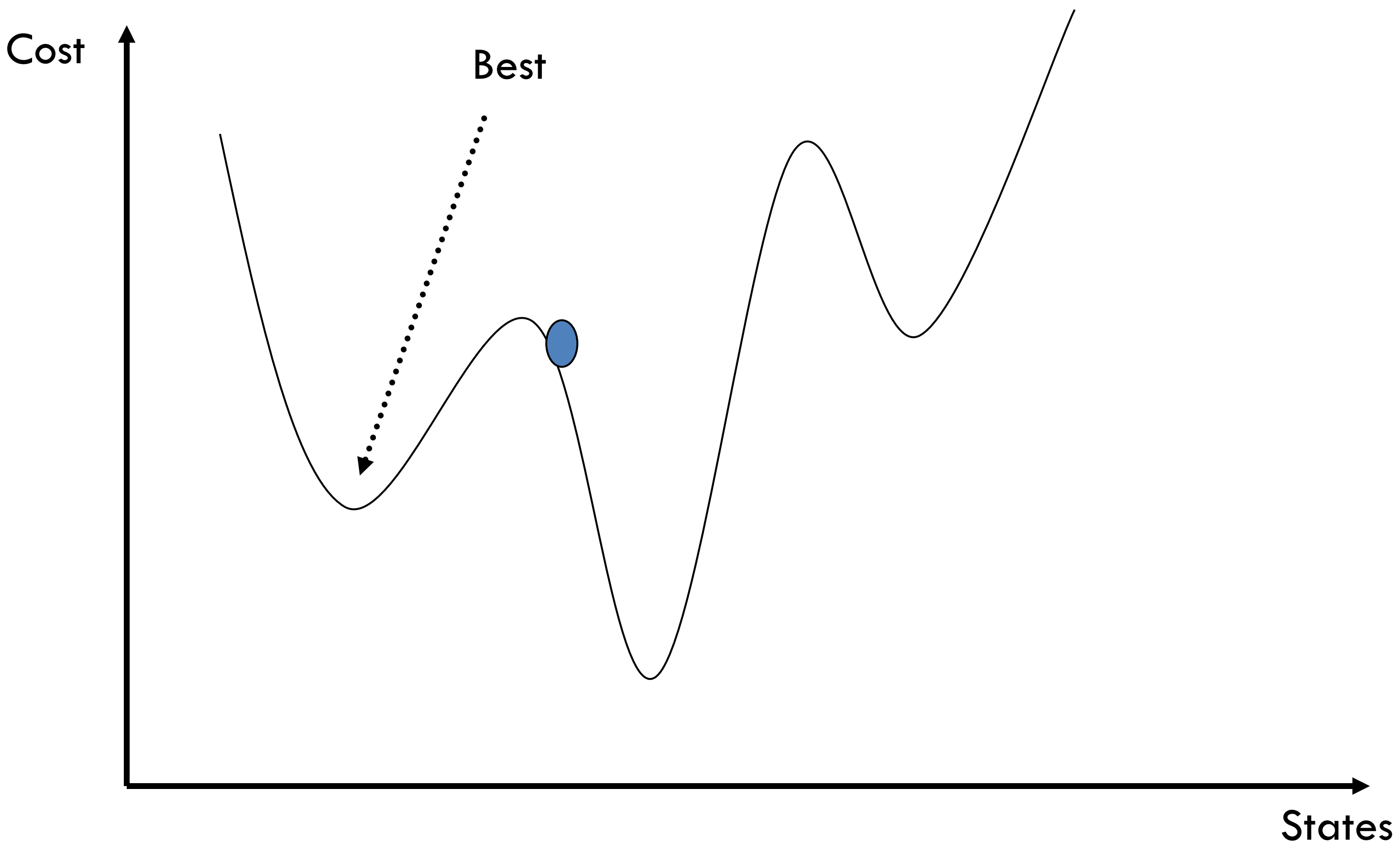
Simulated Annealing



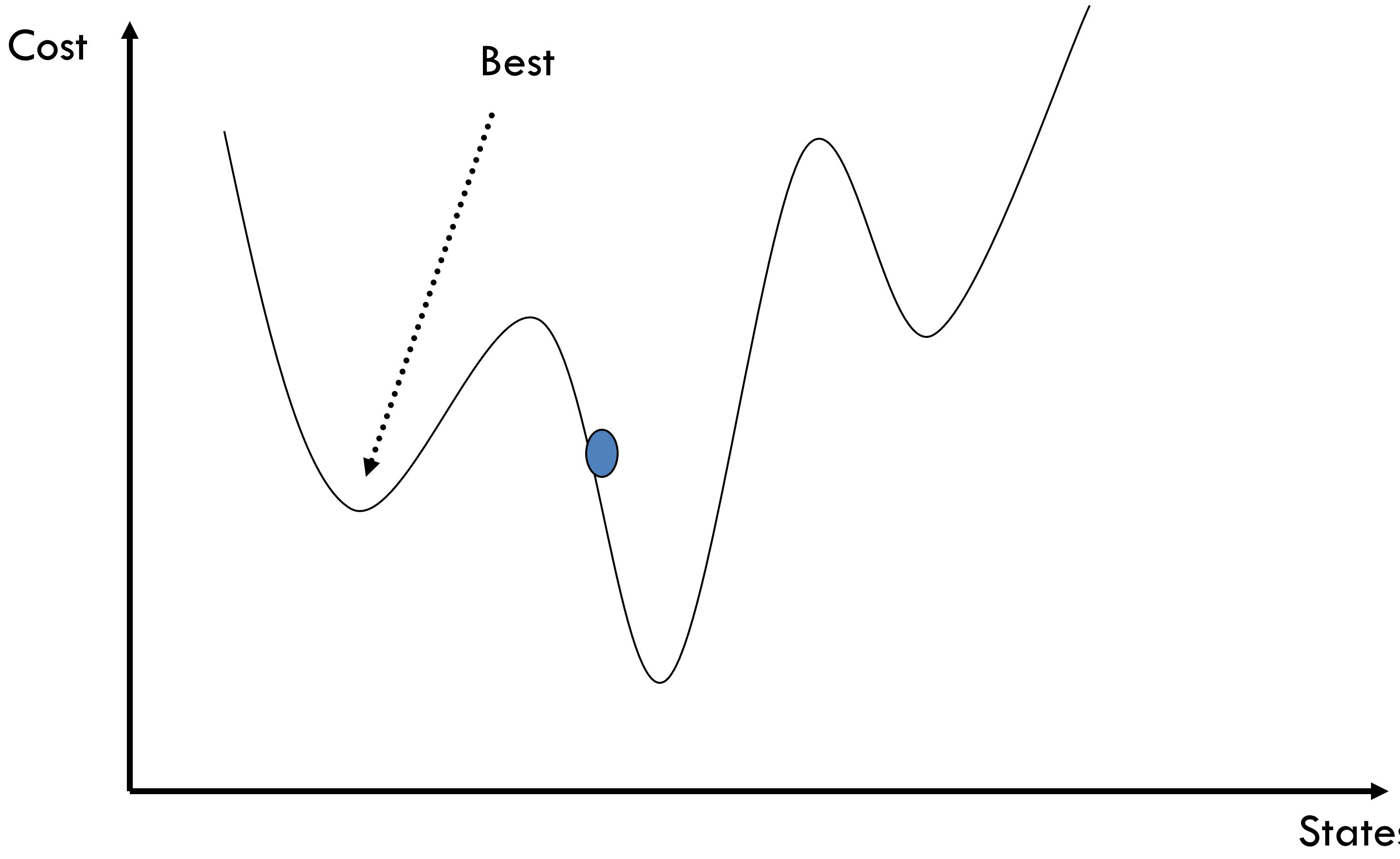
Simulated Annealing



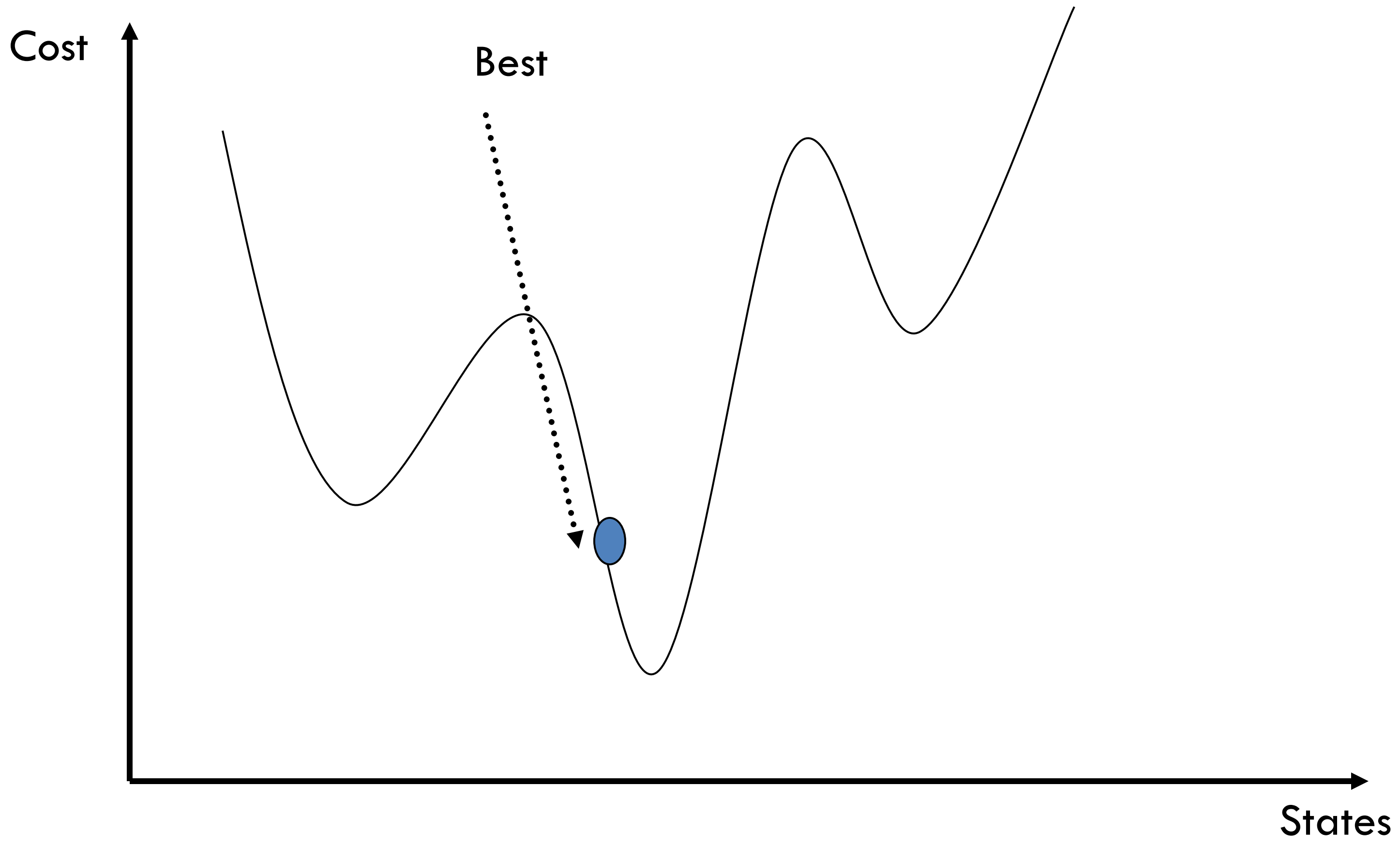
Simulated Annealing



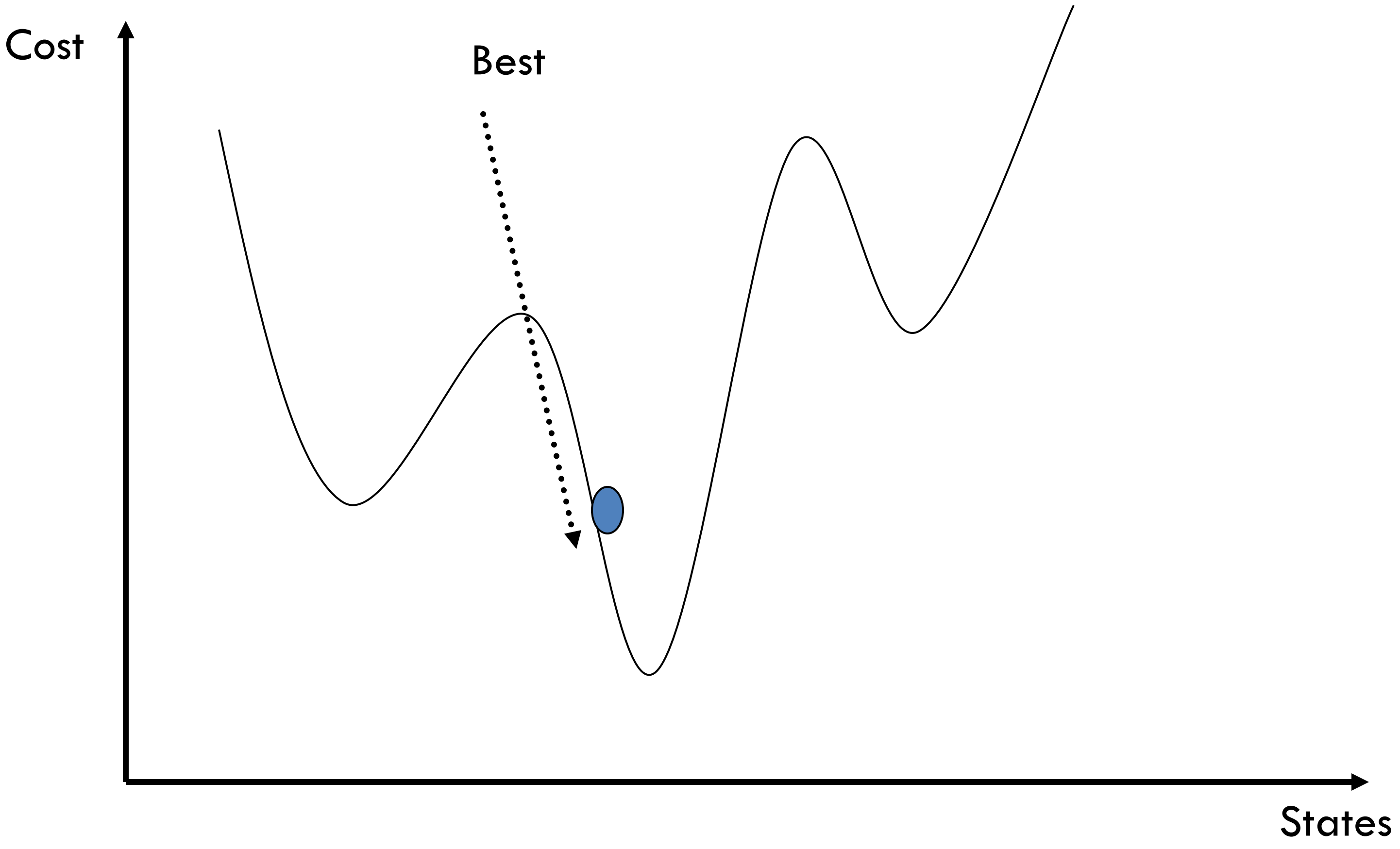
Simulated Annealing



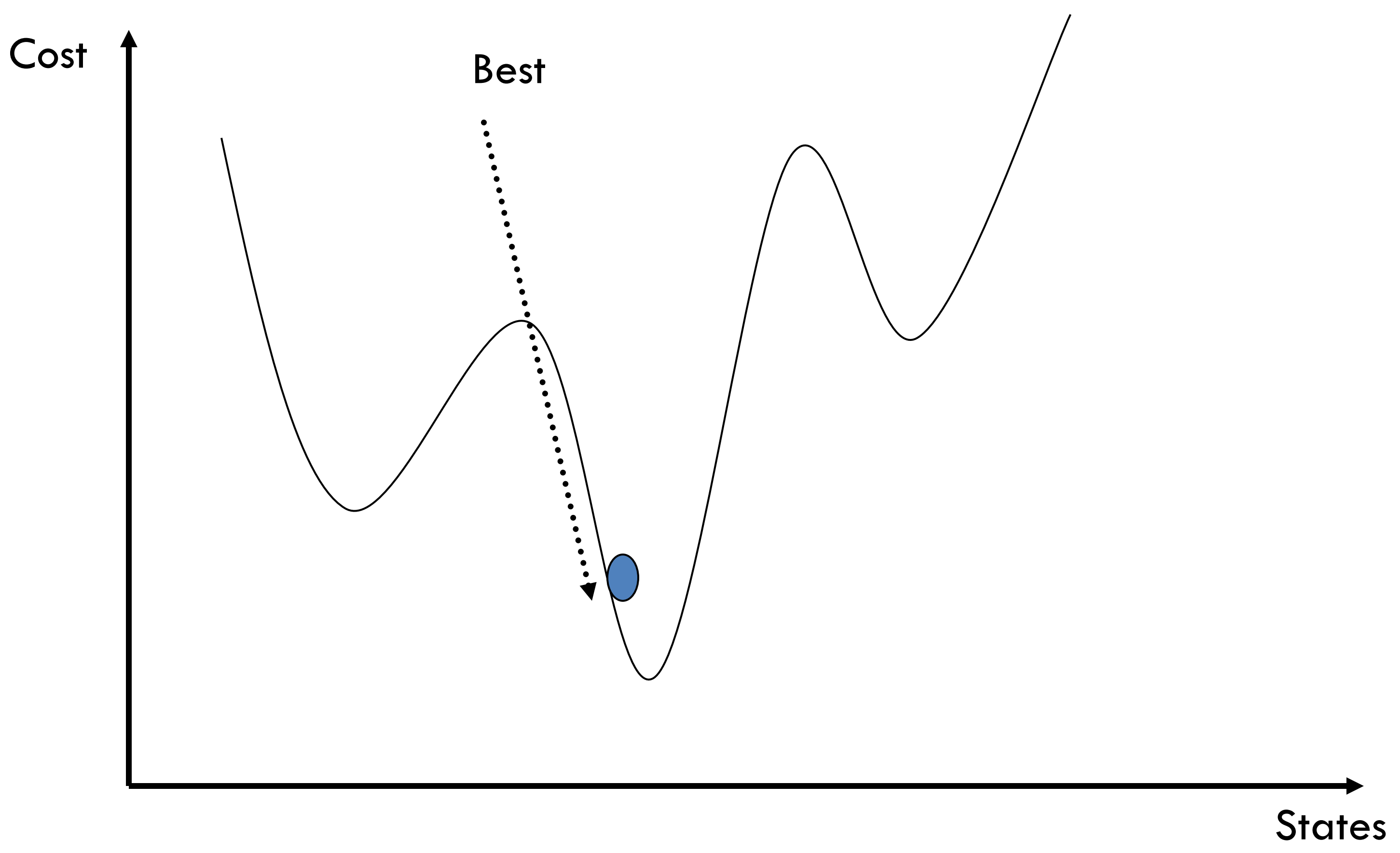
Simulated Annealing



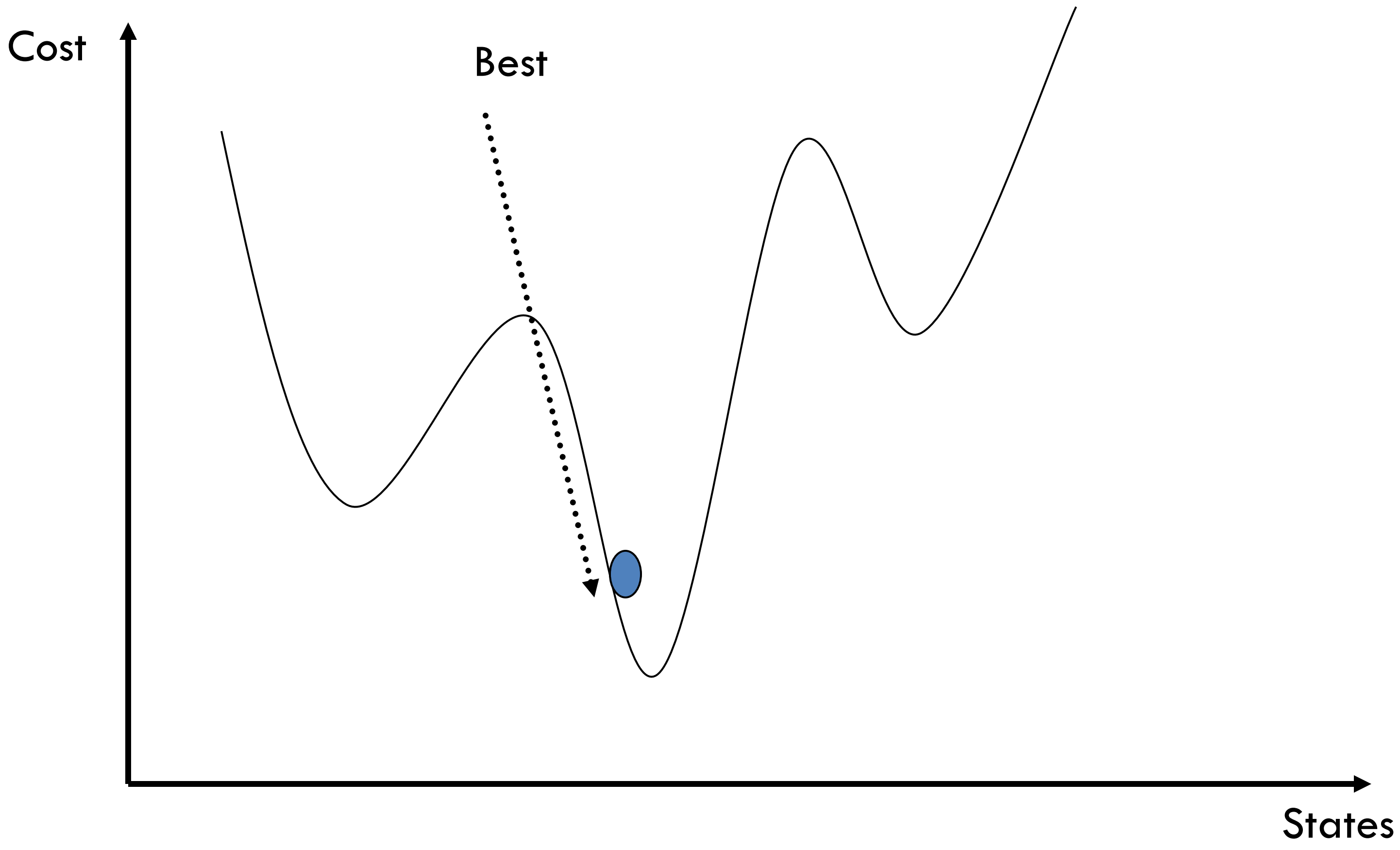
Simulated Annealing



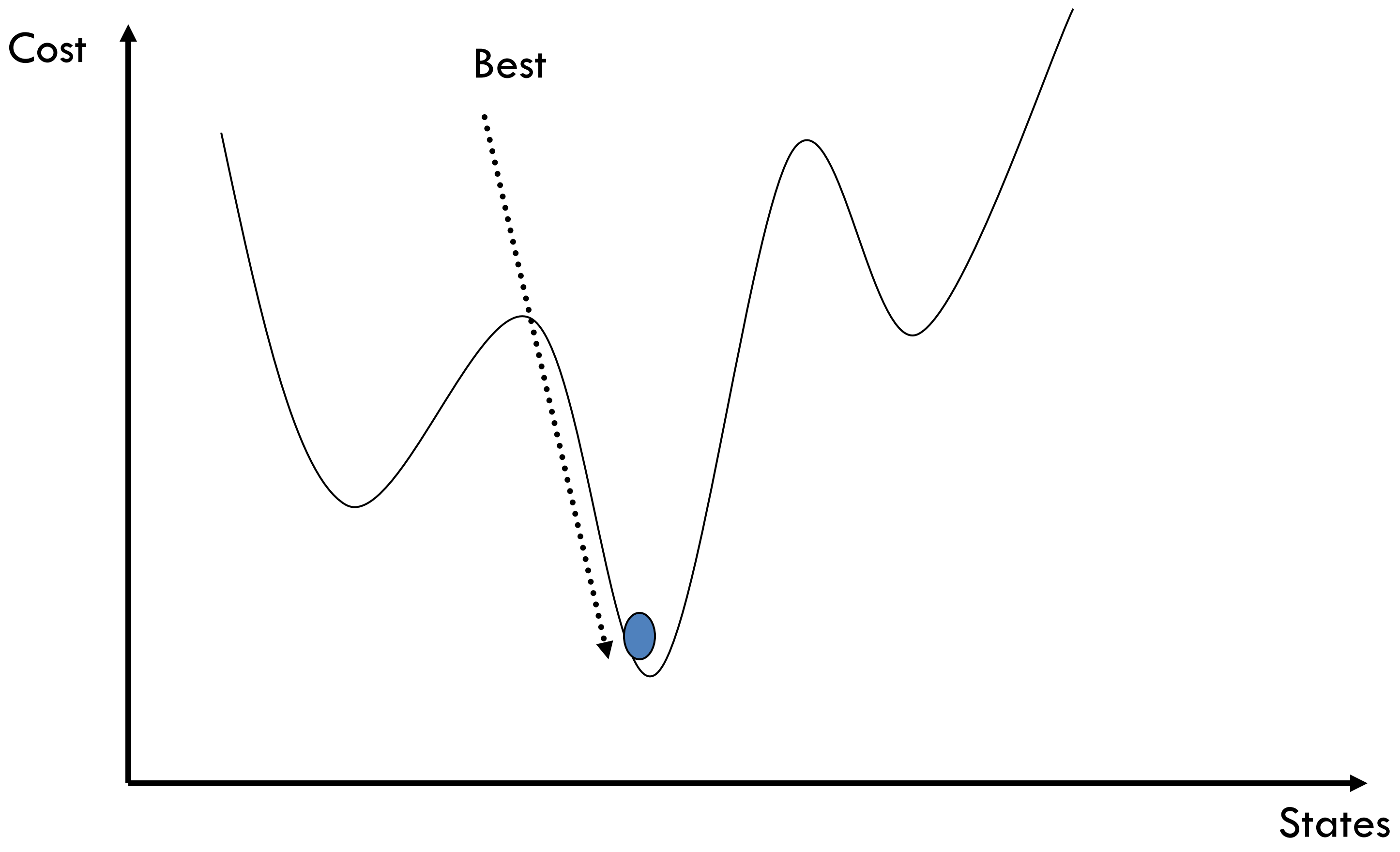
Simulated Annealing



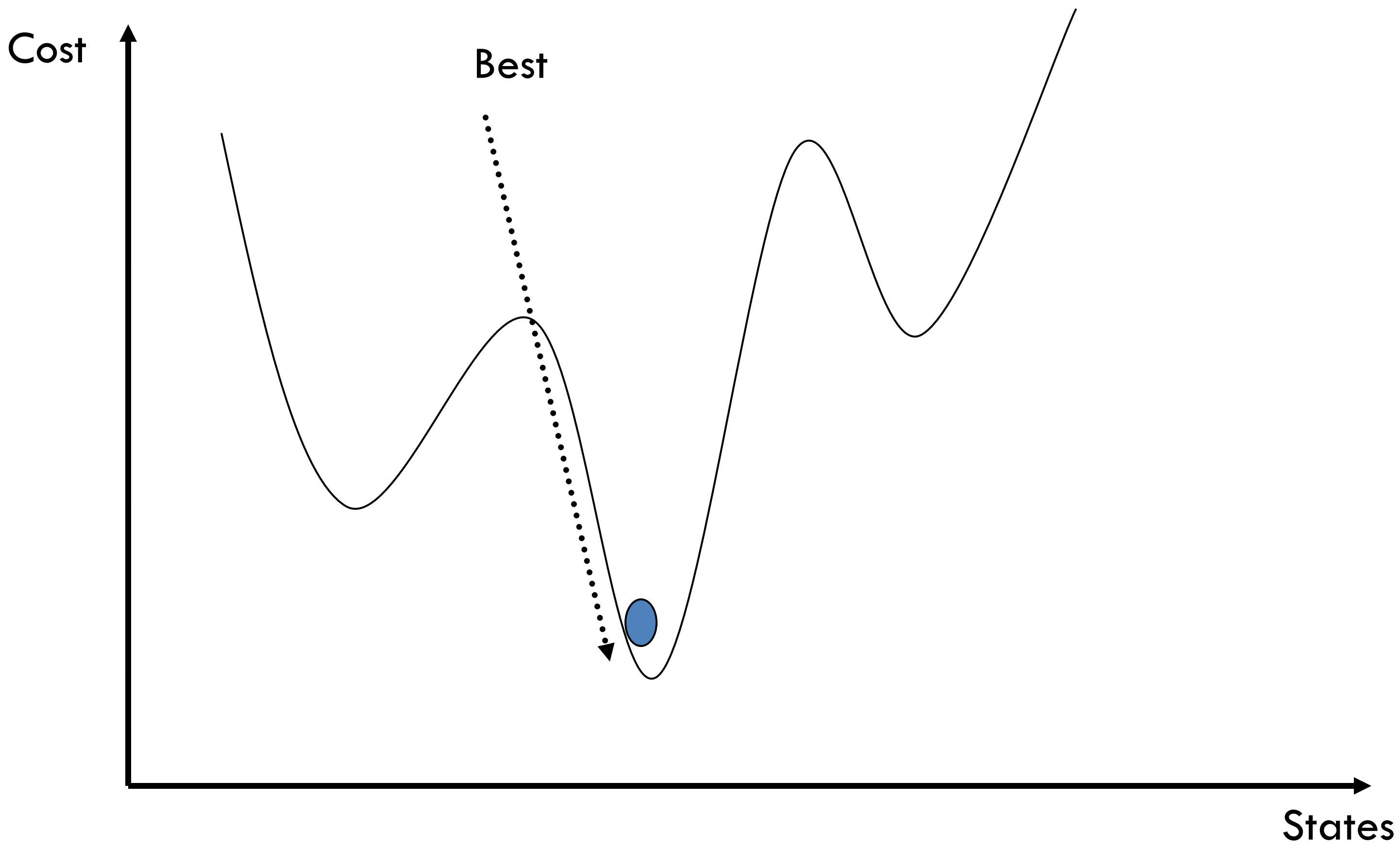
Simulated Annealing



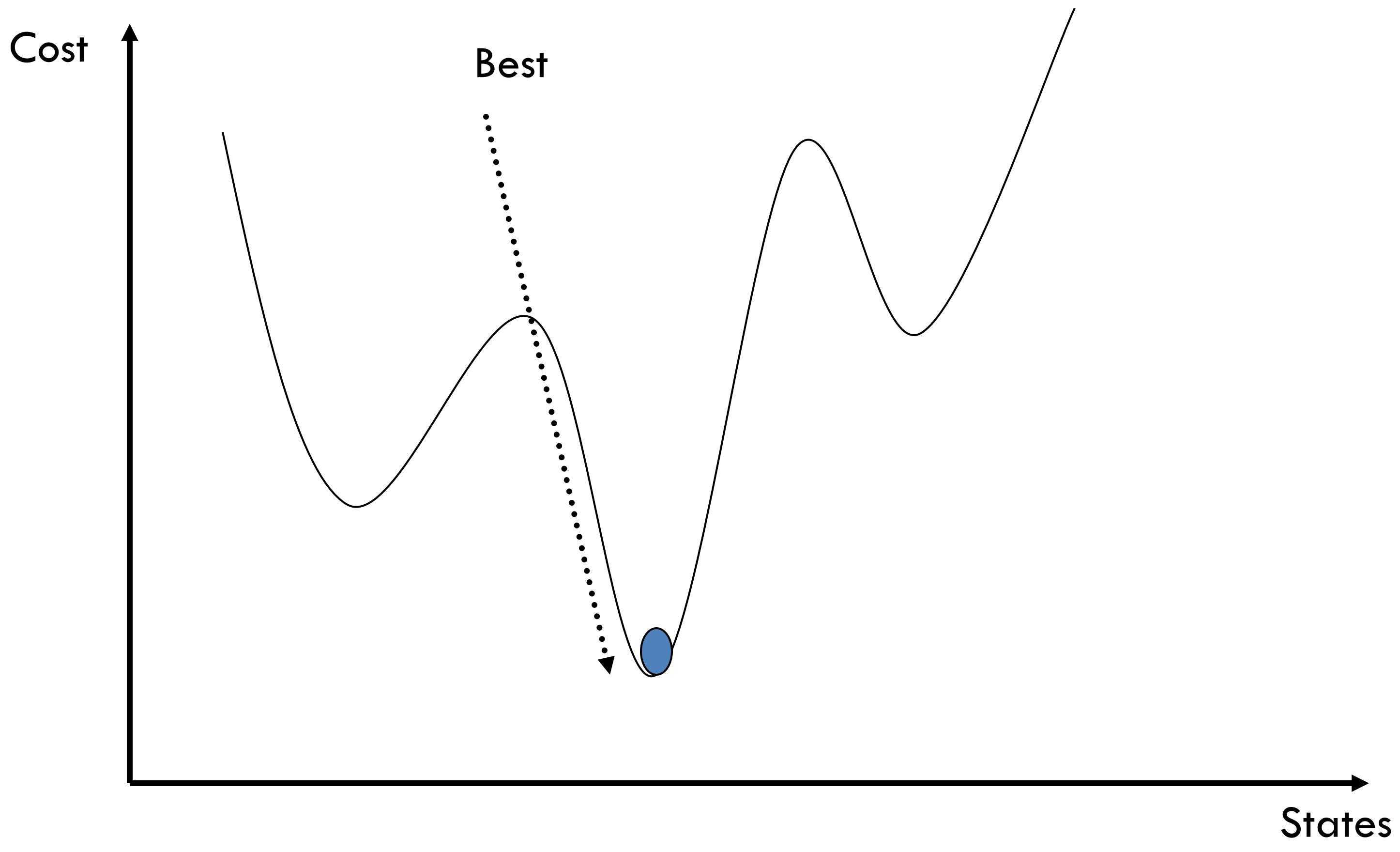
Simulated Annealing



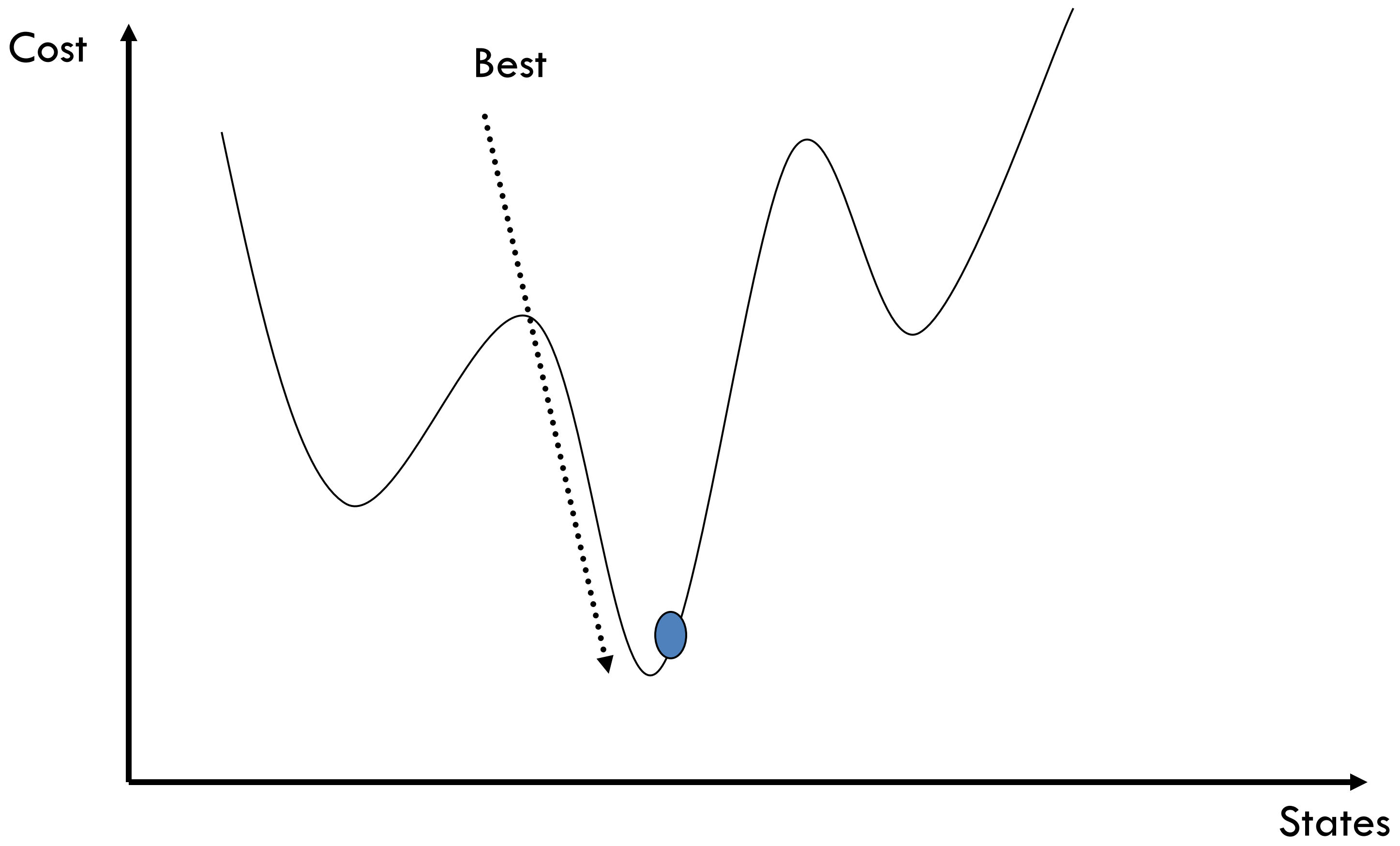
Simulated Annealing



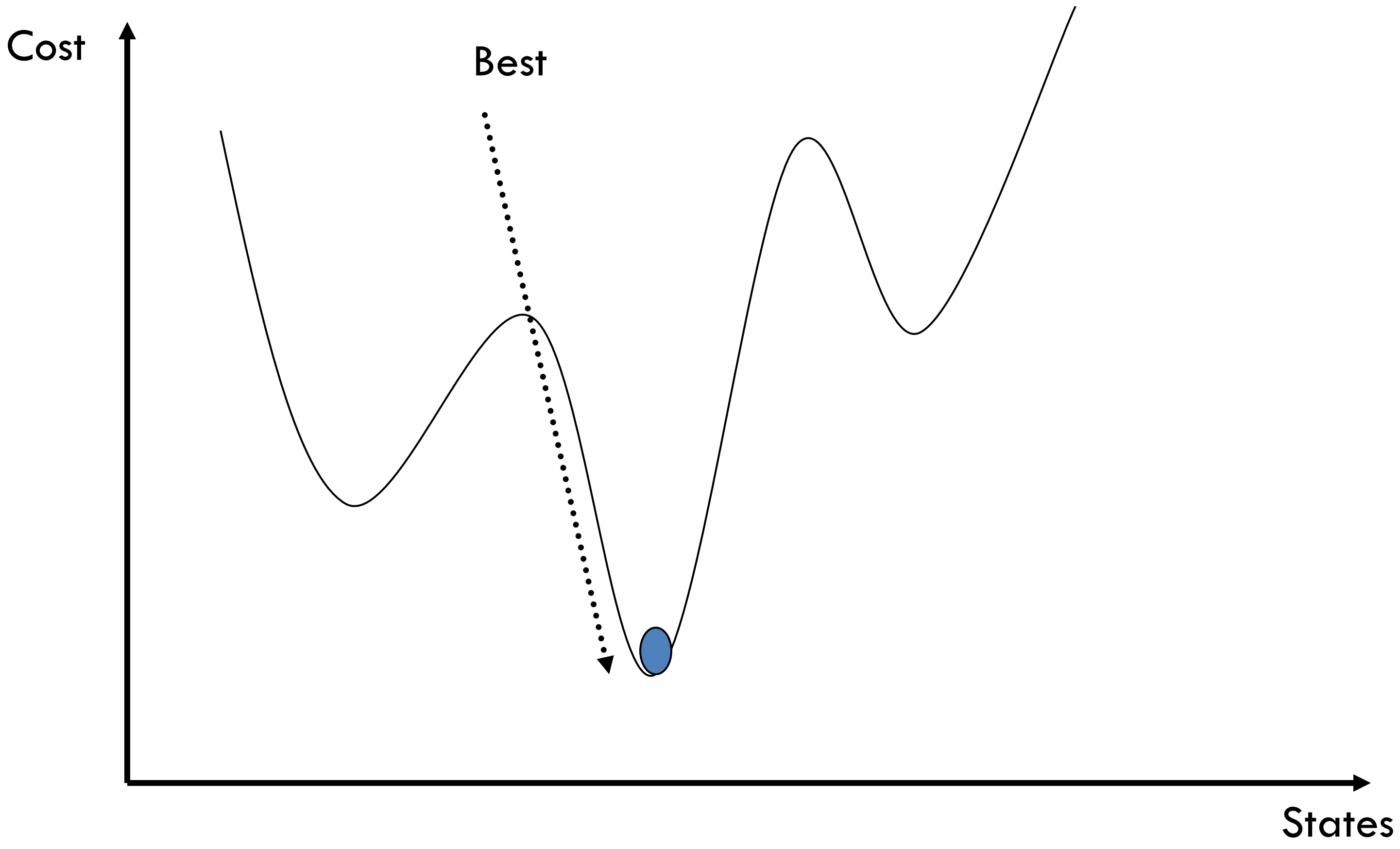
Simulated Annealing



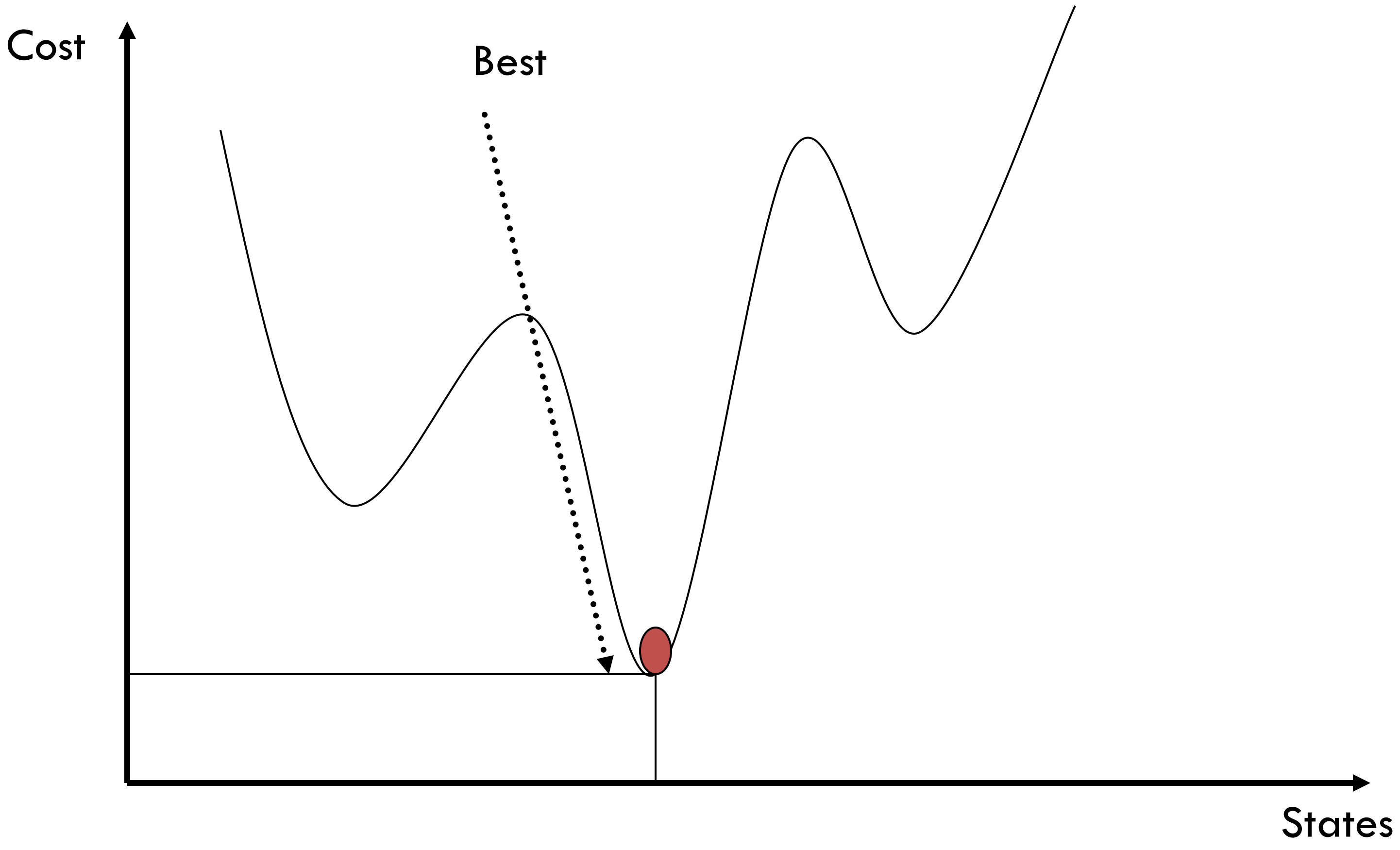
Simulated Annealing



Simulated Annealing



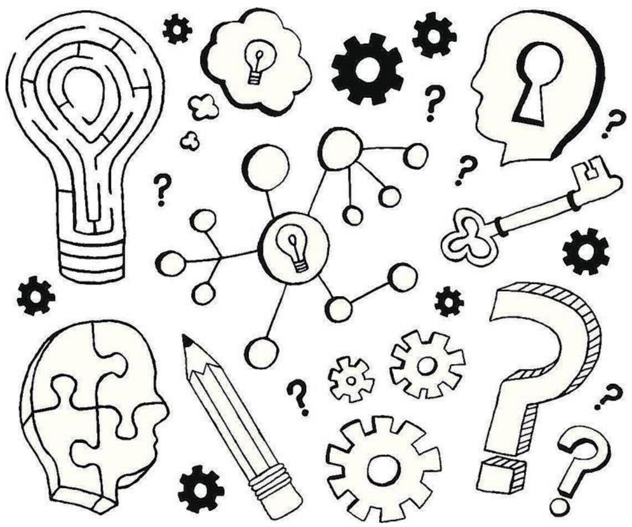
Simulated Annealing



Note:

Finding Global Minimum can only happen if we traverse the required iterations otherwise this method finds the local minimum only.

Activity (Reflection, 20’)



Reflect on what you have studied in **Simulated Annealing Search**.

Time for a break – 20’

Genetic Algorithm (Search)

Genetic Algorithms

- Formally introduced in the US in the 70s by John Holland.
- GAs emulate ideas from genetics and natural selection and can search potentially large spaces.
- Before we can apply Genetic Algorithm to a problem, we need to answer:

  - How is an individual represented?
  - What is the fitness function?
  - How are individuals selected?
  - How do individuals reproduce?

- Note:

What is explained here is one of the approaches to GA. GA is a wide area to study

Genetic Algorithms: Representation of states (solutions)

Each state or individual is represented as a string over a finite alphabet. It is also called **chromosome** which Contains **genes**.

genes

Solution: 607 $\xrightarrow{\text{Encoding}}$ 1001011111

Chromosome:

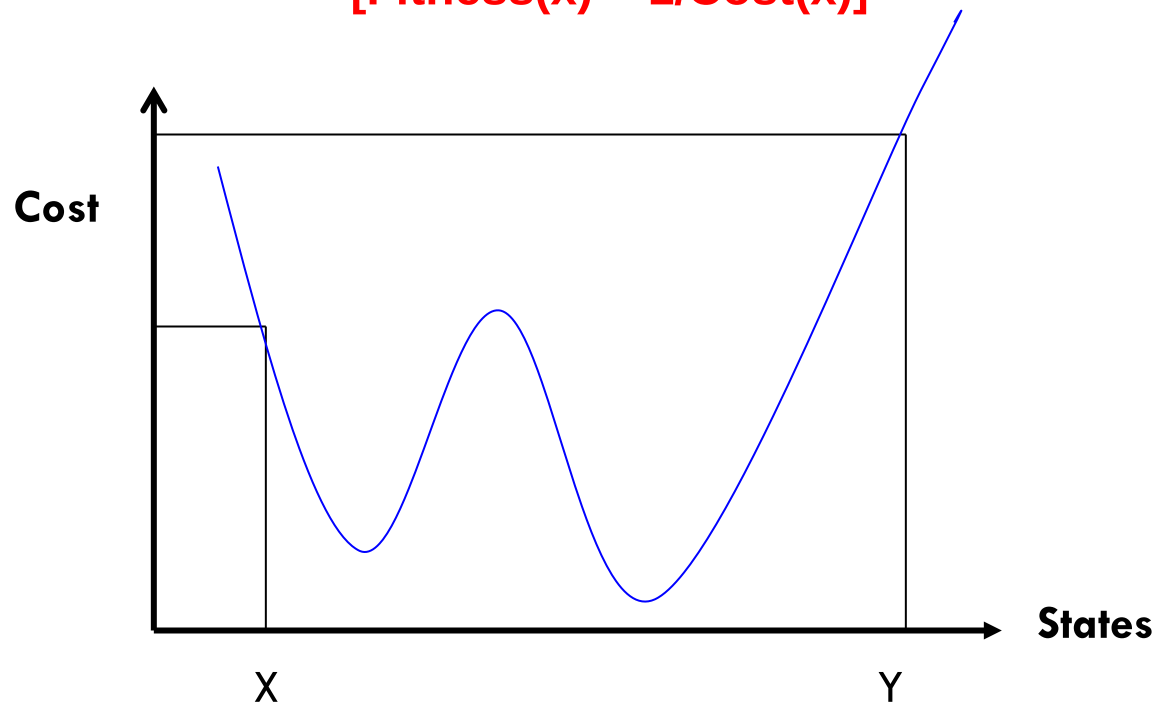
Binary String

Genetic Algorithms: Fitness Function

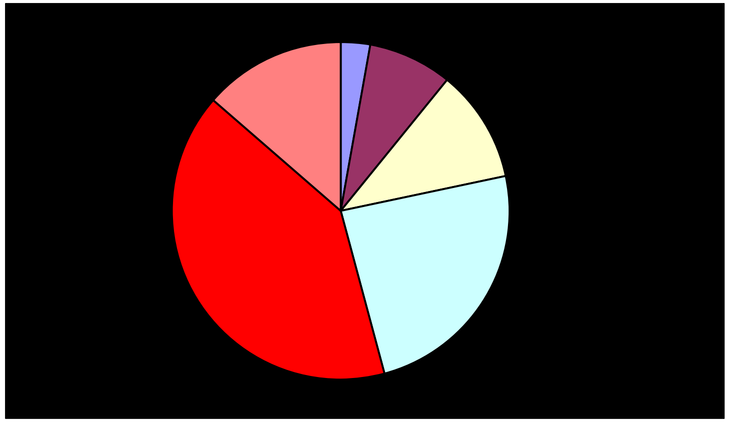
- Each state is rated by the evaluation function called **fitness function**. Fitness function should return higher values for better states:

Fitness(X) should be greater than Fitness(Y) !!

[Fitness(x) = 1/Cost(x)]



GA Parent Selection - Roulette Wheel



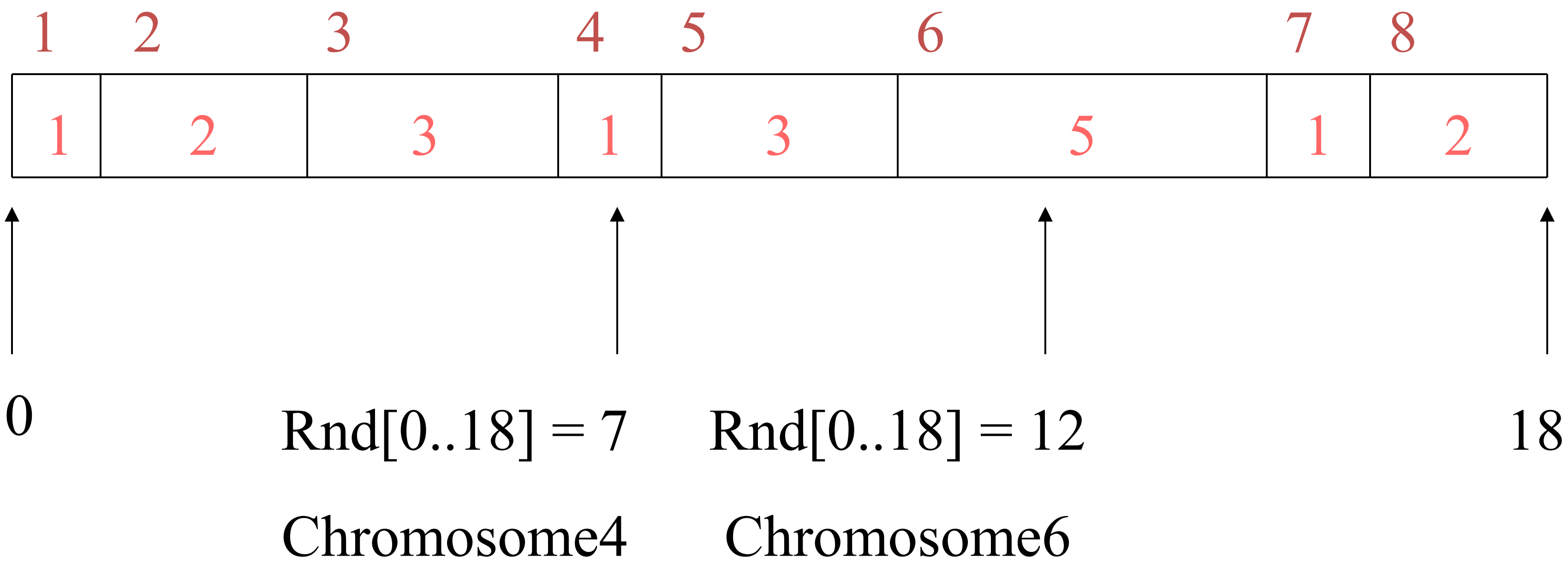
Roulette Wheel Selection

- Sum the fitnesses of all the population members, ***TF***
- Generate a random number, ***m***, between **0** and ***TF***
- Return the **first population member** whose fitness added to the preceding population members is greater than or equal to ***m***

Genetic Algorithms: Selection

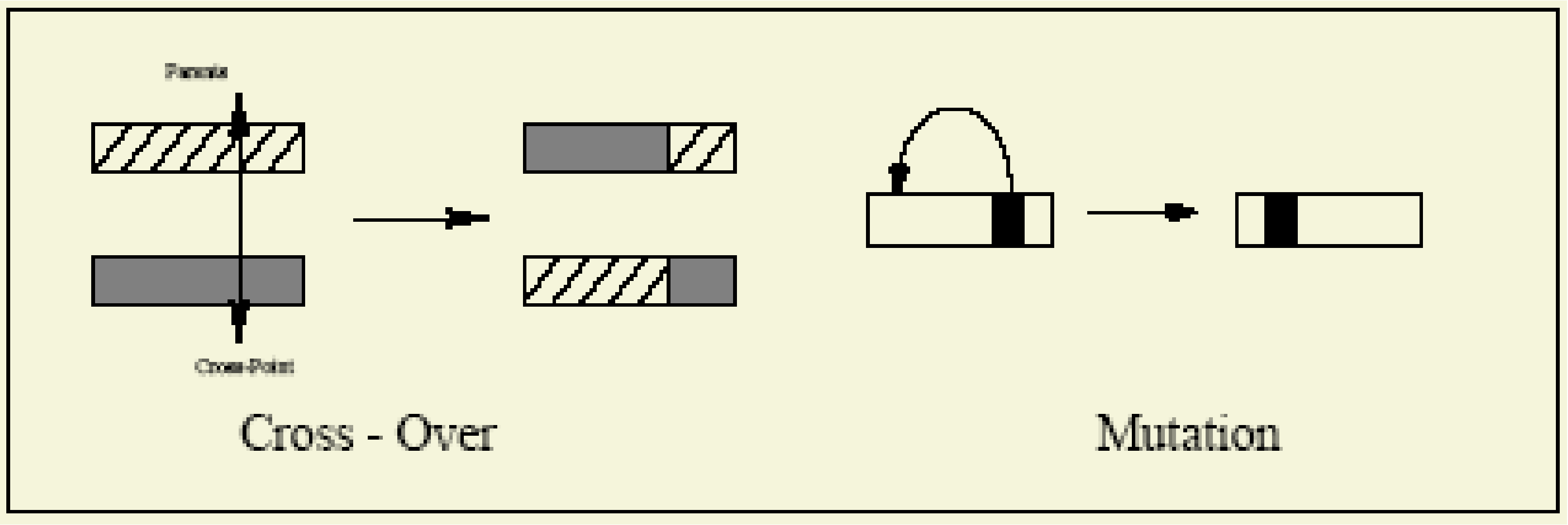
How are individuals selected ?

Roulette Wheel Selection



Genetic Algorithms: Cross-Over and Mutation

How do individuals reproduce ?



Genetic Algorithms
Crossover - Recombination

1010000000	Parent1	Offspring1	1011011111
1001011111	Parent2	Offspring2	1010000000

Crossover single point - random

With some high probability (*crossover rate*) apply crossover to the parents. (*typical values are 0.8 to 0.95*)

Stochastic Search: Genetic Algorithms Mutation

mutate

	Original offspring	Mutated offspring
Offspring1	1011011111	1011001111
Offspring2	1010000000	1000000000

Original offspring

Mutated offspring

With some small probability (the *mutation rate*) flip each bit in the offspring (*typical values between 0.1 and 0.001*)

Example

- If P_3 and P_2 are chosen as parents and we apply one point crossover show the resulting children, C_1 and C_2. Use a crossover point of 1 (first digit)
- Do the same using P_4 and P_2 with a crossover point of 2(two digit) and create C_3 and C_4
- Do multiple point crossover using parent P1 and P3 to give C5 and C6 on digts 1 and 4

Chromosome	Binary String
P_1	11100
P_2	01111
P_3	10111
P_4	00100

Chromosome	Binary String
C_1	11111
C_2	00111
C_3	00111
C_4	01100

C_5	10110
C_6	11101

Genetic Algorithms

Algorithm:

1. Initialize population with **p** Individuals at random
2. For each Individual **h** compute its fitness
3. While max fitness < threshold do

   Create a new generation Ps
4. Return the Individual with highest fitness

Activity (Reflection, 20’)

Reflect on what you have studied in **GA Search.**