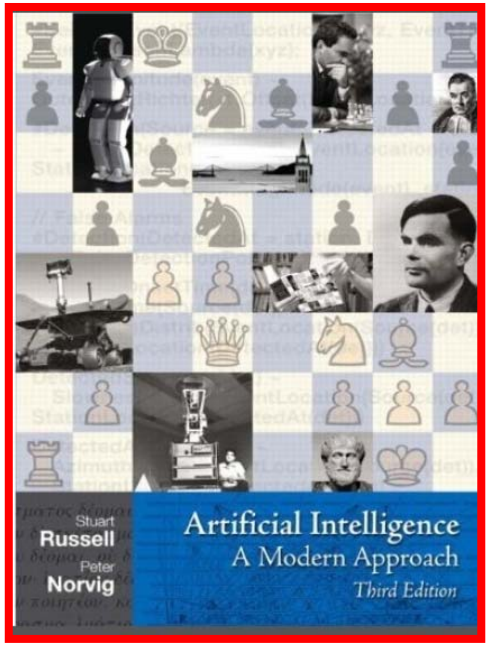


(Chapter-4-1)
BEYOND CLASSICAL SEARCH

Yanmei Zheng

LOCAL SEARCH STRATEGY

- **Hill-Climbing Search.**
- **Simulated Annealing Search.**
- **Local Beam Search.**
- **Genetic Algorithms.**

Classical search versus Local Search

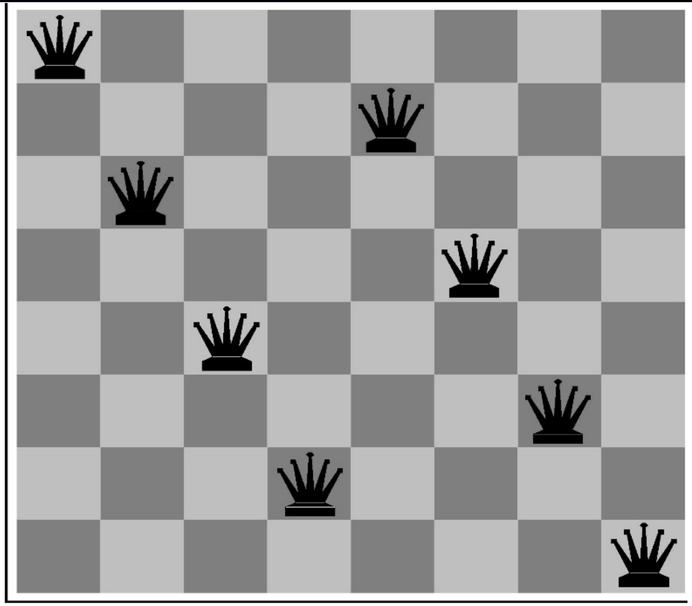
Classical search	Local Search
➢ systematic exploration of search space. ➢ Keeps one or more paths in memory. ➢ Records which alternatives have been explored at each point along the path. ➢ The **path** to the goal is a **solution** to the problem.	➢ In many optimization problems, the path to the goal is **irrelevant**; **the goal state itself is the solution**. ➢ State space = set of "complete" configurations. ➢ Find configuration satisfying constraints, Find best state according to some objective function h(s). e.g., **n-queens**, h(s)= number of attacking queens. **In such cases, we can use Local Search Algorithms.**

Local Search Algorithms

- **Local Search Algorithms keep a single "current" state, and move to neighboring states in order to try improve it.**
- **Solution path needs not be maintained.**
- **Hence, the search is "local".**
- **Local search suitable for problems in which path is not important; the goal state itself is the solution.**
- **It is an optimization search**

Example: n-queens

- **Put n queens on an n × n board with no two queens on the same row, column, or diagonal.**



- **In the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.**

Local Search: Key Idea

Key idea:

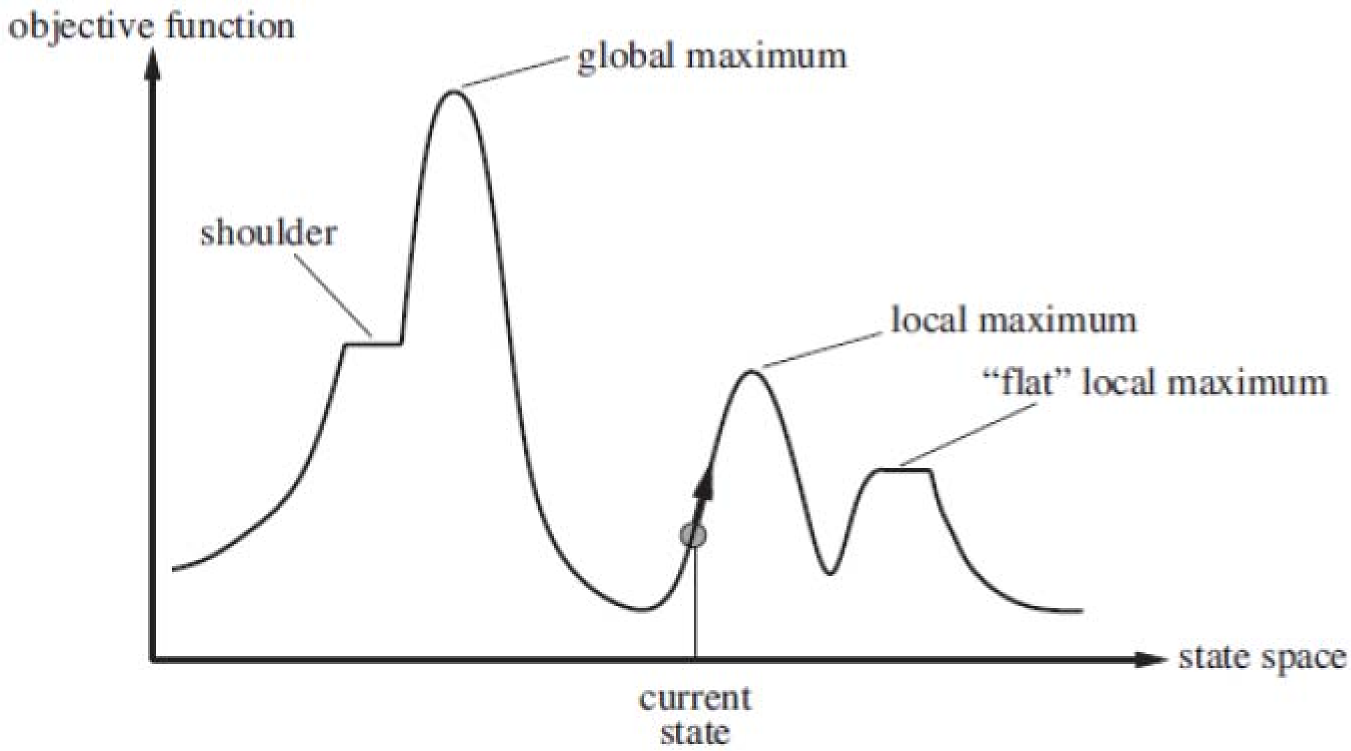
1. **Select (random) initial state (generate an initial guess).**
2. **Make local modification to improve current state (evaluate current state and move to other states).**
3. **Repeat Step 2 until goal state found (or out of time).**

Local Search: Key Idea

Advantages	Drawback:
➢ Use very little memory – usually a constant amount. ➢ Can often find reasonable solutions in large or infinite state spaces (e.g., continuous). For which systematic search is unsuitable.	➢ Local Search can get stuck in local maxima and not find the optimal solution.

State Space Landscape

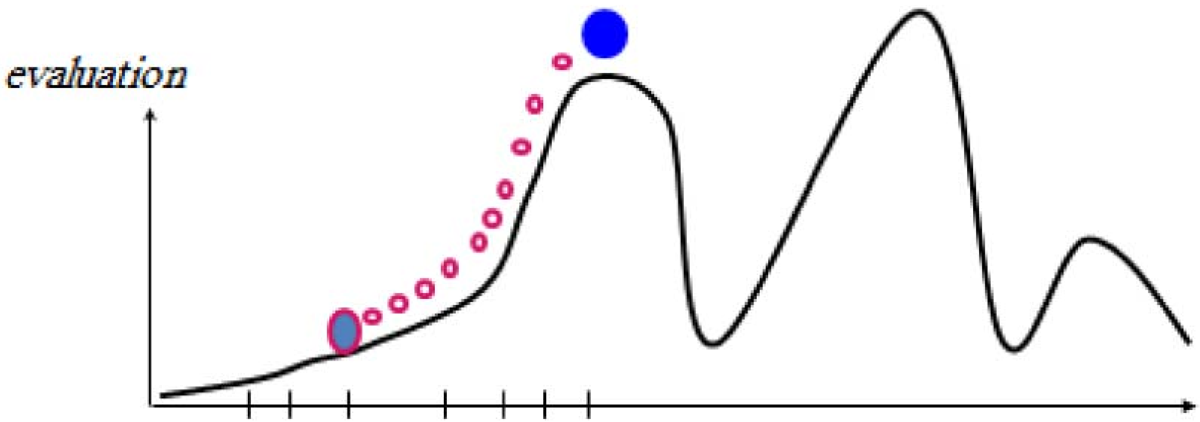
- **A state space landscape: is a graph of states associated with their costs.**

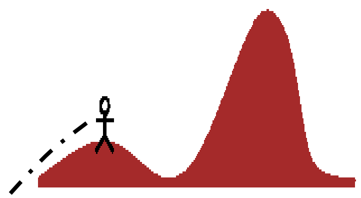


Hill-Climbing Search

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.



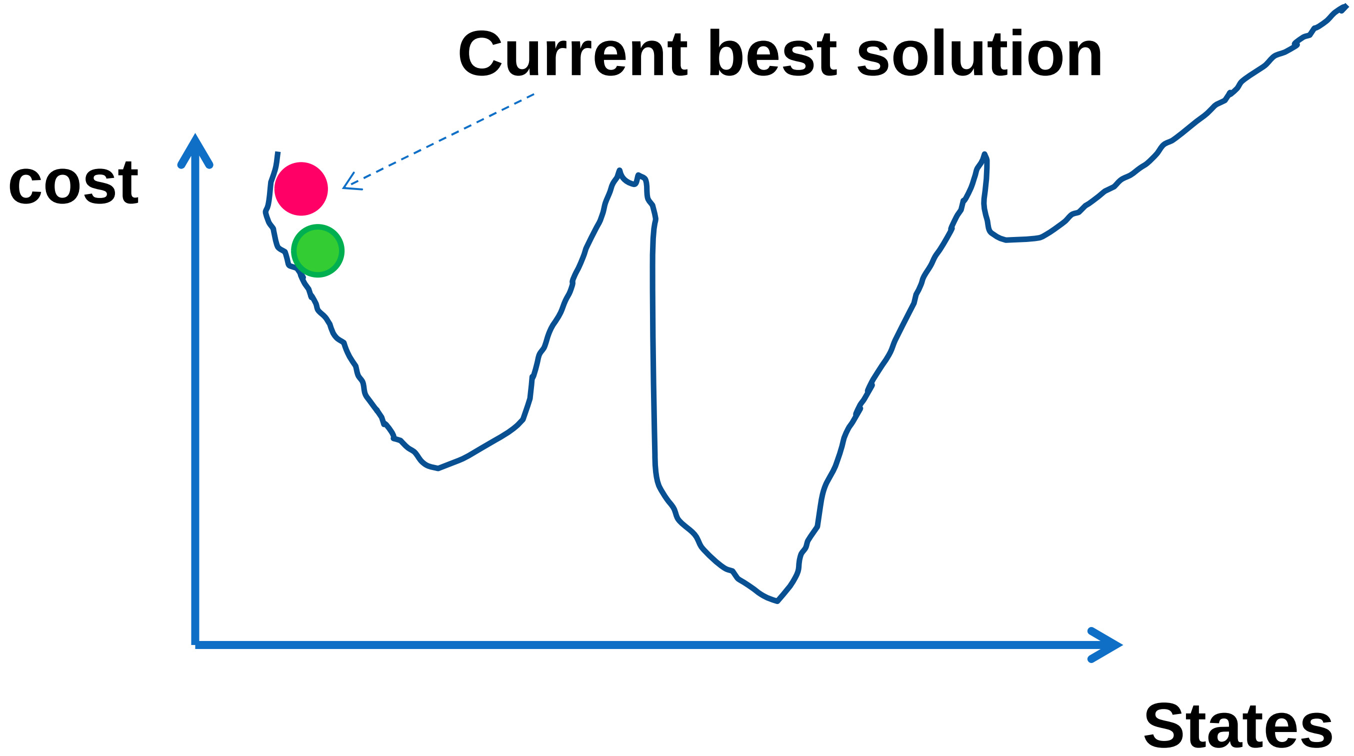


Hill-Climbing Search

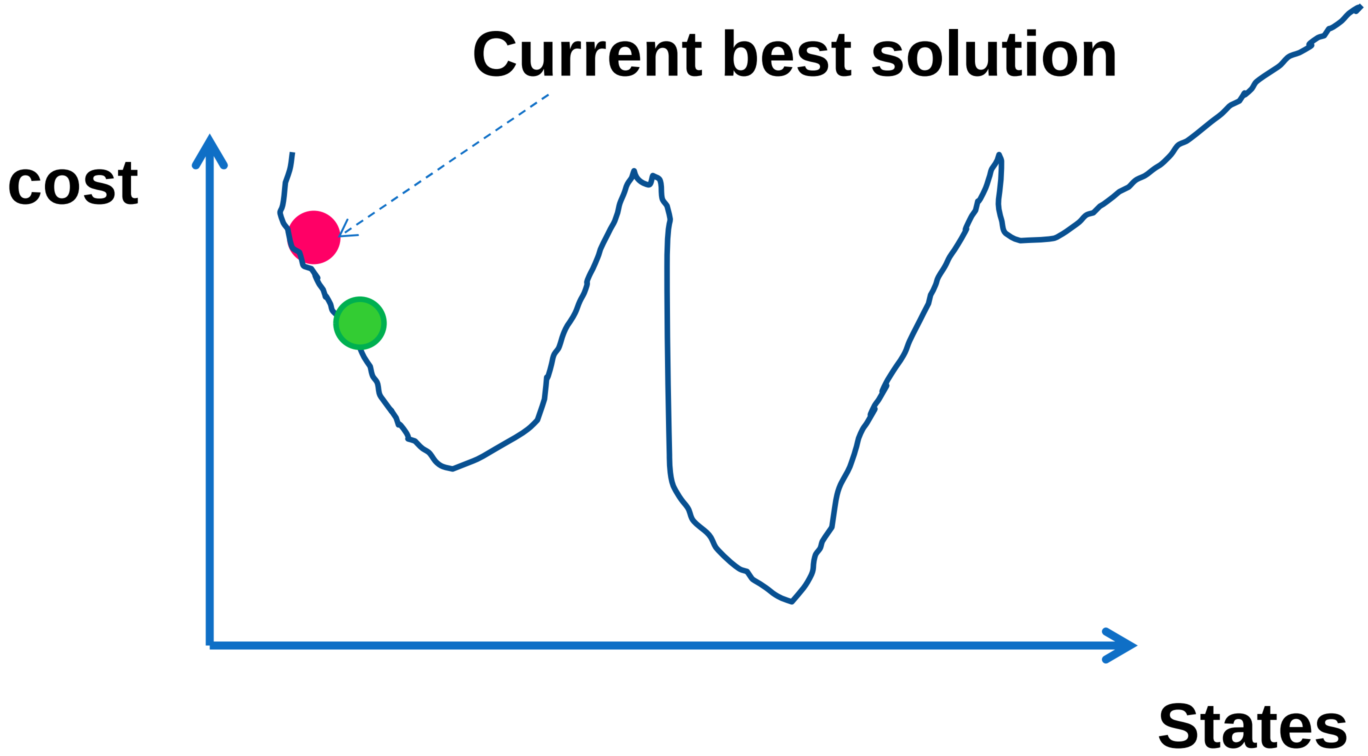
- "Like climbing Everest in thick fog with amnesia"

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*

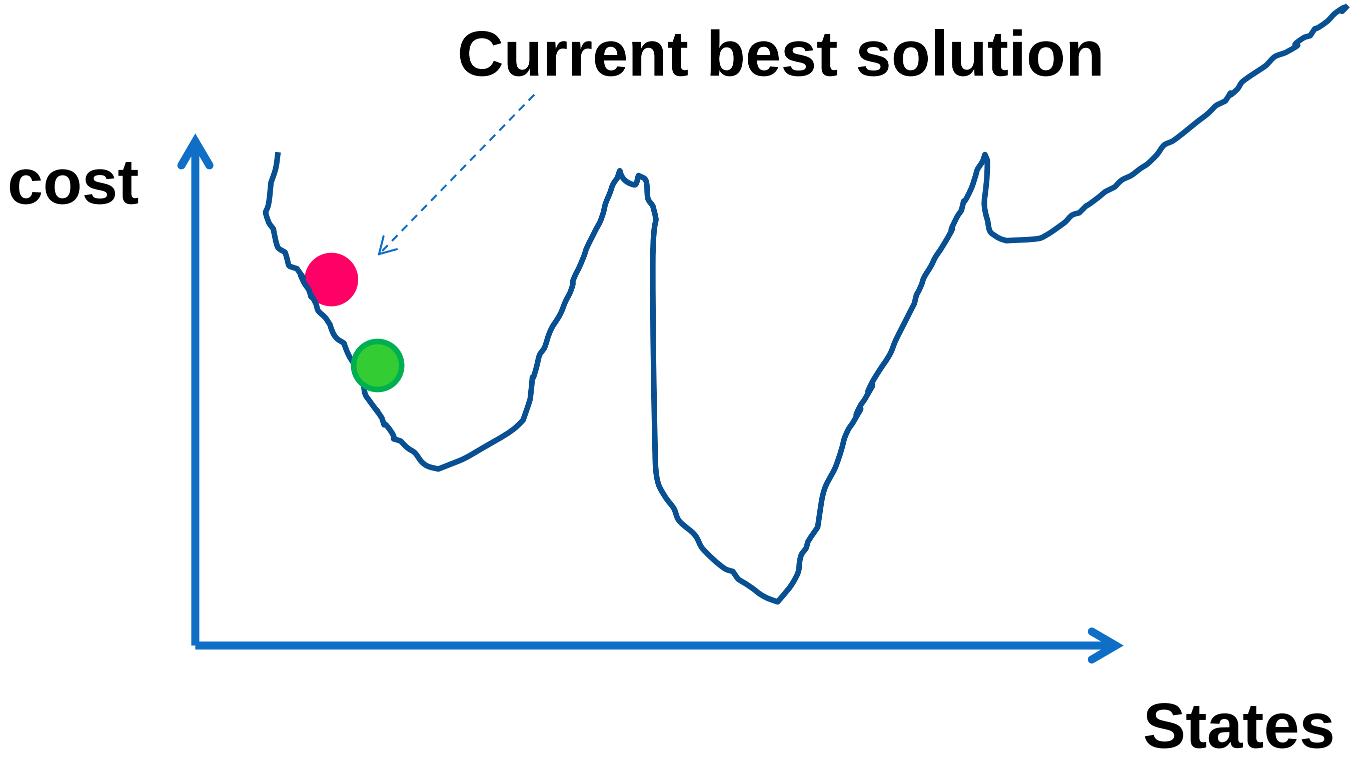
Hill-Climbing in Action …



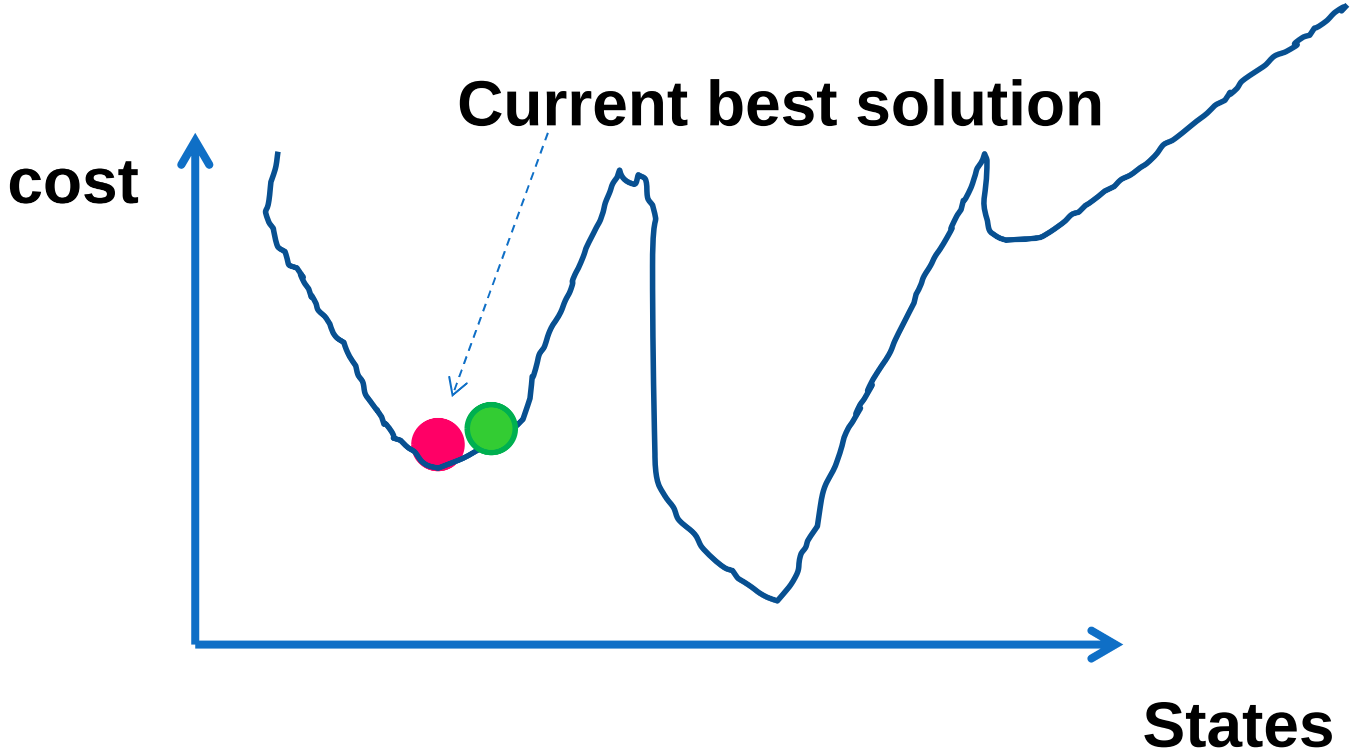
Hill-Climbing in Action …



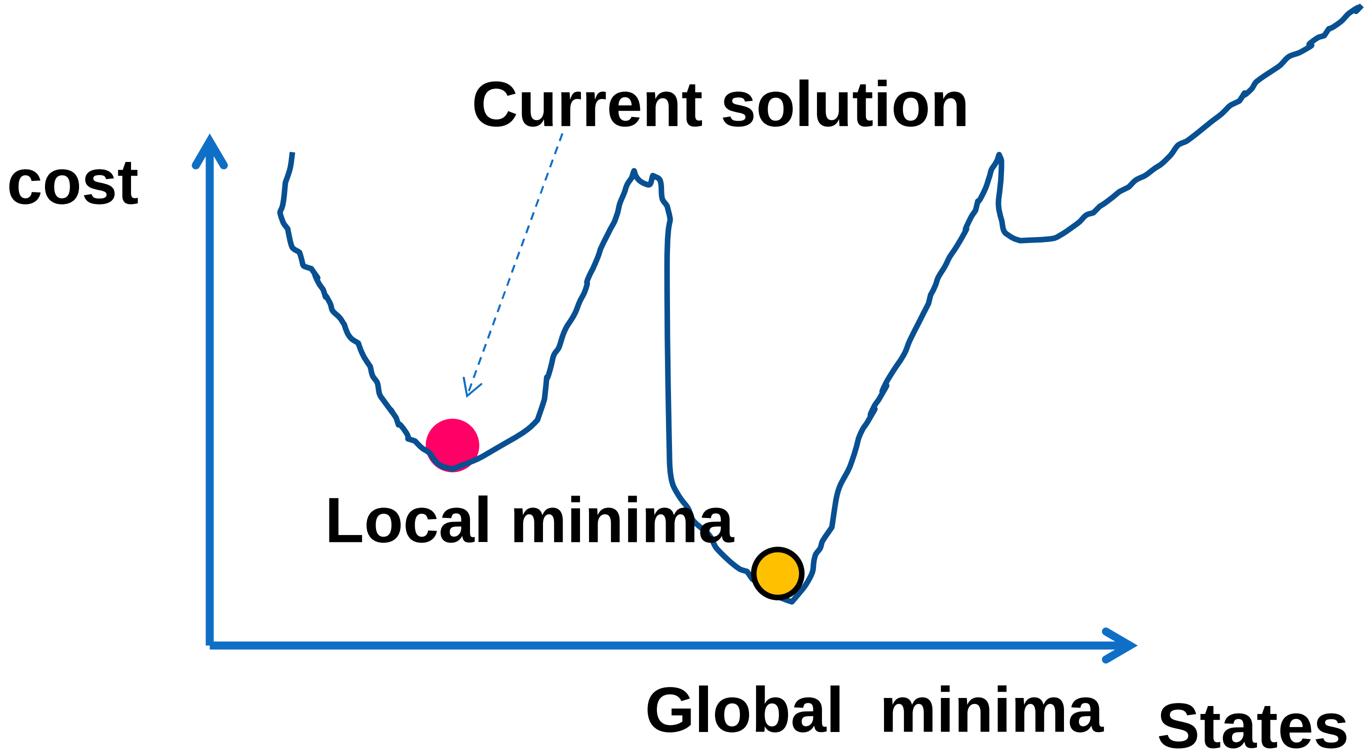
Hill-Climbing in Action …



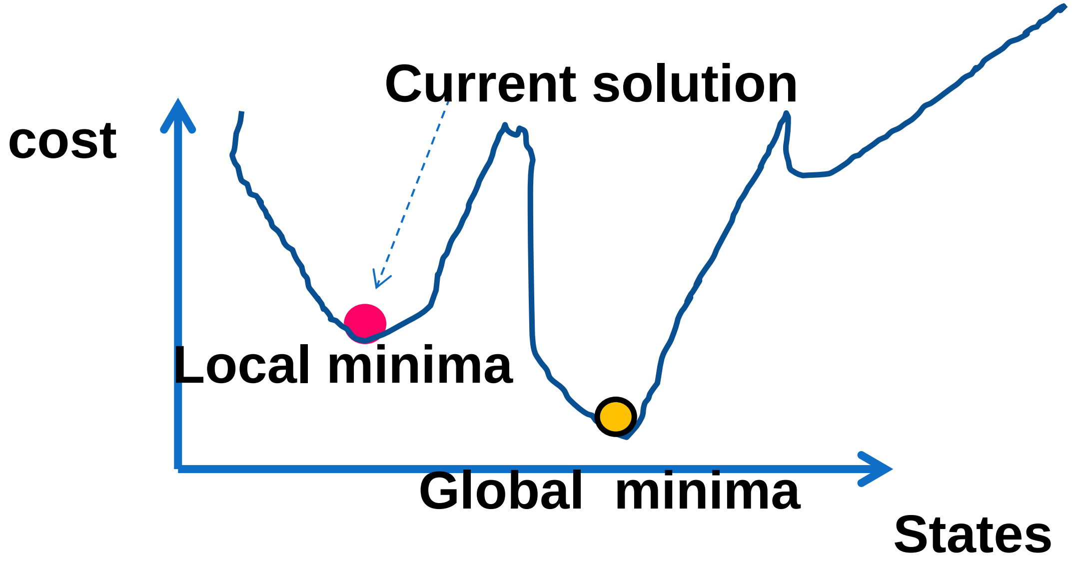
Hill-Climbing in Action …



Hill-Climbing in Action …



Hill-Climbing in Action …

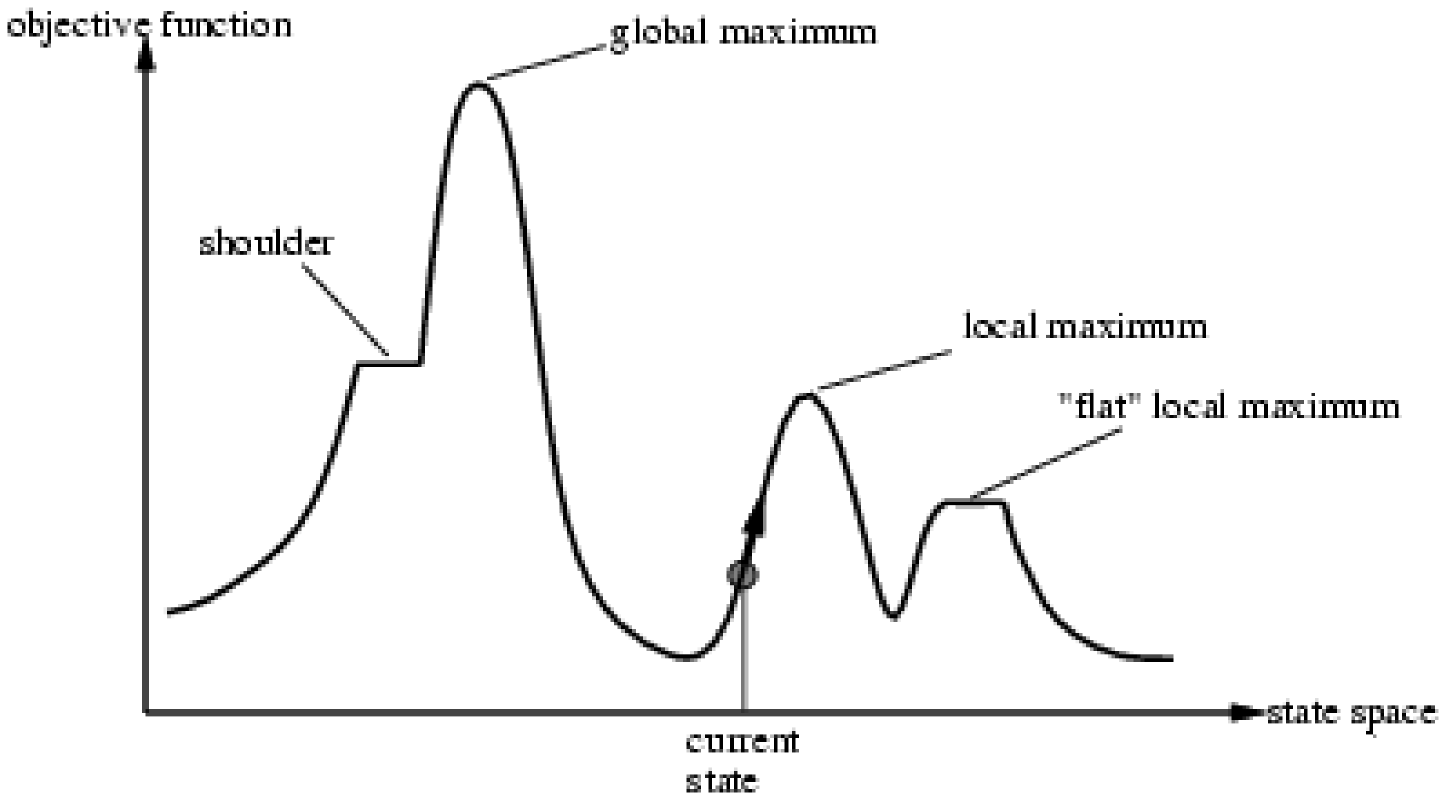


Drawback: Depending on initial state, it can get stuck in local maxima/minimum or flat local maximum and not find the solution.

Cure: Random restart.

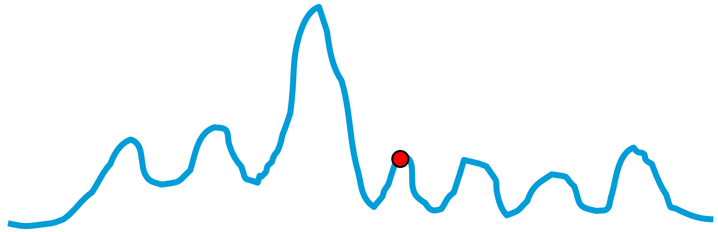
Hill-climbing search

- **Problem: depending on initial state, can get stuck in local maxima**

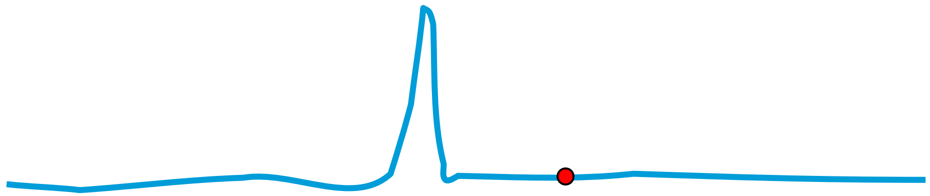


Hill Climbing Problems

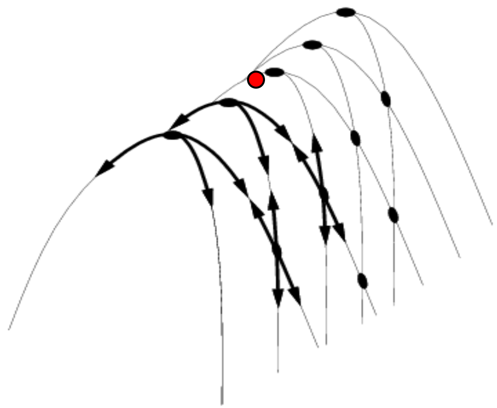
Local maxima



Plateaus



Diagonal ridges



What is it sensitive to?
Does it have any advantages?

Solving the Problems

- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part of the search space

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?

Simulated Annealing

- Variant of hill climbing (so up is good)
- Tries to explore enough of the search space early on, so that the final solution is less sensitive to the start state
- May make some downhill moves before finding a good way to move uphill.

Annealing

- One manner in which crystals are formed

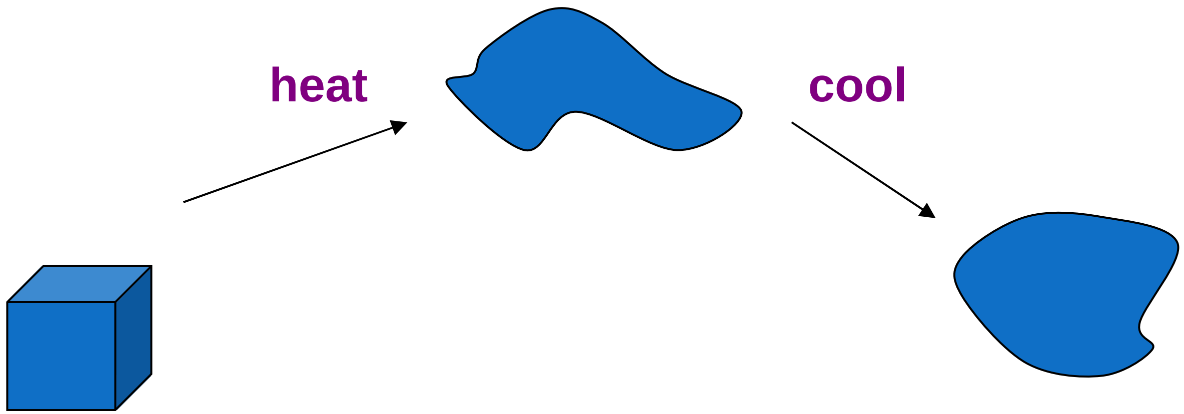
- Gradual cooling of liquid …

  - ✓ At high temperatures, molecules move freely
  - ✓ At low temperatures, molecules are "stuck"
  - ✓ If cooling is slow

    Low energy, organized crystal lattice formed

Simulated Annealing

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is

  p = e^(-ΔE/kT)

  where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann's constant.

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

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - ΔE/kT gets bigger
  - (-ΔE/kT) gets smaller
  - e^(-ΔE/kT) gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

For Simulated Annealing

- ΔE represents the change in the value of the objective function.

- Since the physical relationships no longer apply, drop k. So p = $e^{(-\Delta E/T)}$

- We need an annealing schedule, which is a sequence of values of T: $T_0, T_1, T_2, \ldots$

The relationship between simulated annealing algorithm and physical annealing

Simulated annealing algorithm	Physical annealing
Solution	State of the particle
The optimal solution	Lowest energy state
Set the initial temperature	Melting process
Metropolis sampling process	Isothermal process
The control parameter T goes down	Cooling
The objective function	Energy

Simulated annealing Search

```
function SIMULATED-ANNEALING(problem, schedule) ret urns 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∞ to
    T←schedule[t]
    if T= 0 then return current
    next←a randomly selected successor of current
    ΔE←VALUE[next]-VALUE[current]
    if ΔE>0 then current←next
    else current←next only with probability e^(ΔE/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

Dr.Yanmei Zheng

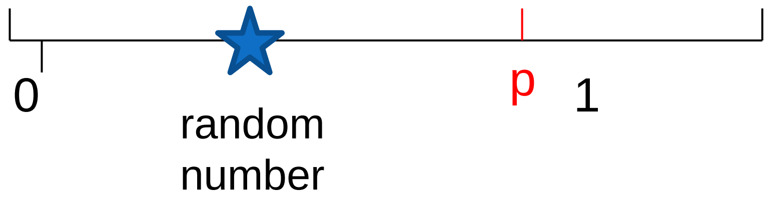
Simulated Annealing Algorithm

- *current* ← start node;
- for each T on the schedule /* need a schedule */
  - *next* ← randomly selected successor of *current*
  - evaluate next; it it's a goal, return it
  - ΔE ← *next*.Value – *current*.Value /* already negated */
  - if ΔE > 0
    - then *current* ← *next* /* better than current */
    - else *current* ← *next* with probability e^(ΔE/T)

How would you do this probabilistic selection?

Probabilistic Selection

- **Select *next* with probability p**



- **Generate a random number**
- **If it's <= p, select *next***

Simulated Annealing Properties

- At a fixed "temperature" T, state occupation probability reaches the Boltzman distribution

$$p(x) = \alpha e\hat{}(E(x)/kT)$$

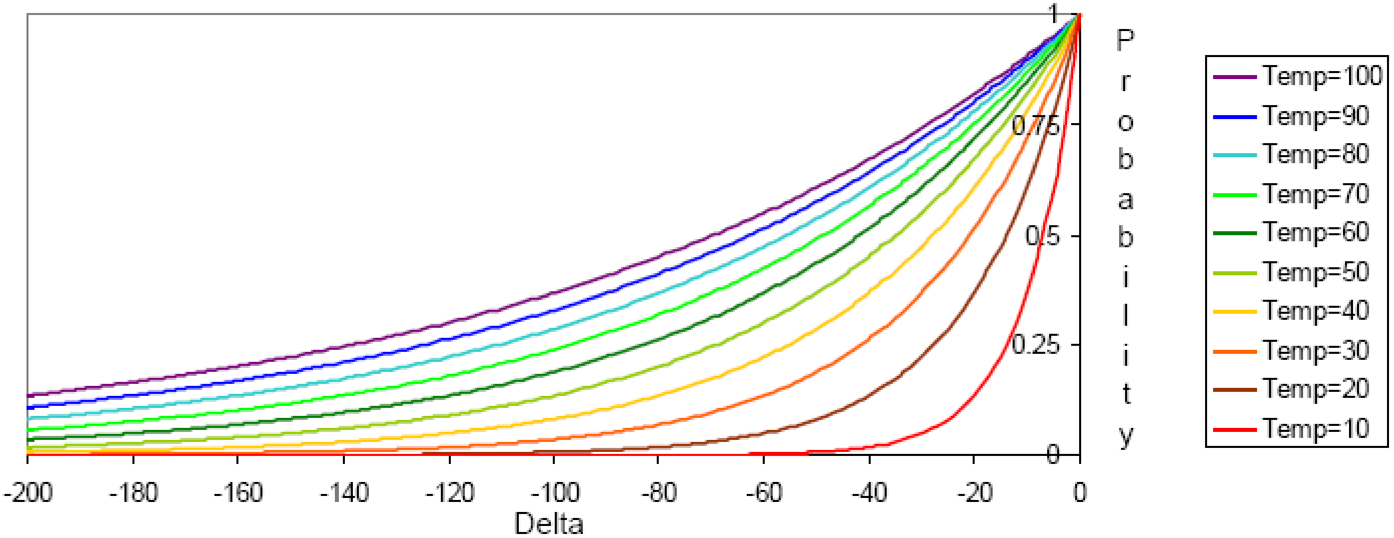
- If T is decreased slowly enough (very slowly), the procedure will reach the best state.
- Slowly enough has proven too slow for some researchers who have developed alternate schedules.

Simulated Annealing Schedules

- **Acceptance criterion and cooling schedule**

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */

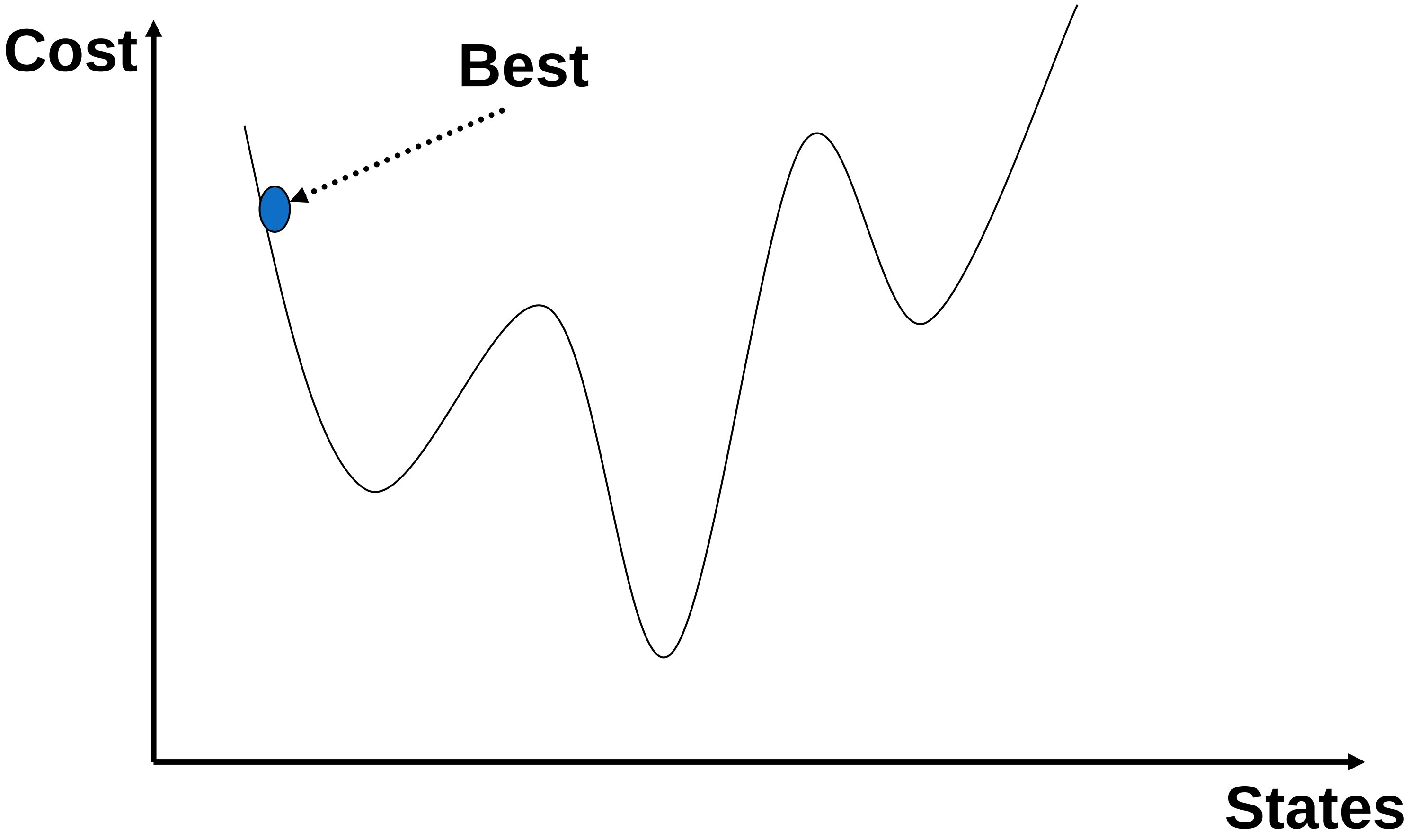


Initially temperature is very high (most bad moves accepted)

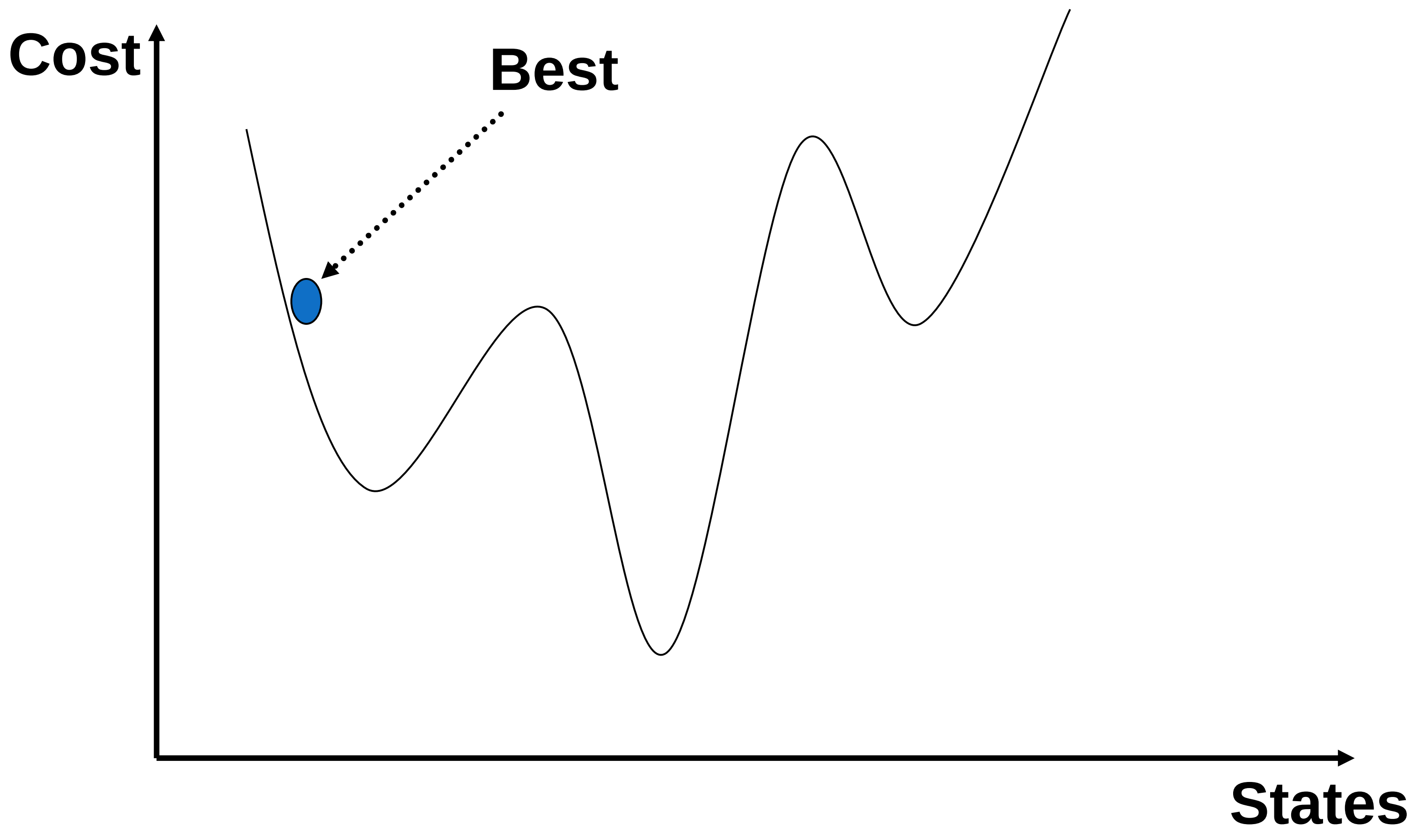
Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

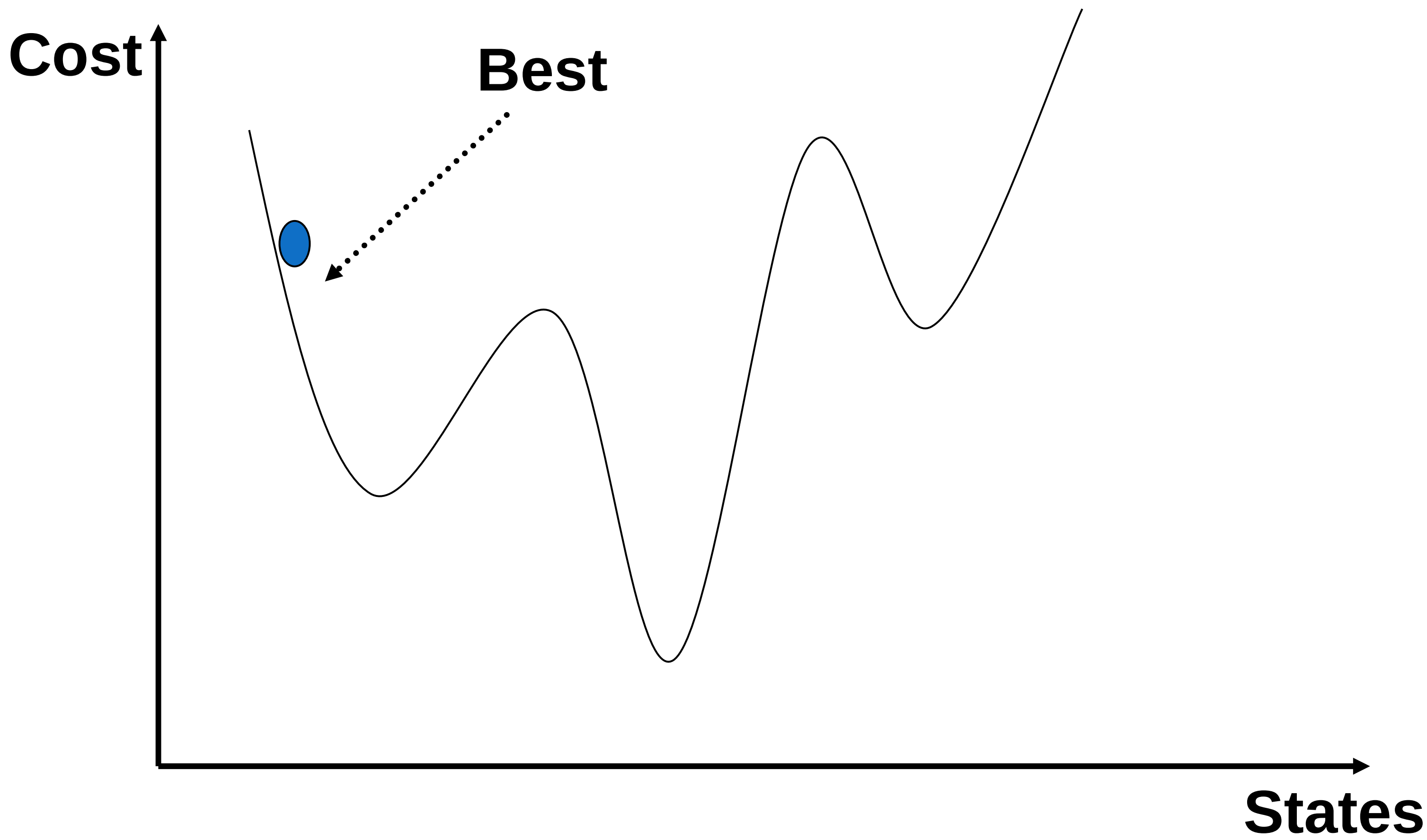
Simulated Annealing



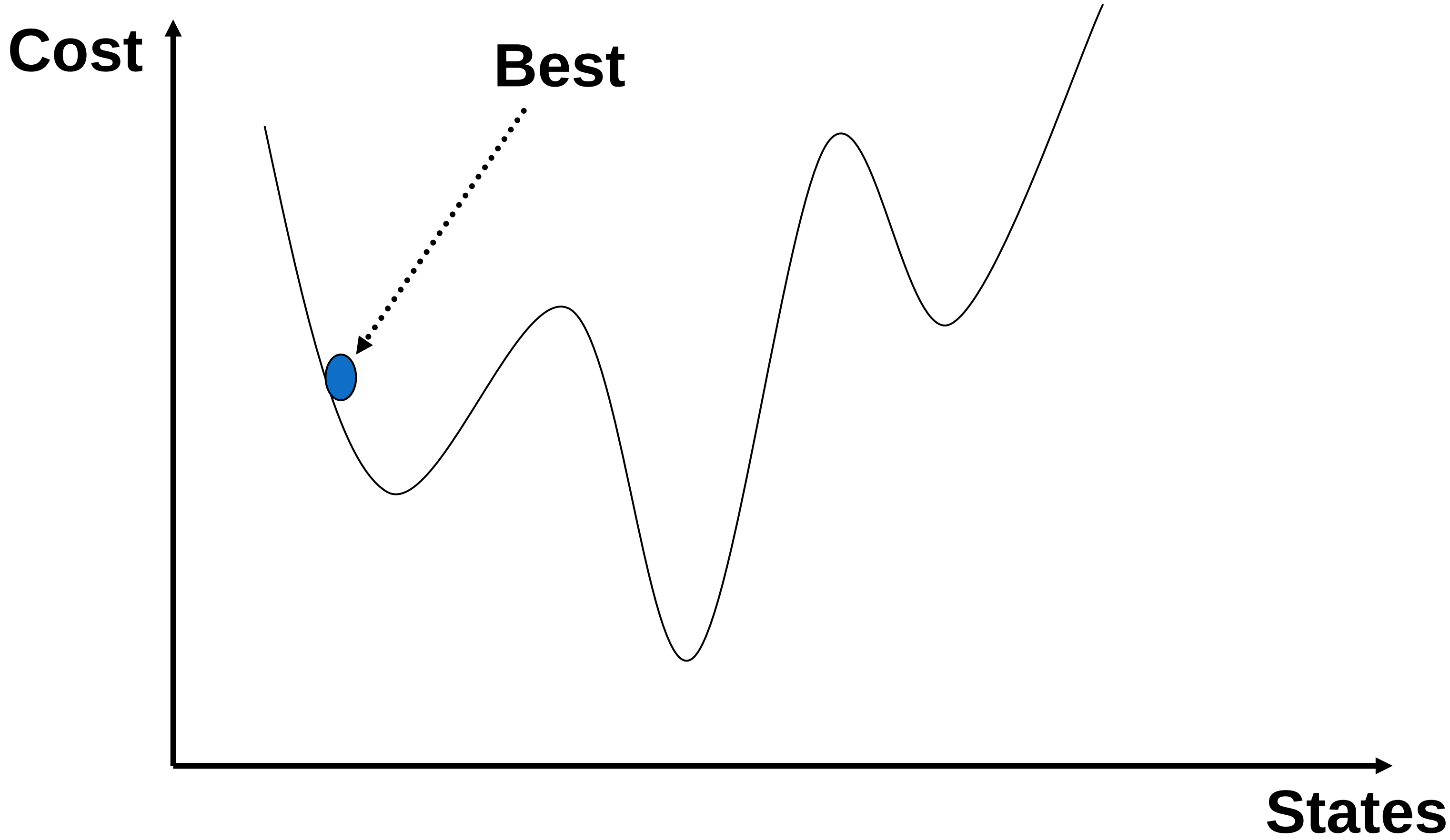
Simulated Annealing



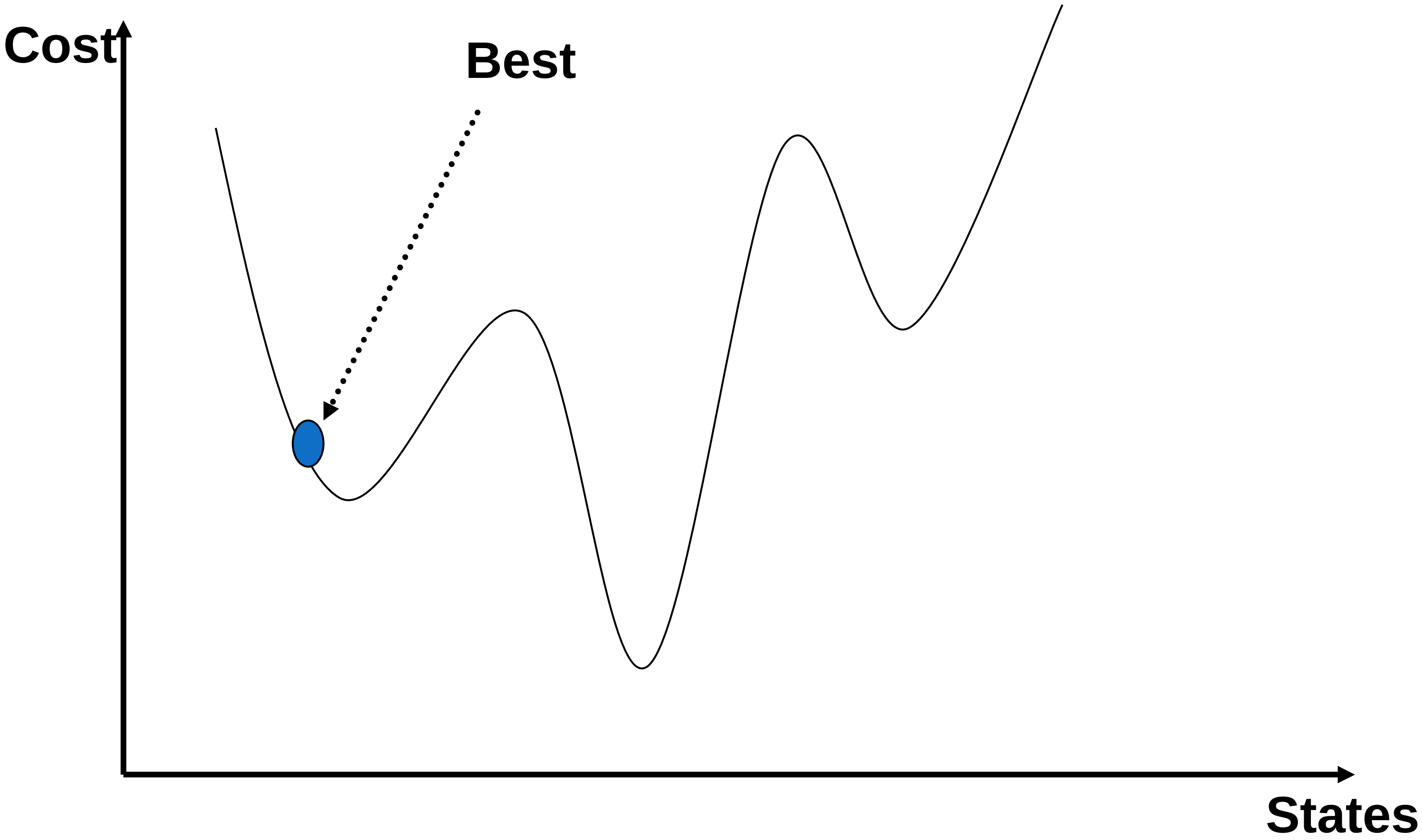
Simulated Annealing



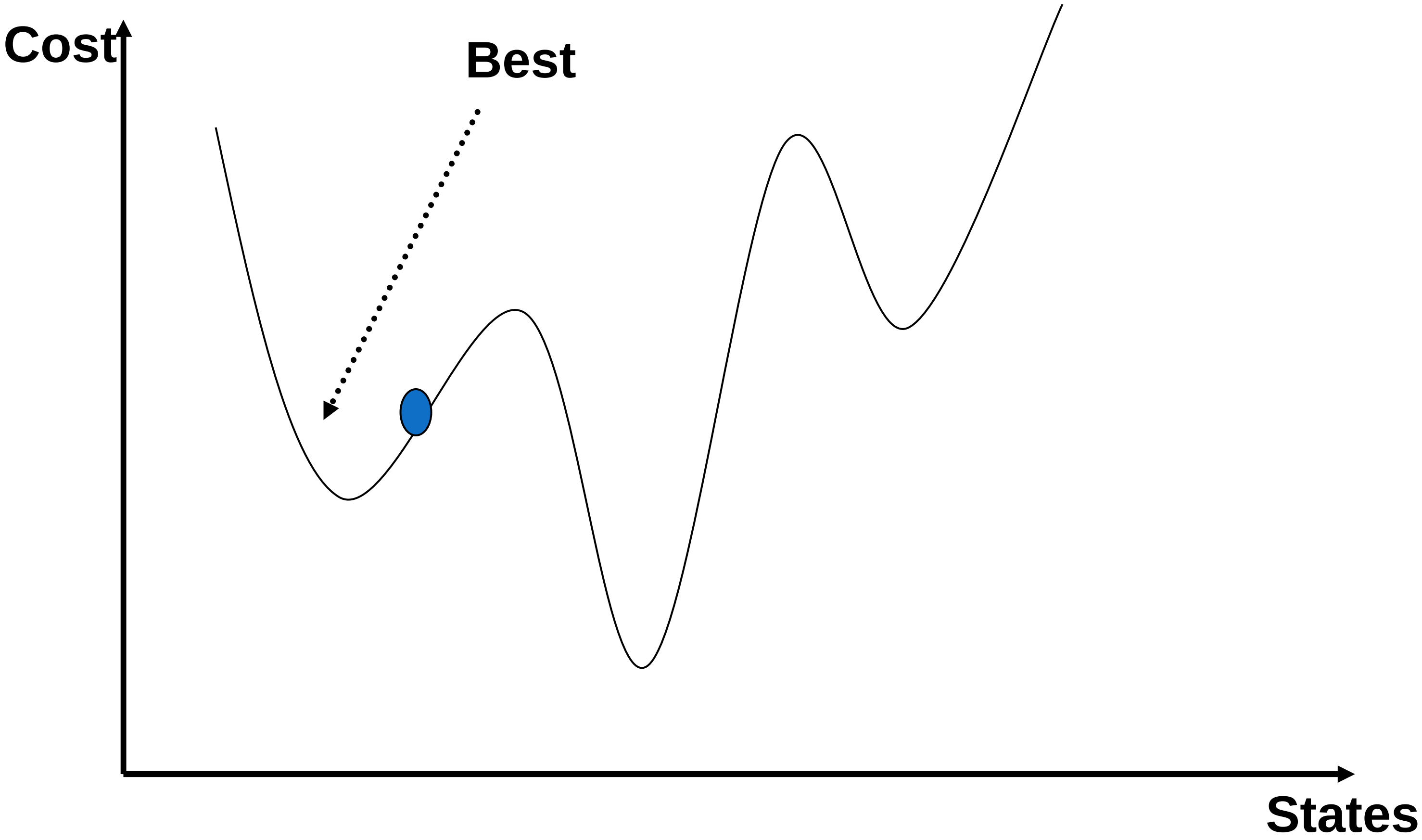
Simulated Annealing



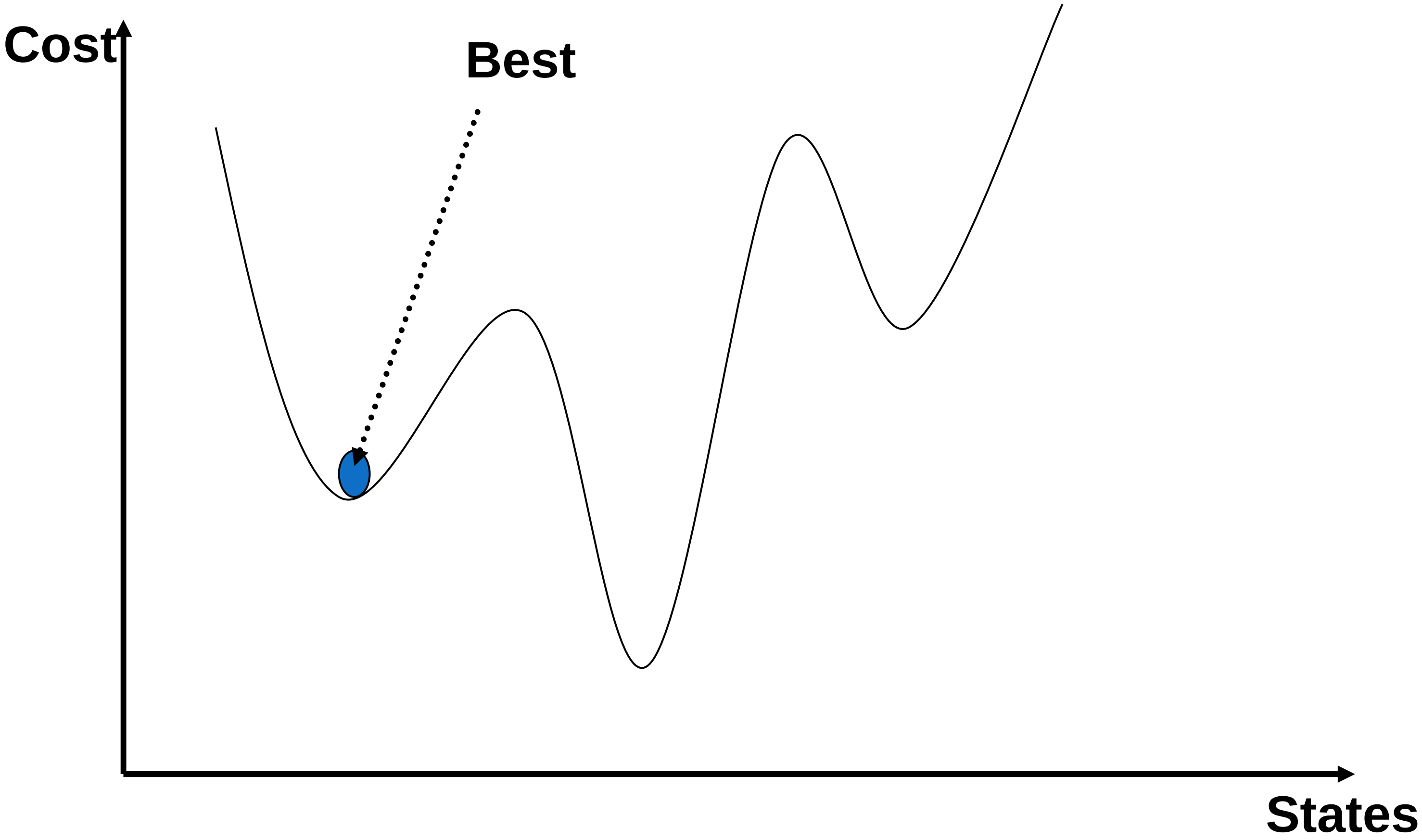
Simulated Annealing



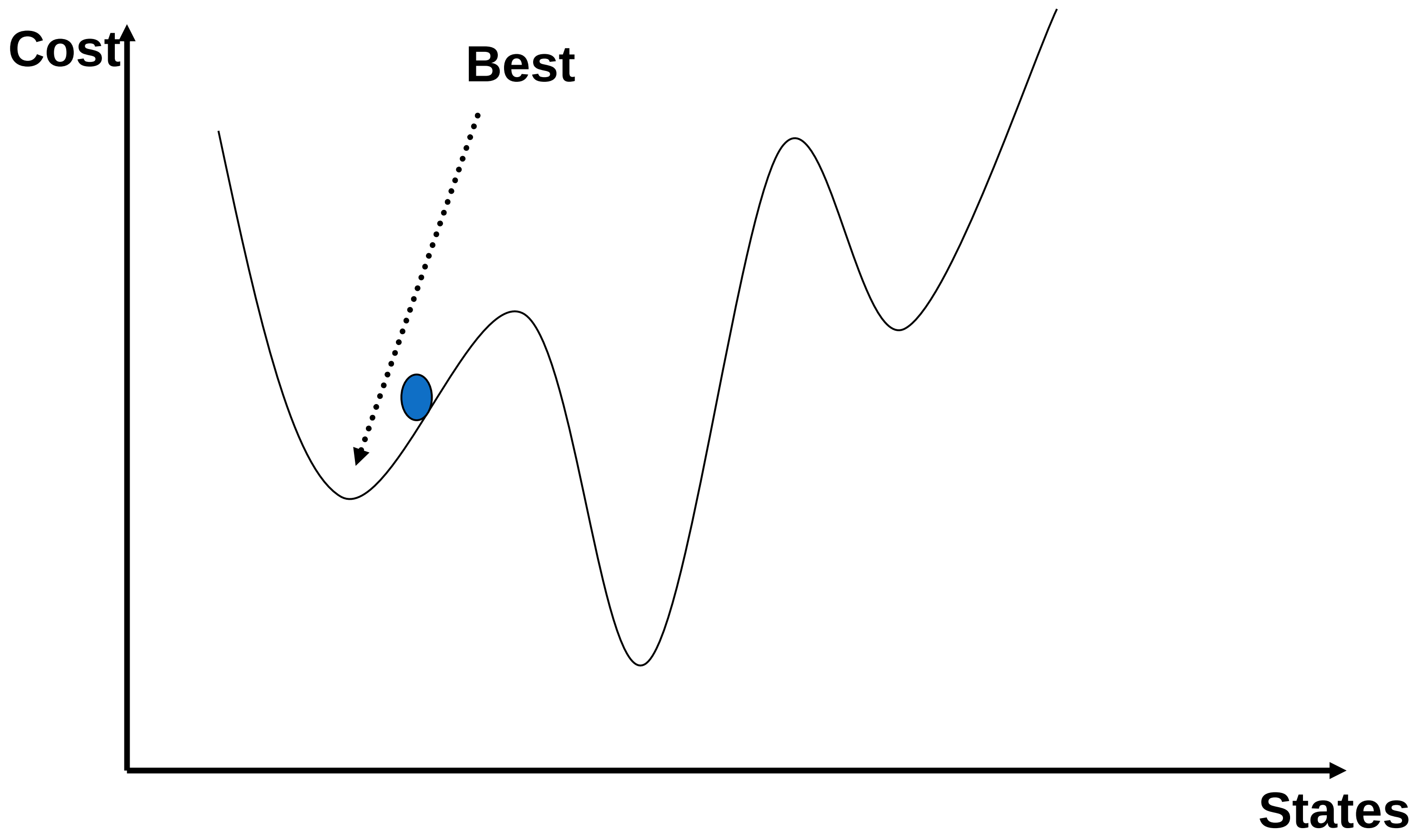
Simulated Annealing



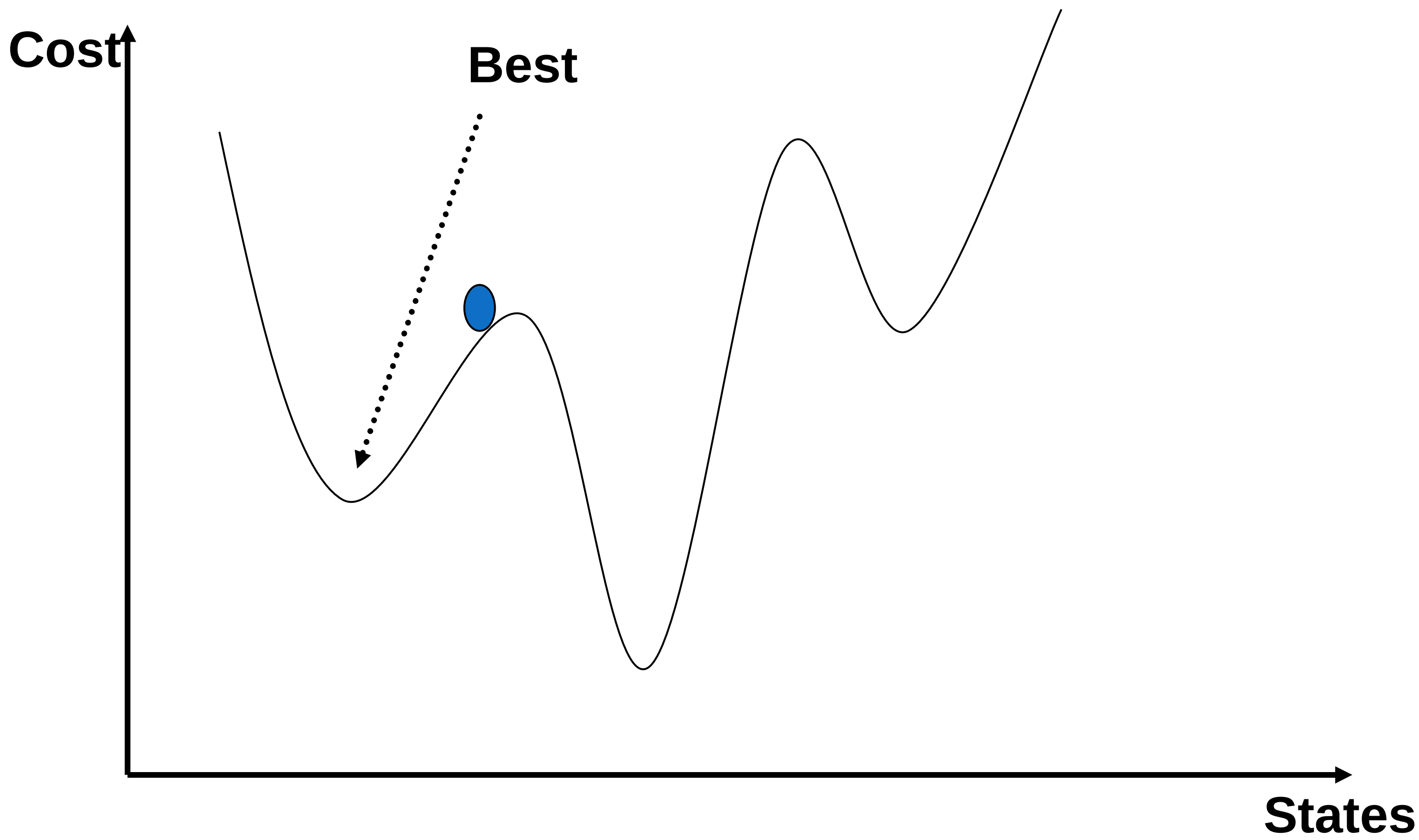
Simulated Annealing



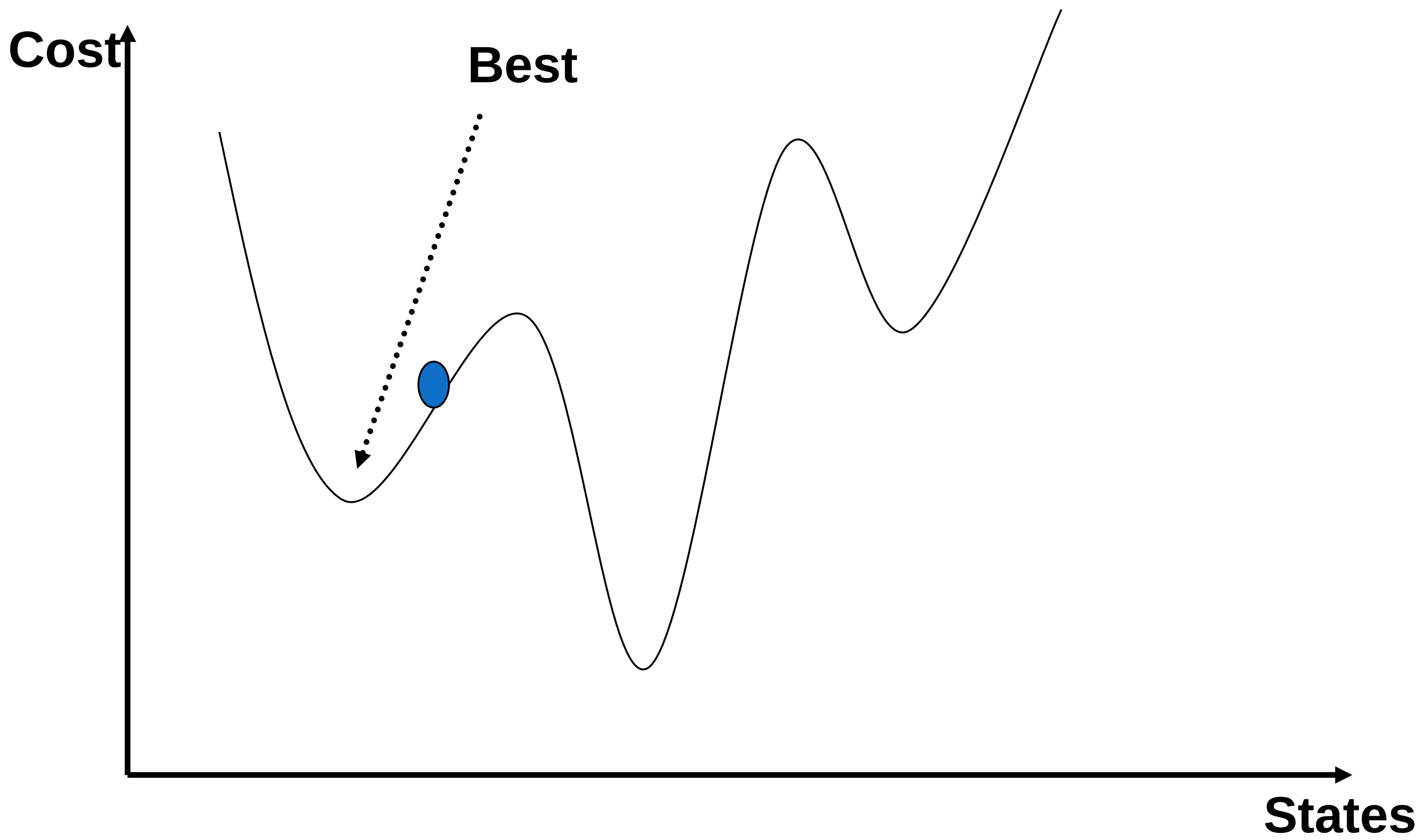
Simulated Annealing



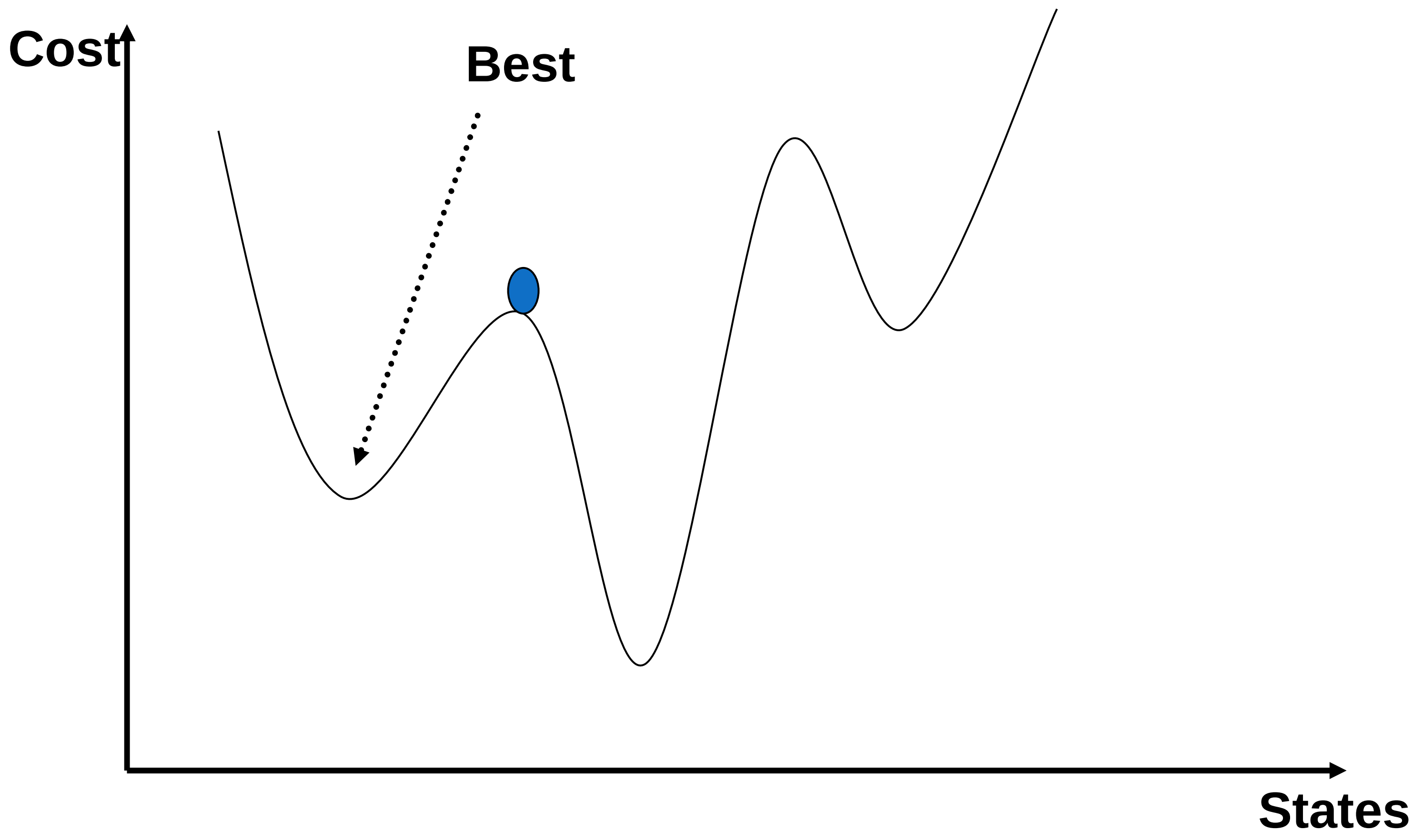
Simulated Annealing



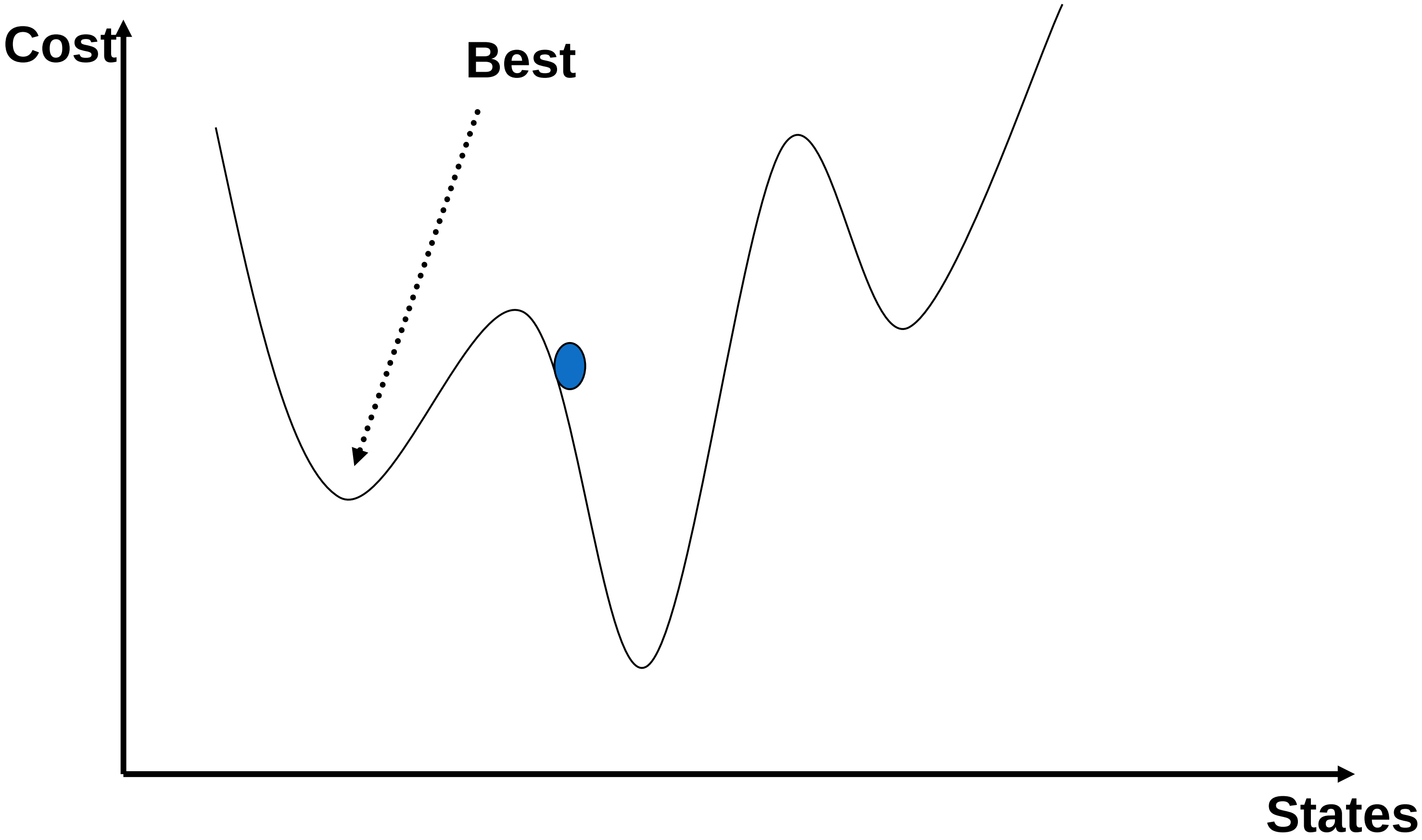
Simulated Annealing



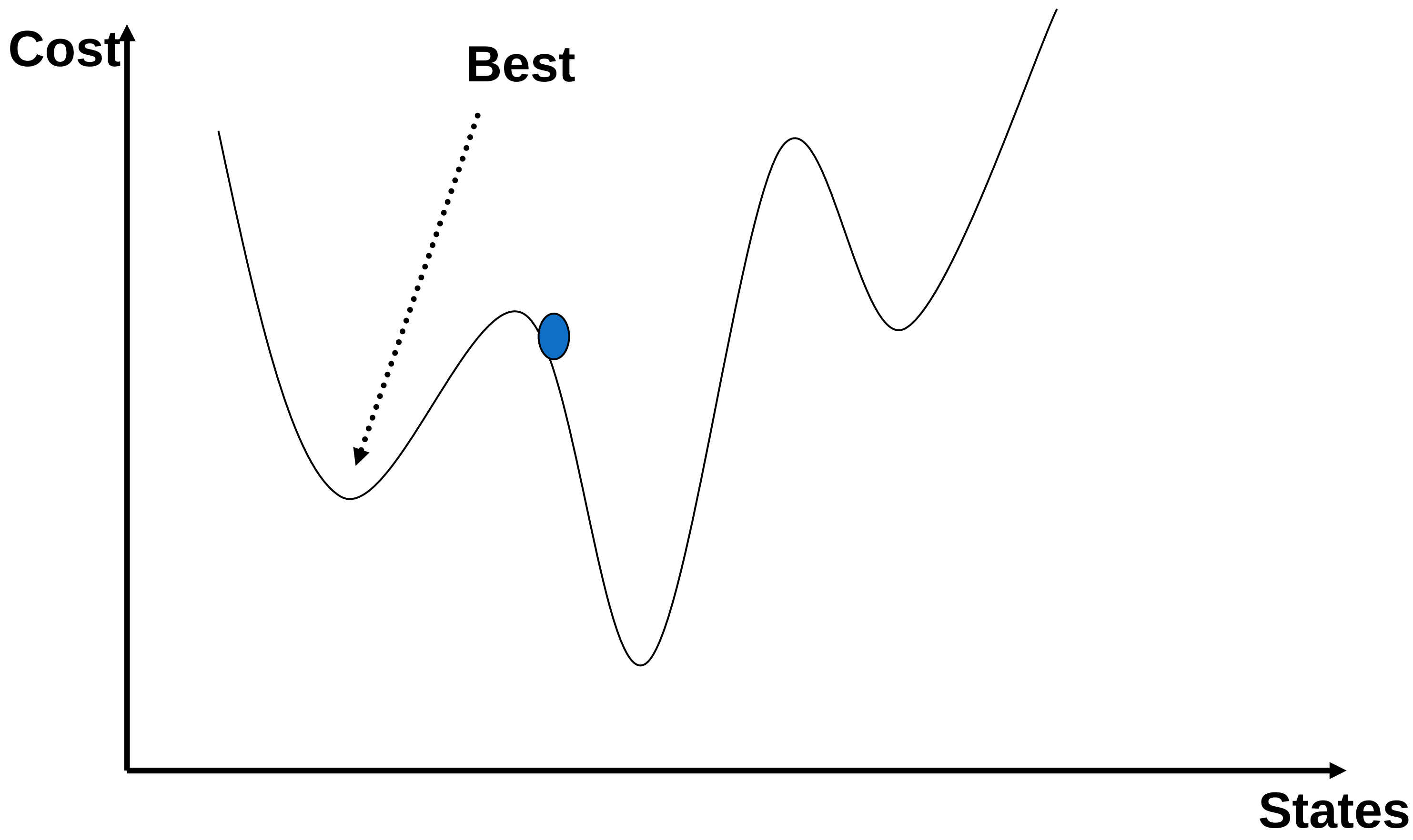
Simulated Annealing



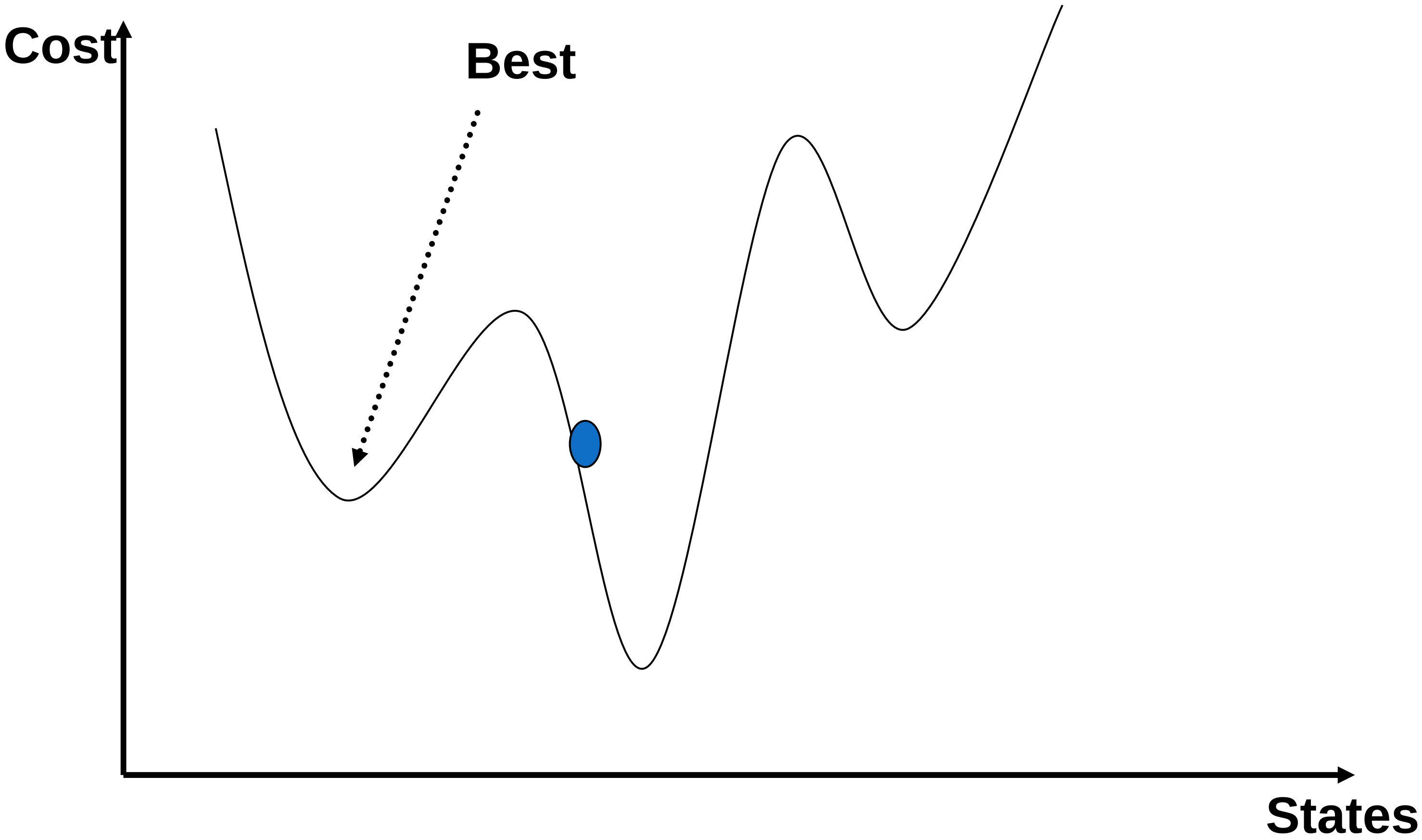
Simulated Annealing



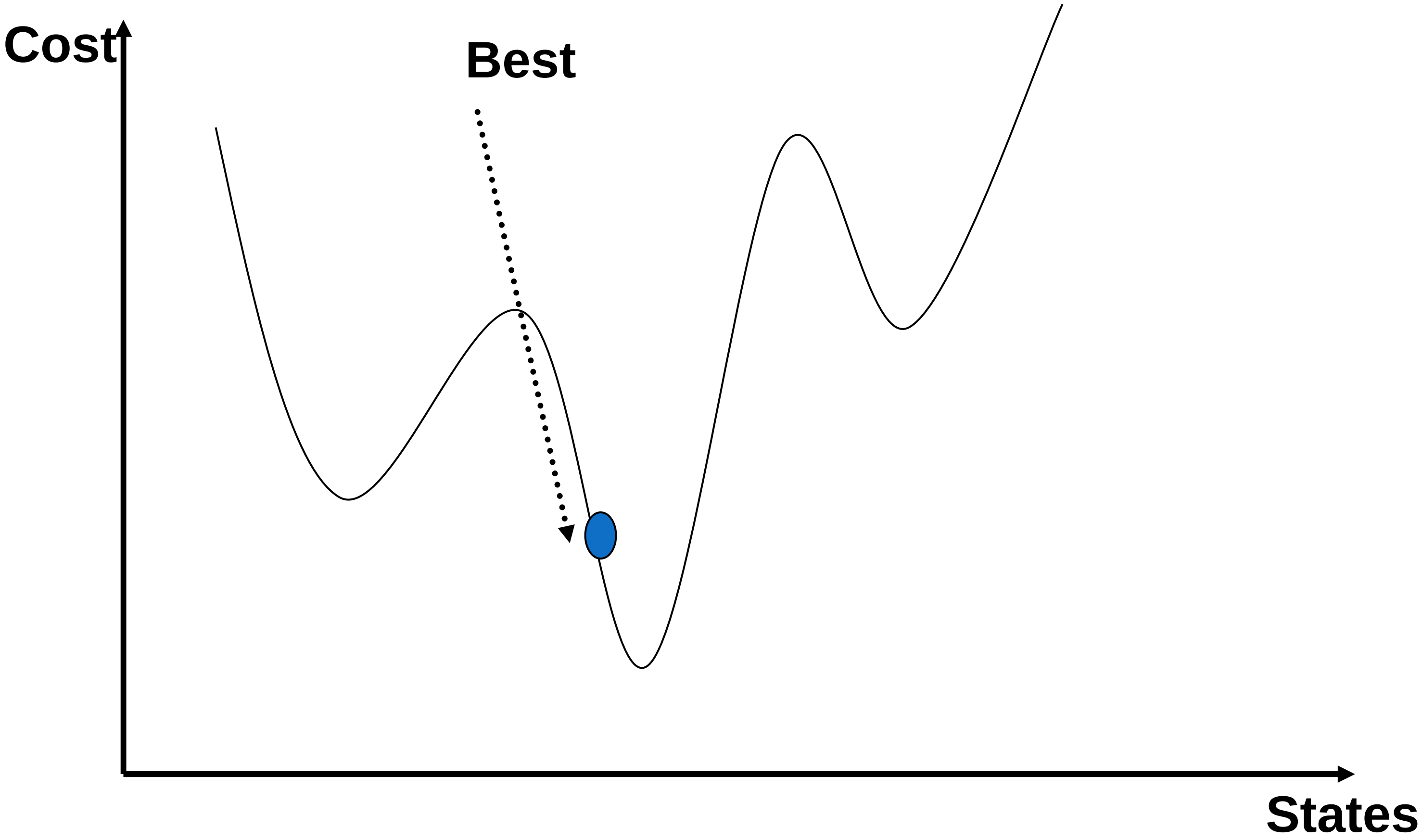
Simulated Annealing



Simulated Annealing

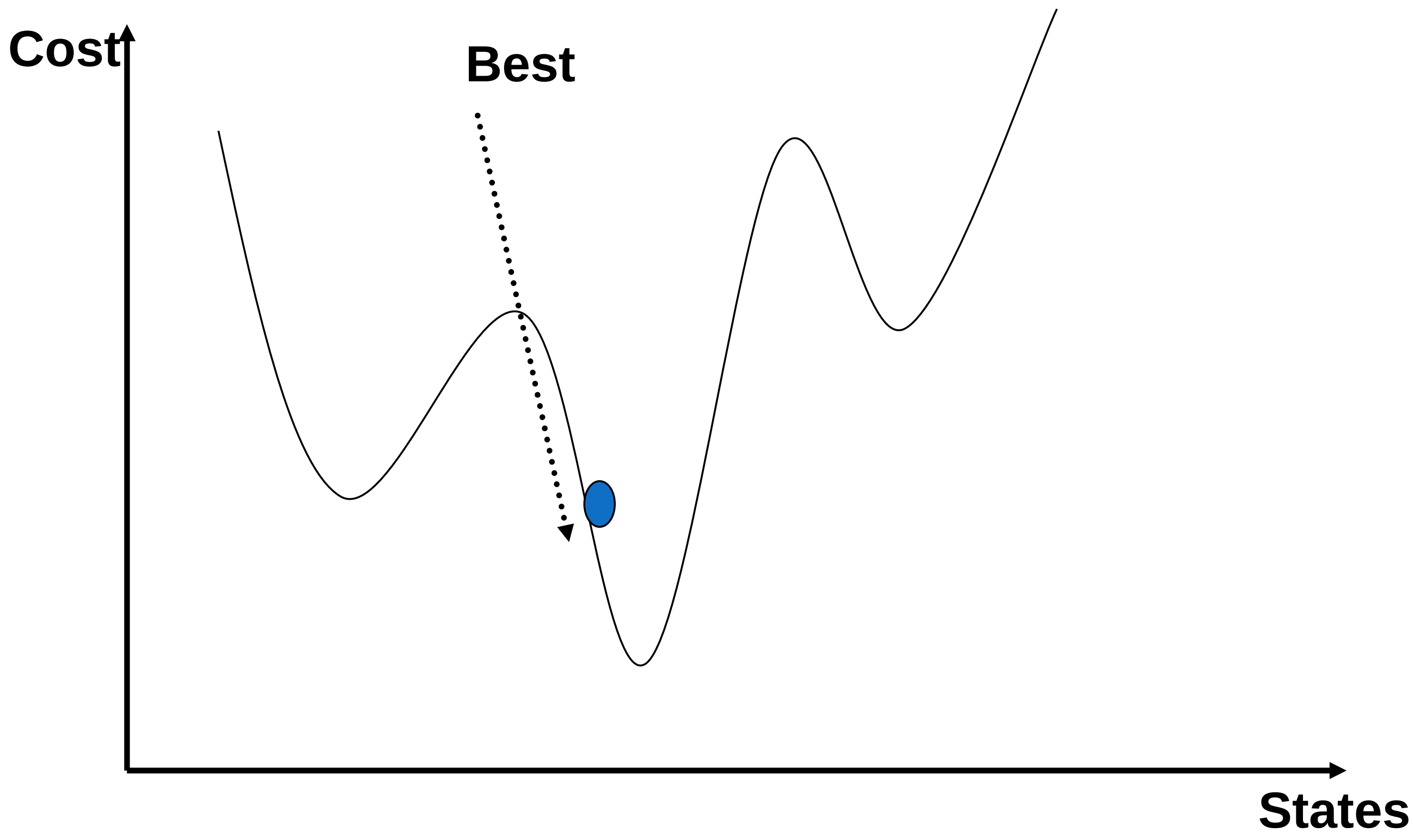


Simulated Annealing

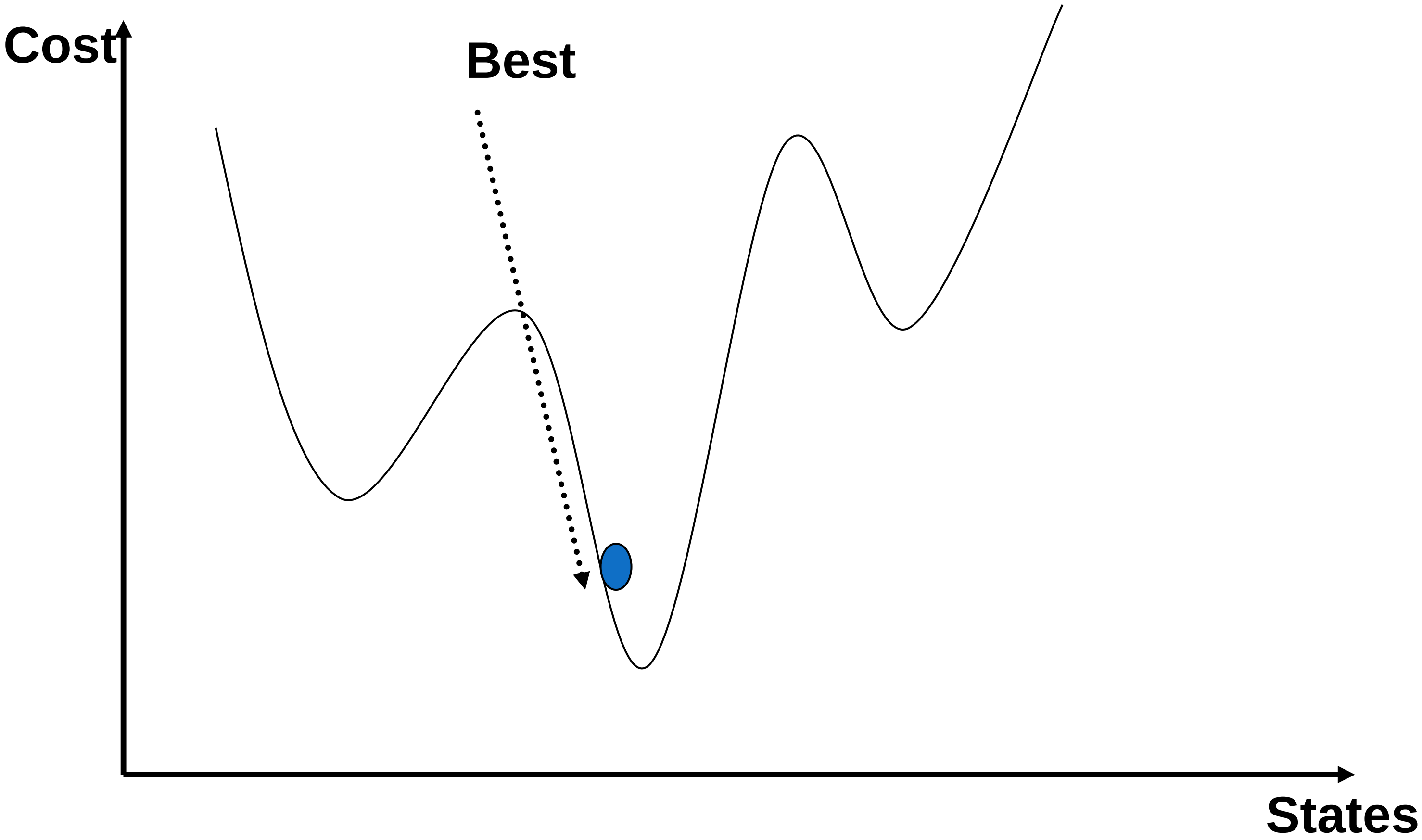


Dr.Yanmei Zheng

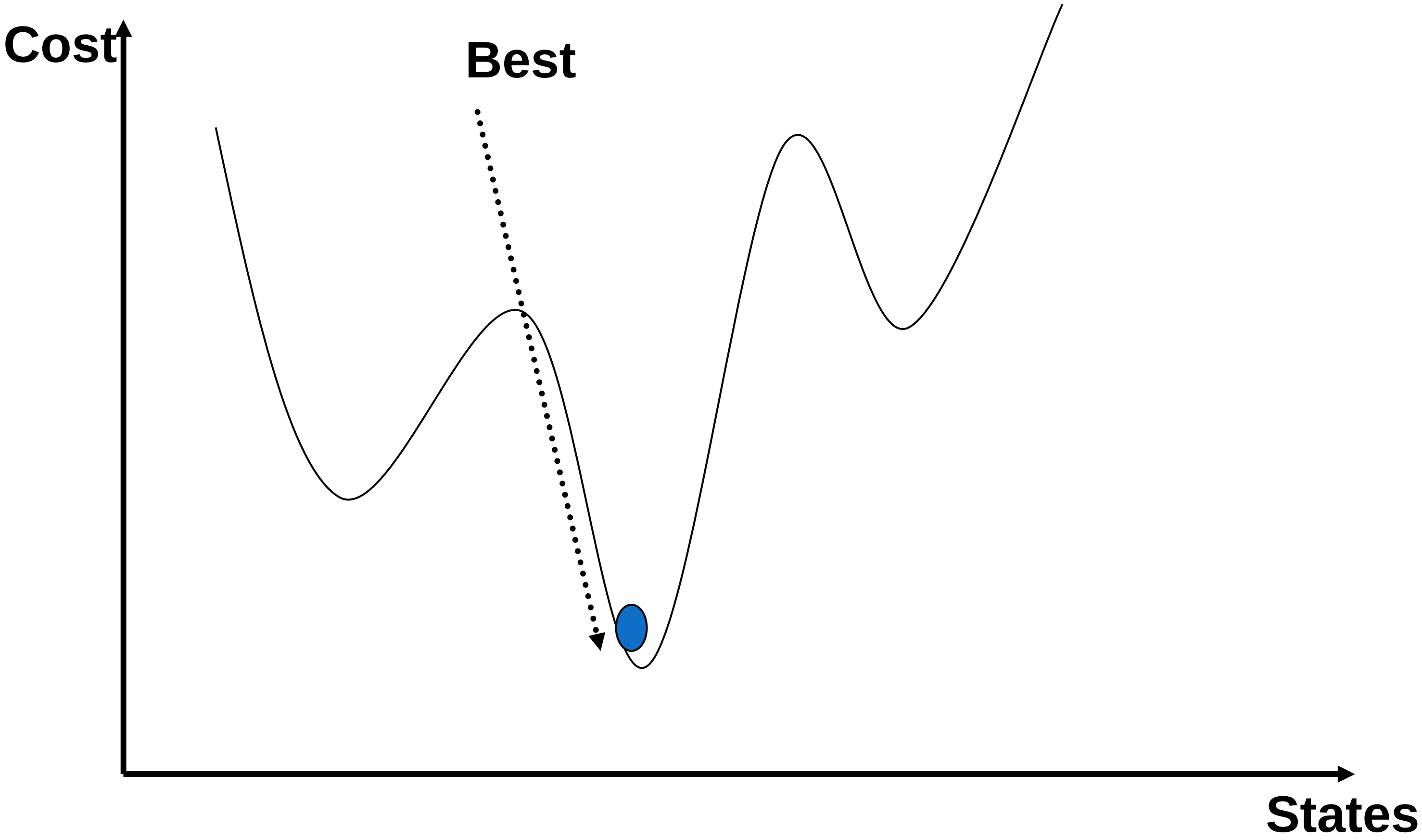
Simulated Annealing



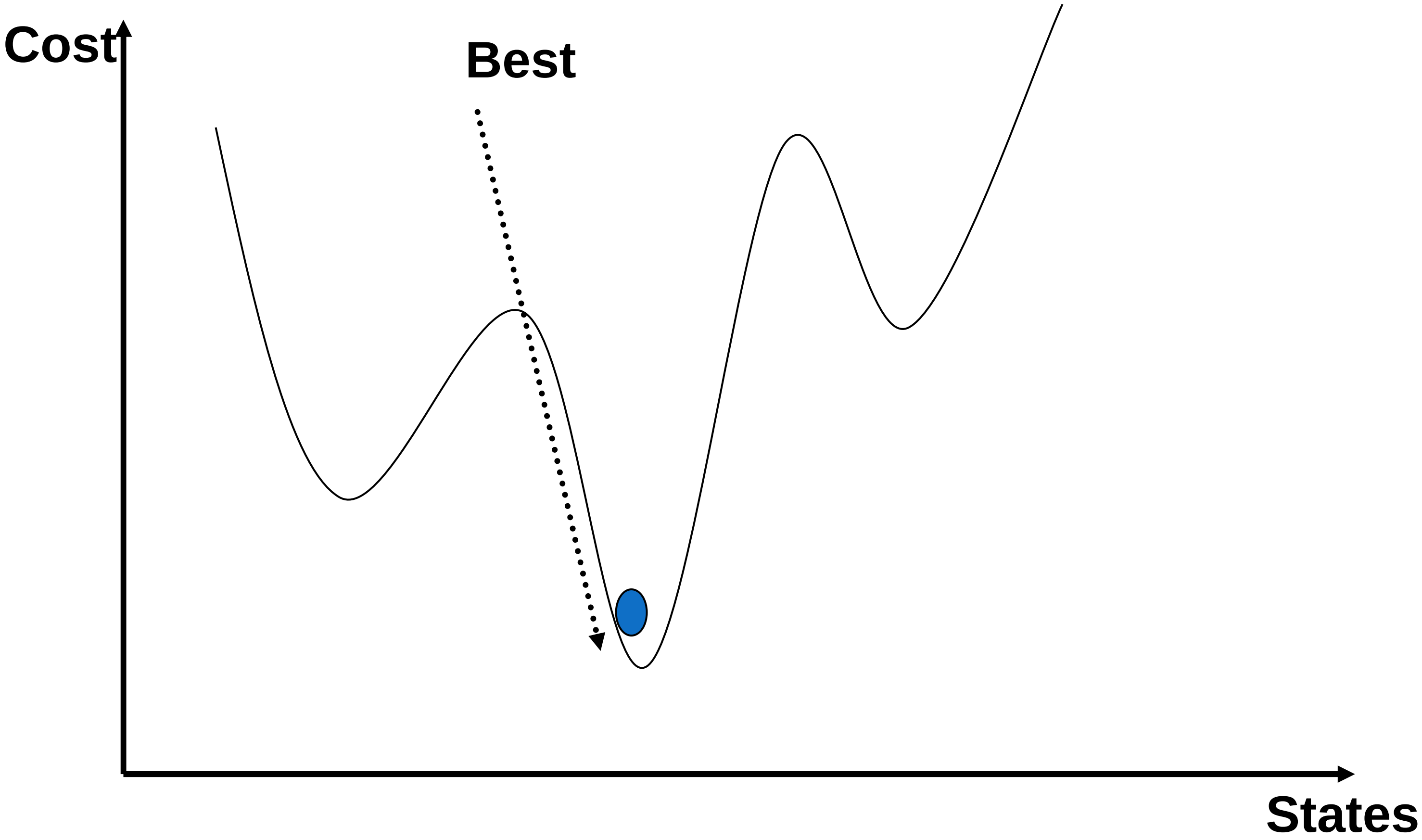
Simulated Annealing



Simulated Annealing

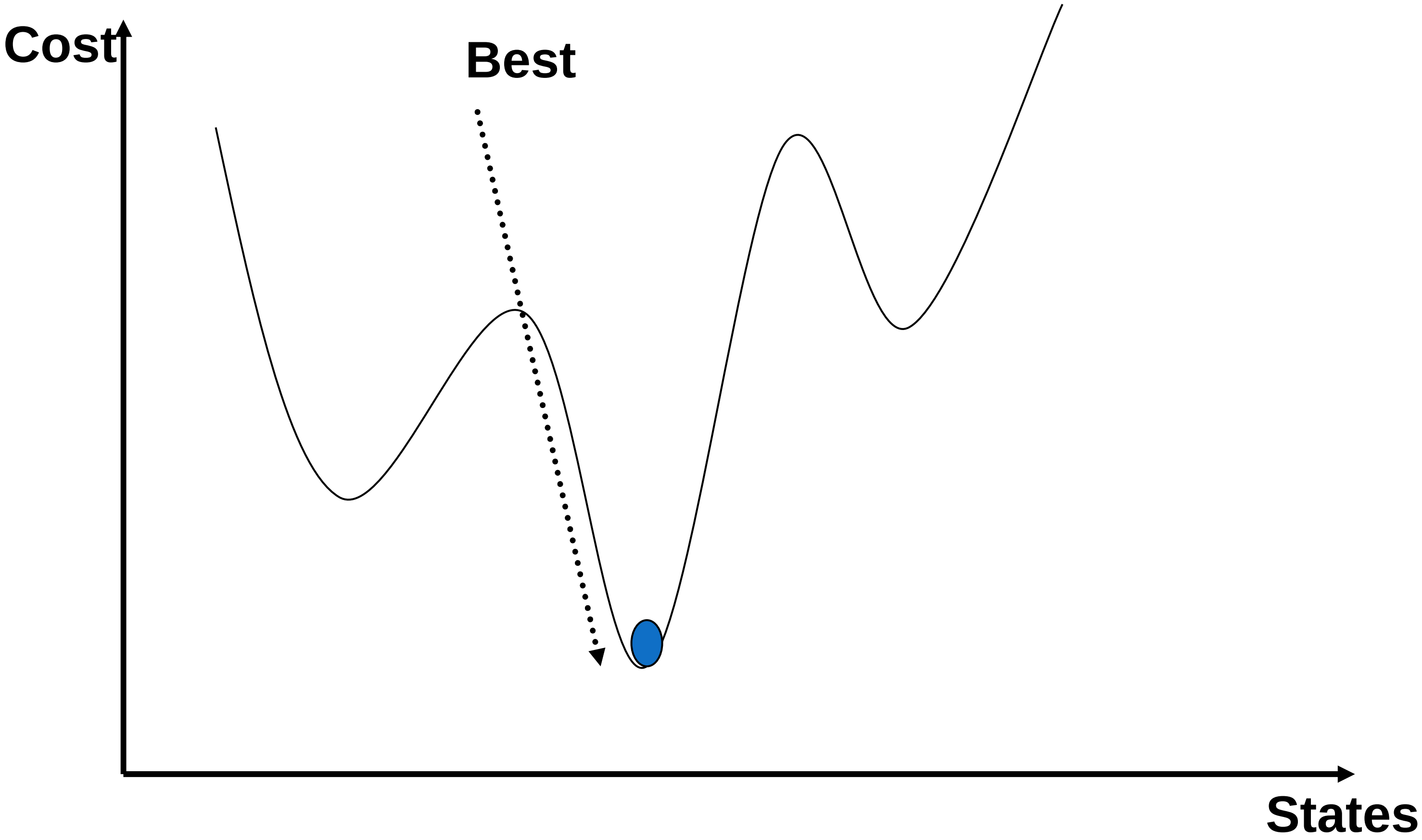


Simulated Annealing

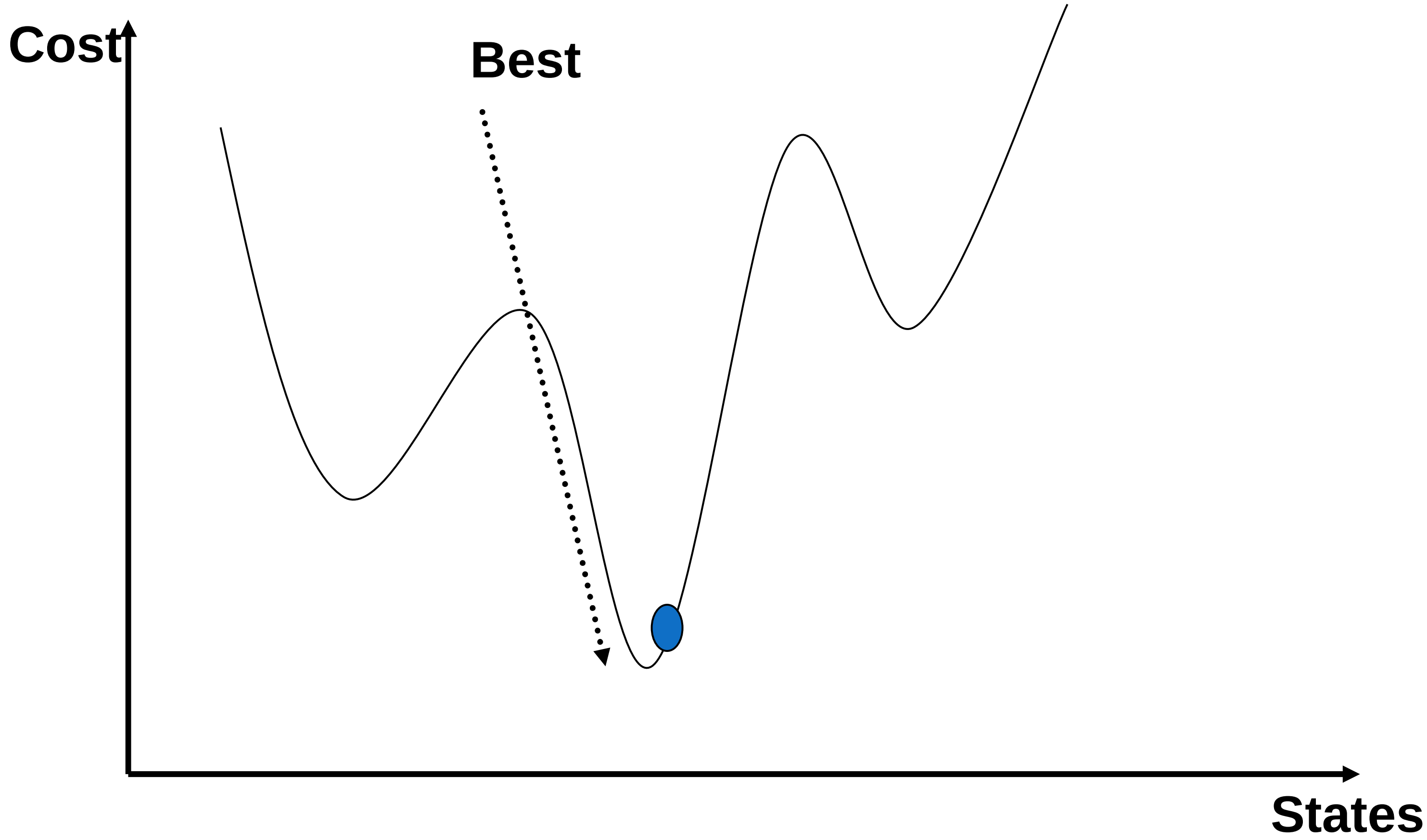


Dr.Yanmei Zheng

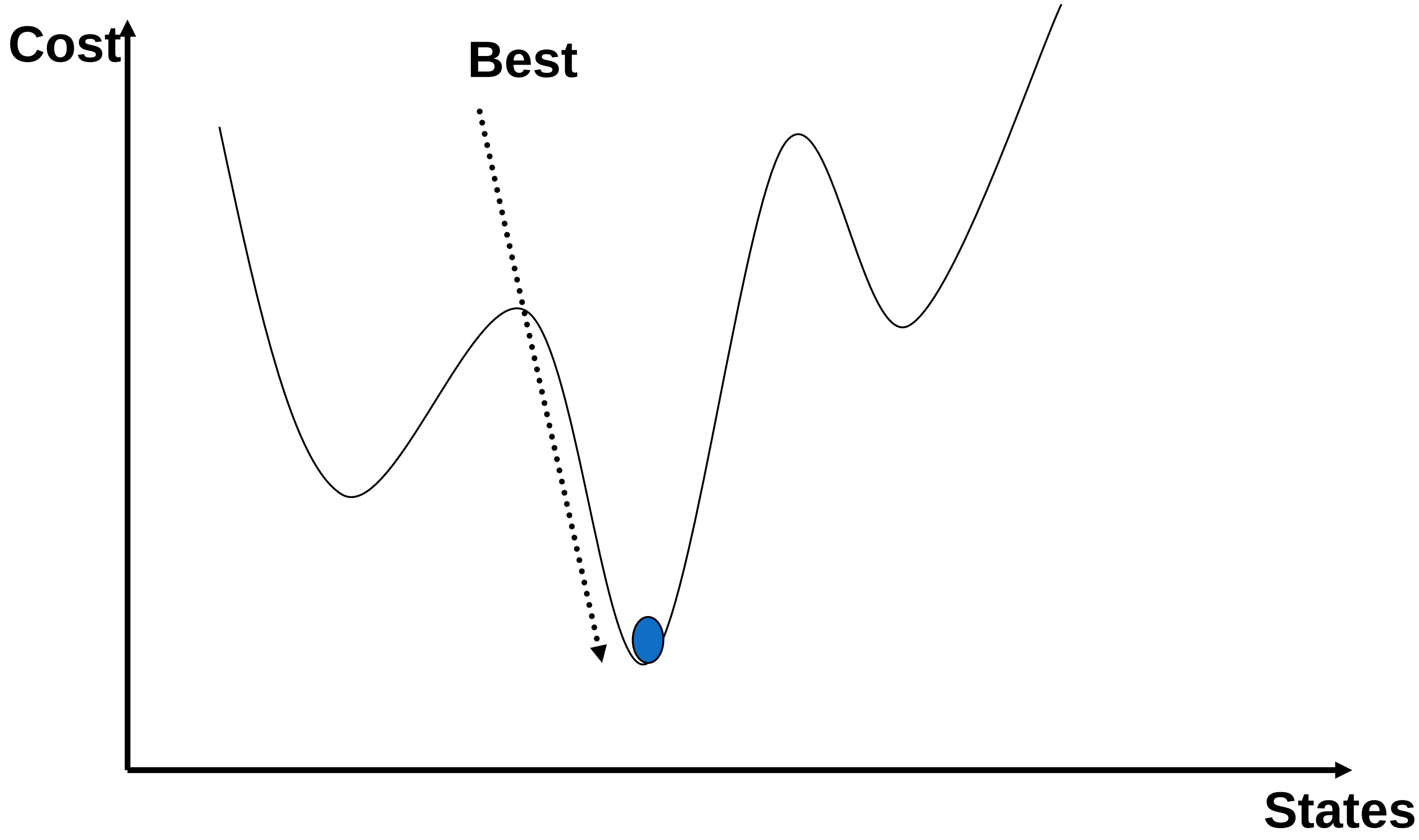
Simulated Annealing



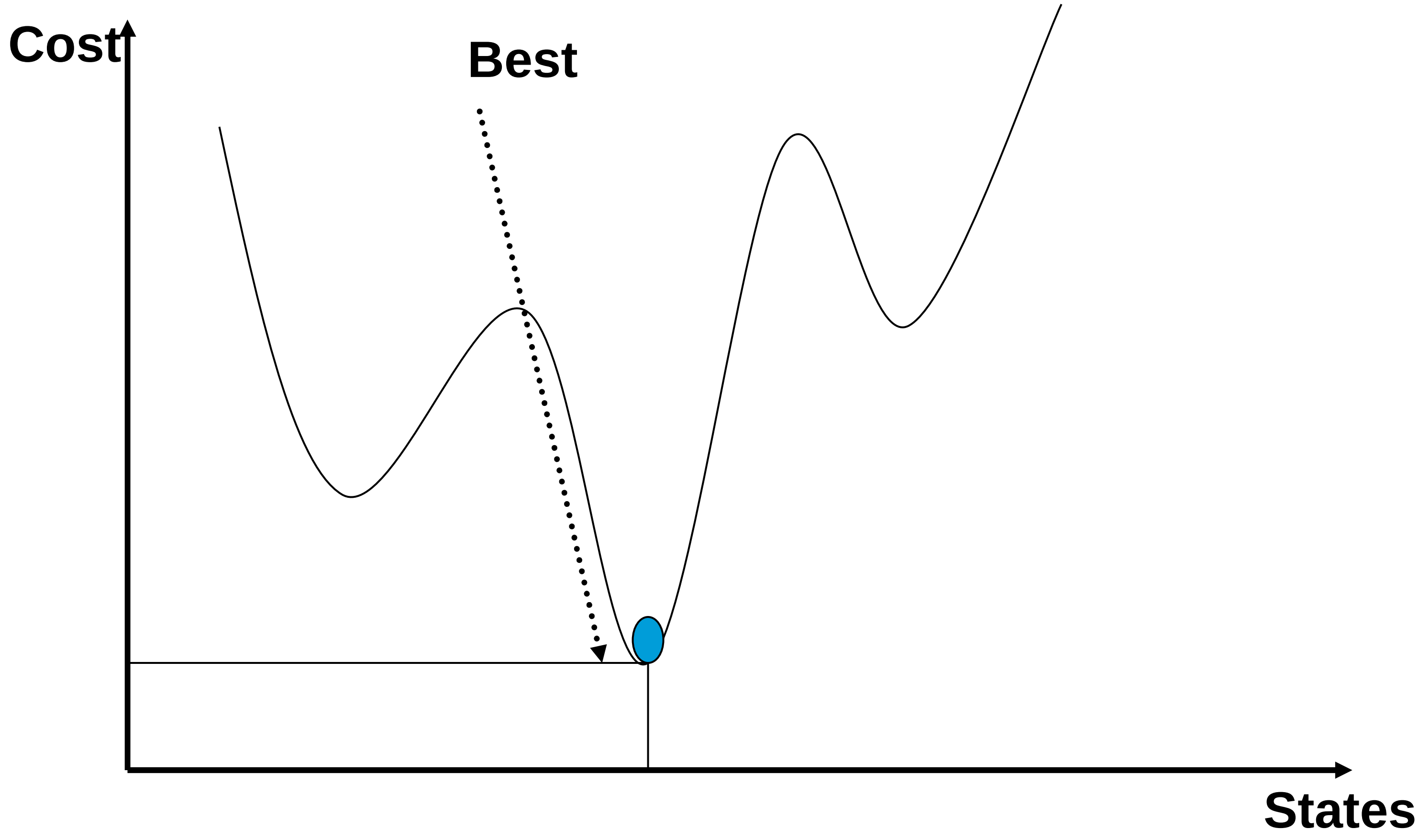
Simulated Annealing



Simulated Annealing



Simulated Annealing



Simulated annealing Search

- Lets say there are 3 moves available, with changes in the objective function of d1 = -0.1, d2 = 0.5, d3 = -5. (Let T = 1).
- pick a move randomly:
  - if d2 is picked, move there.
  - if d1 or d3 are picked, probability of move = exp(d/T)
  - move 1: prob1 = exp(-0.1) = 0.9,
    - i.e., 90% of the time we will accept this move
  - move 3: prob3 = exp(-5) = 0.05
    - i.e., 5% of the time we will accept this move

Properties of Simulated Annealing

- **Cooling Schedule: determines rate at which the temperature T is lowered.**

Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

Simulated Annealing Applications

Solve the TSP problem

TSP，Traveling Salesman Problem： There are N cities. We need to start from one of them, go to all the cities at one time, and finally return to the city where we started, and find the shortest route.

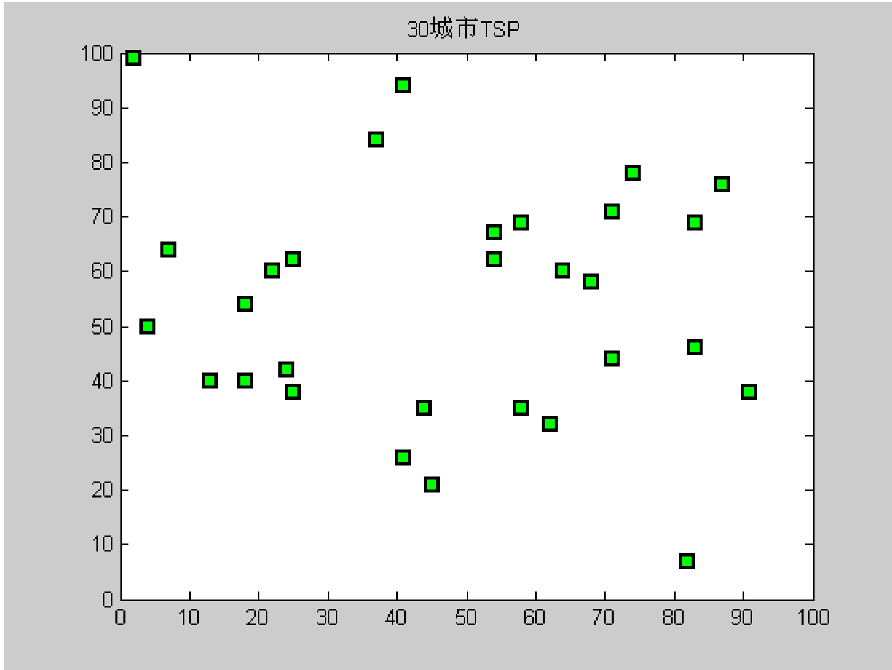
The shortest route between several cities →

Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

- **TSP Problem**

  41 94;37 84;54 67;25 62; 7 64;2 99;68 58;71 44;54 62;83 69;64 60;18 54;22 60;83 46;91 38;25 38;24 42;58 69;71 71;74 78;87 76;18 40;13 40;82 7;62 32; 58 35;45 21;41 26;44 35;4 50



Solution of TSP problem

1.Select an initial path P(i) and calculate L(P(i)).

2. Generate the length of a new traversal path P(i+1), and calculate the length L(P(i +1) of path P(i+1));

3. If L(P(i +1)) is less than L(P(i)), then accept P(i +1) as the new path, otherwise accept P(i+1) following the Metropolis criterion and then cool down.

4. Repeat 2,3 until the exit conditions are satified.

There are several ways to generate a new path:

1). Randomly select 2 nodes and exchange the order of these 2 points in the path.

2). Randomly select 2 nodes and reverse the order of nodes between 2 nodes in the path.

3). Randomly select 3 nodes m, n and k, and then move the nodes between m and n to the node behind k.

Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

- **Calculation of initial temperature**

```
for i=1:100
    route=randperm(CityNum);
    fval0(i)=CalDist(dislist,route);
end
t0=-(max(fval0)-min(fval0))/log(0.9);
```

Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

- **Design of the state generation functions**

（1） Swap operation, randomly swap the order of two cities

2	8	3	5	9	1	4	6	7		2	8	1	5	9	3	4	6	7

（2） Inversion operation, invert the cities between two random locations;

2	8	3	5	9	1	4	6	7	⟹	2	8	3	4	1	9	5	6	7

（3） Insert operation, randomly select a point to insert a random position.

2	8	3	5	9	1	4	6	7	⟹	2	3	5	9	8	1	4	6	7

Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

- **Parameter setting**

Temperature tf=0.01;

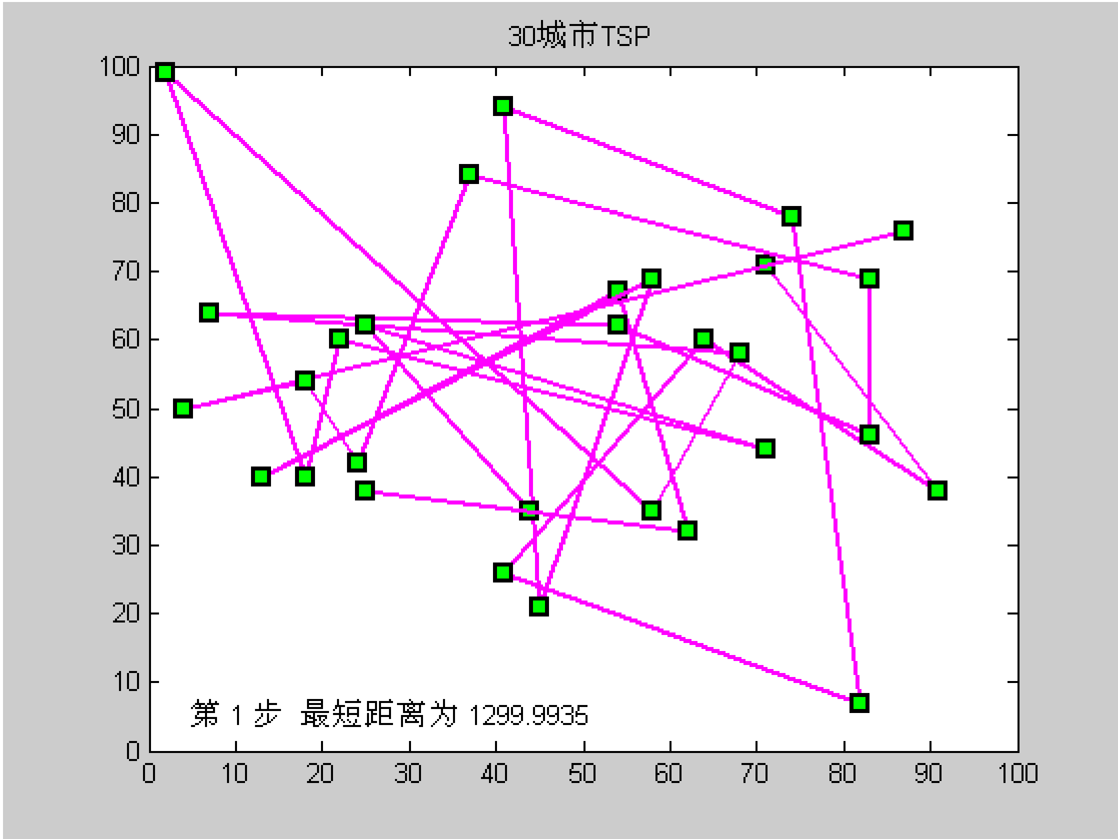
Annealing temperature coefficient alpha=0.90;

Number of iterations L=200*CityNum;

Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

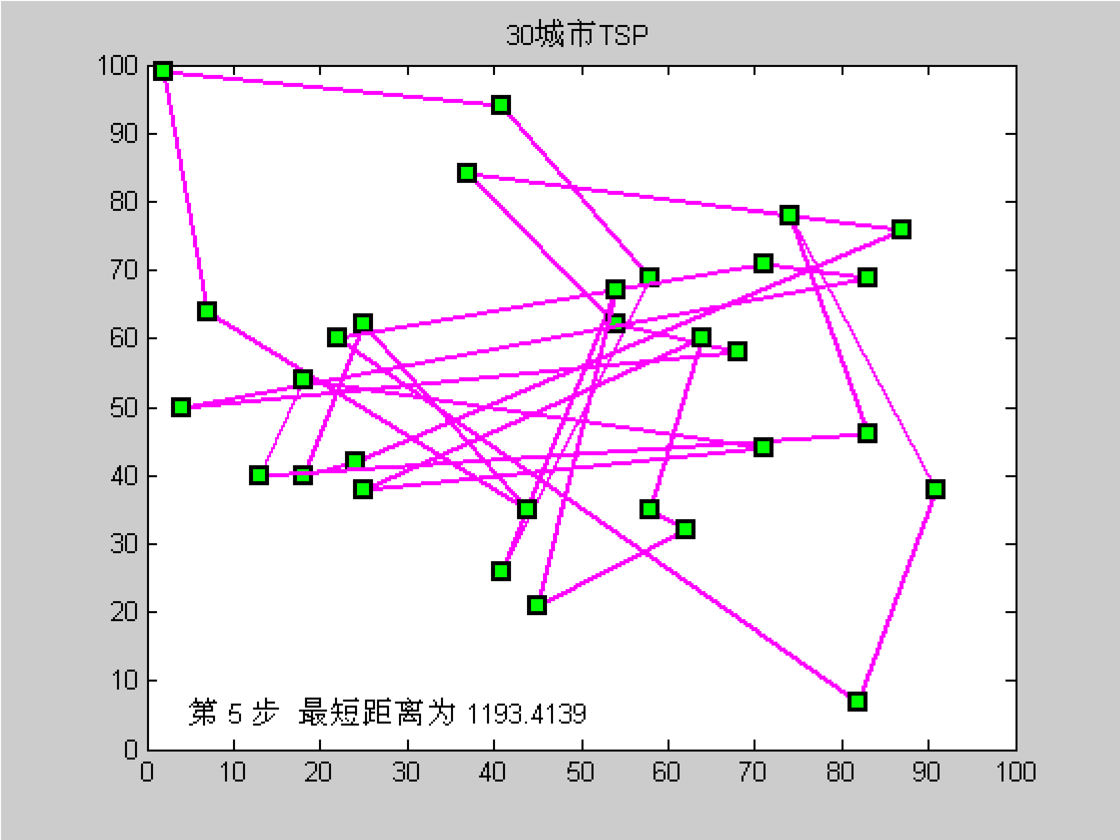
- **Step 1st**



Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

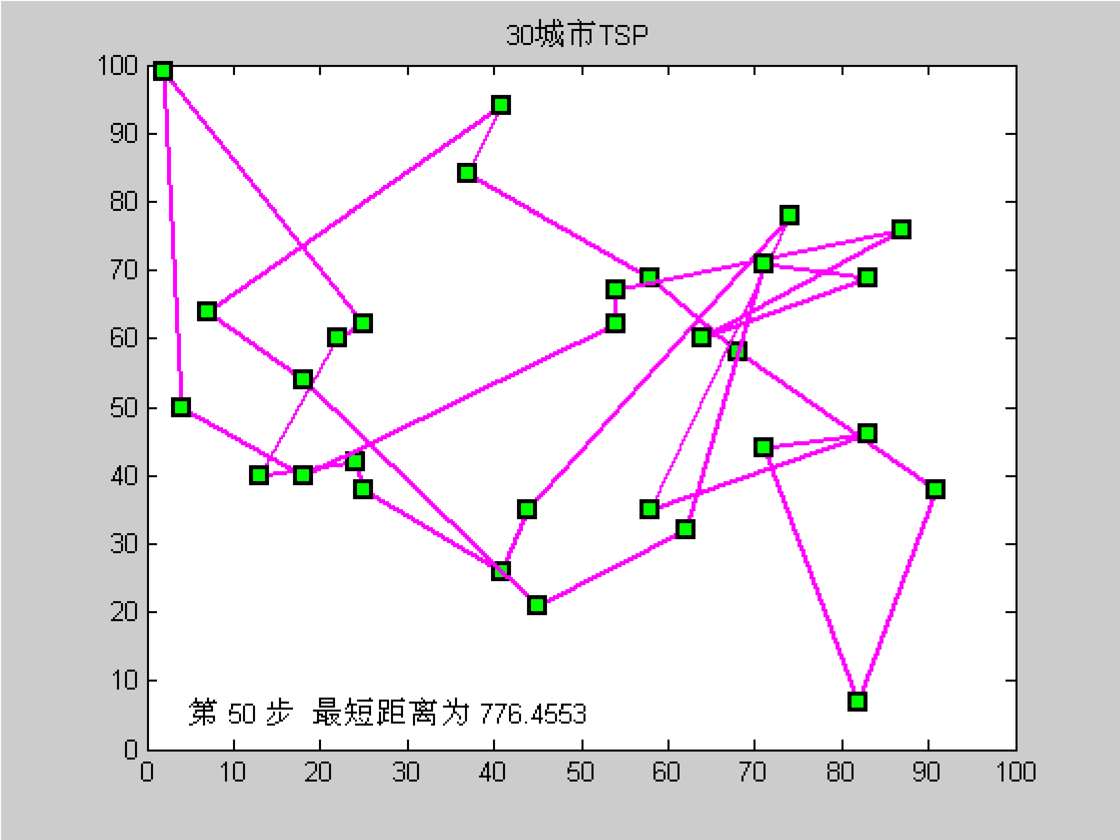
- **Step 5th**



Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

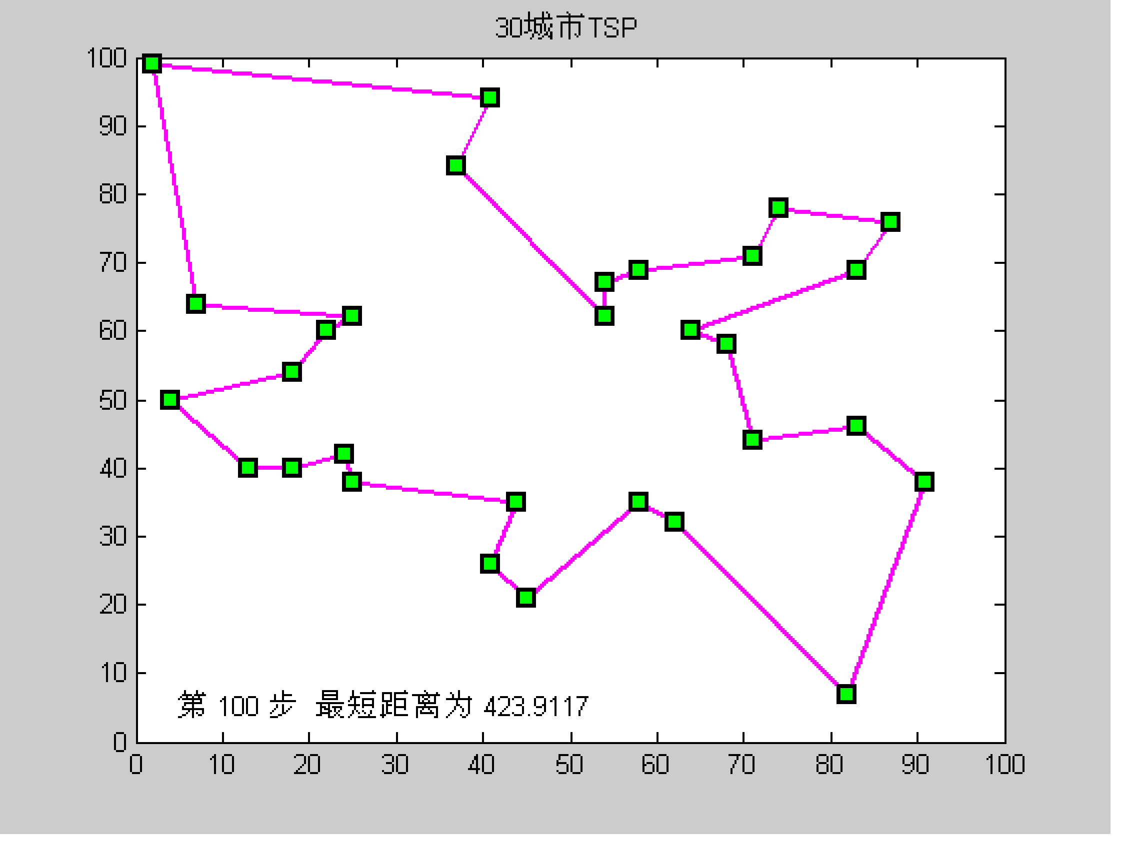
- Step 50th



Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

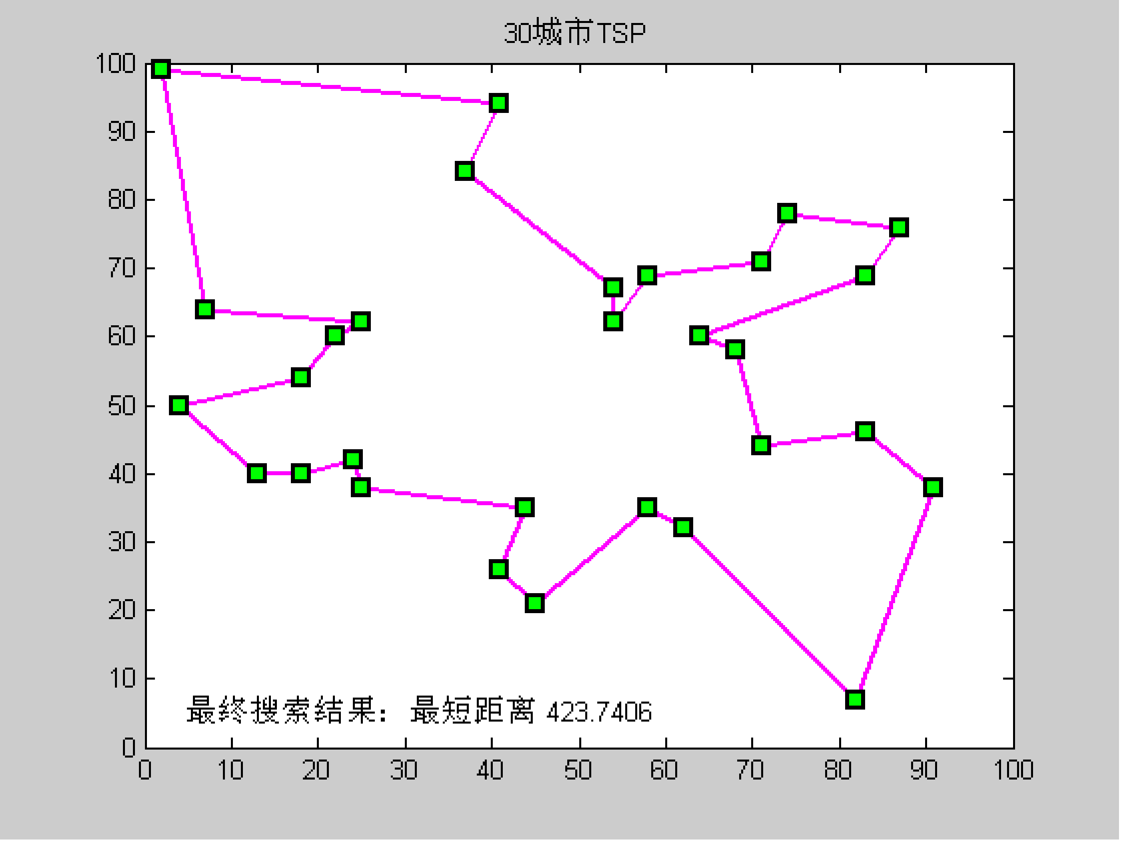
- **Step 100th**



Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

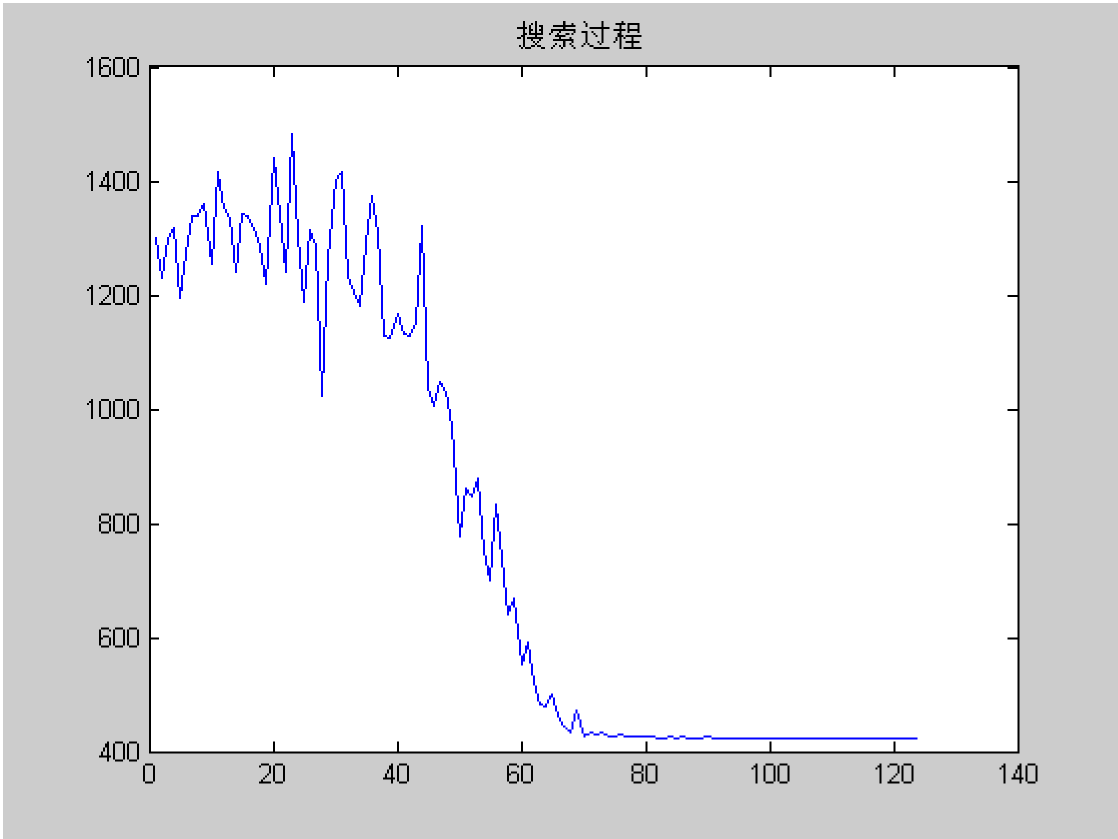
- **Final result**



Simulated Annealing Applications

30 Cities TSP problem （d^*=423.741 by D B Fogel）

- **The objective function**

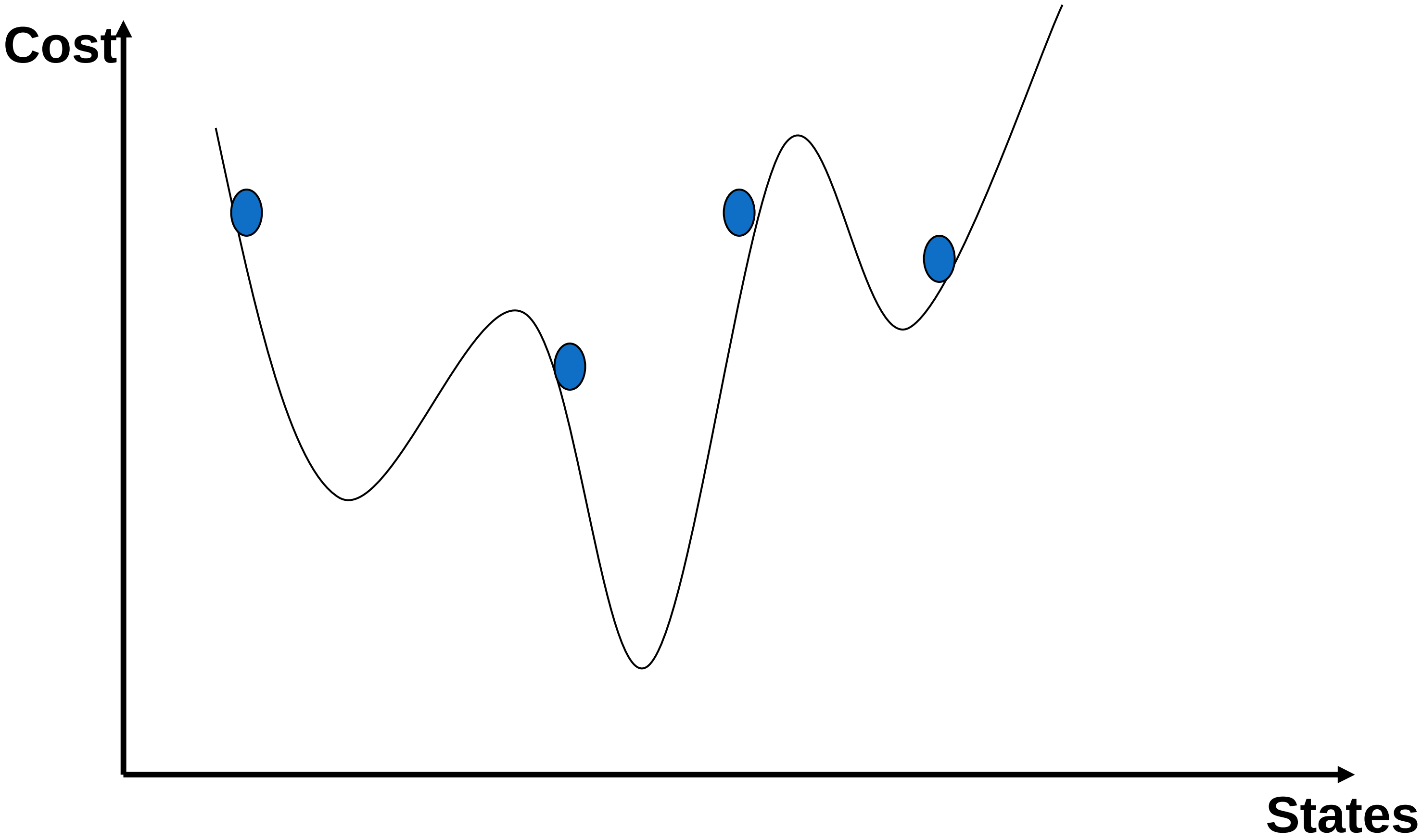


Local Beam Search

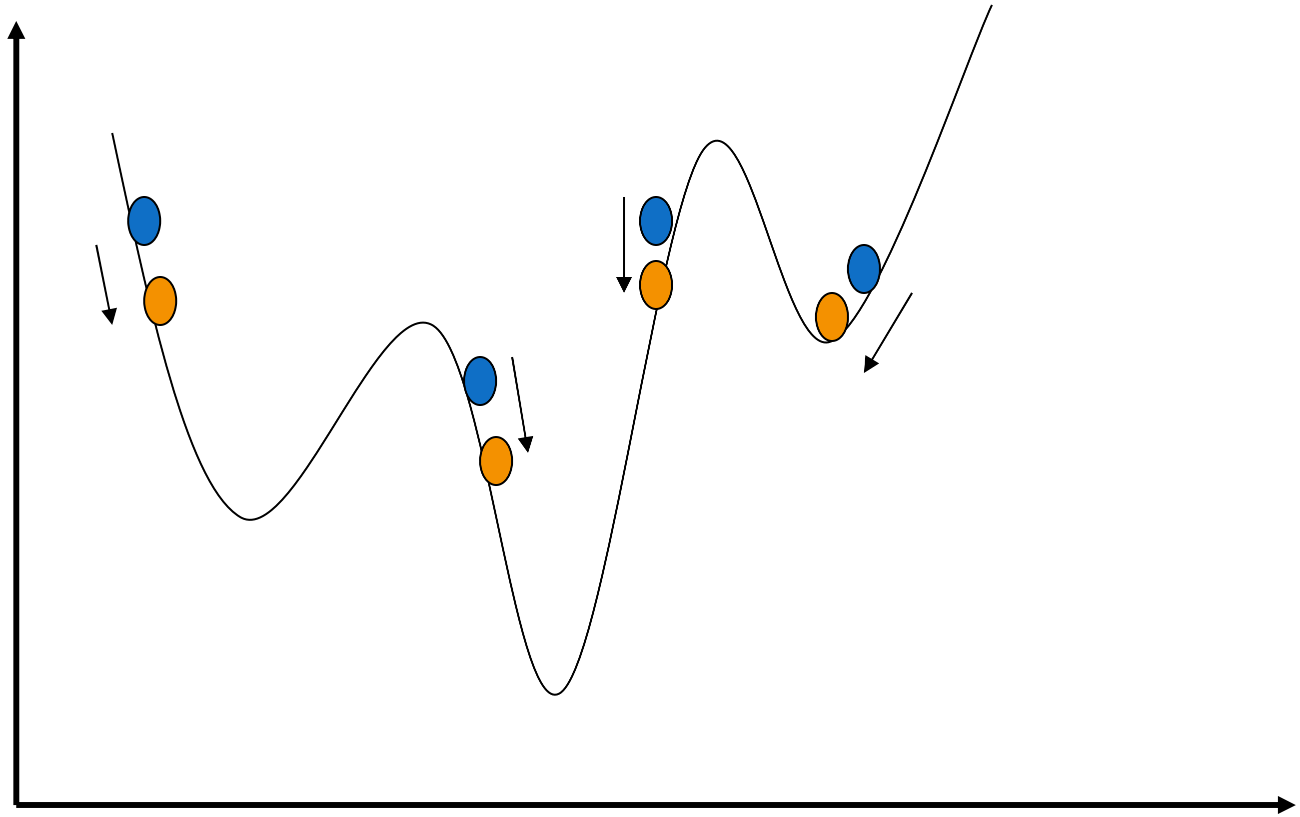
Local Beam Search

- **Main 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.
- **Drawback**: the k states tend to regroup very quickly in the same region → lack of diversity.

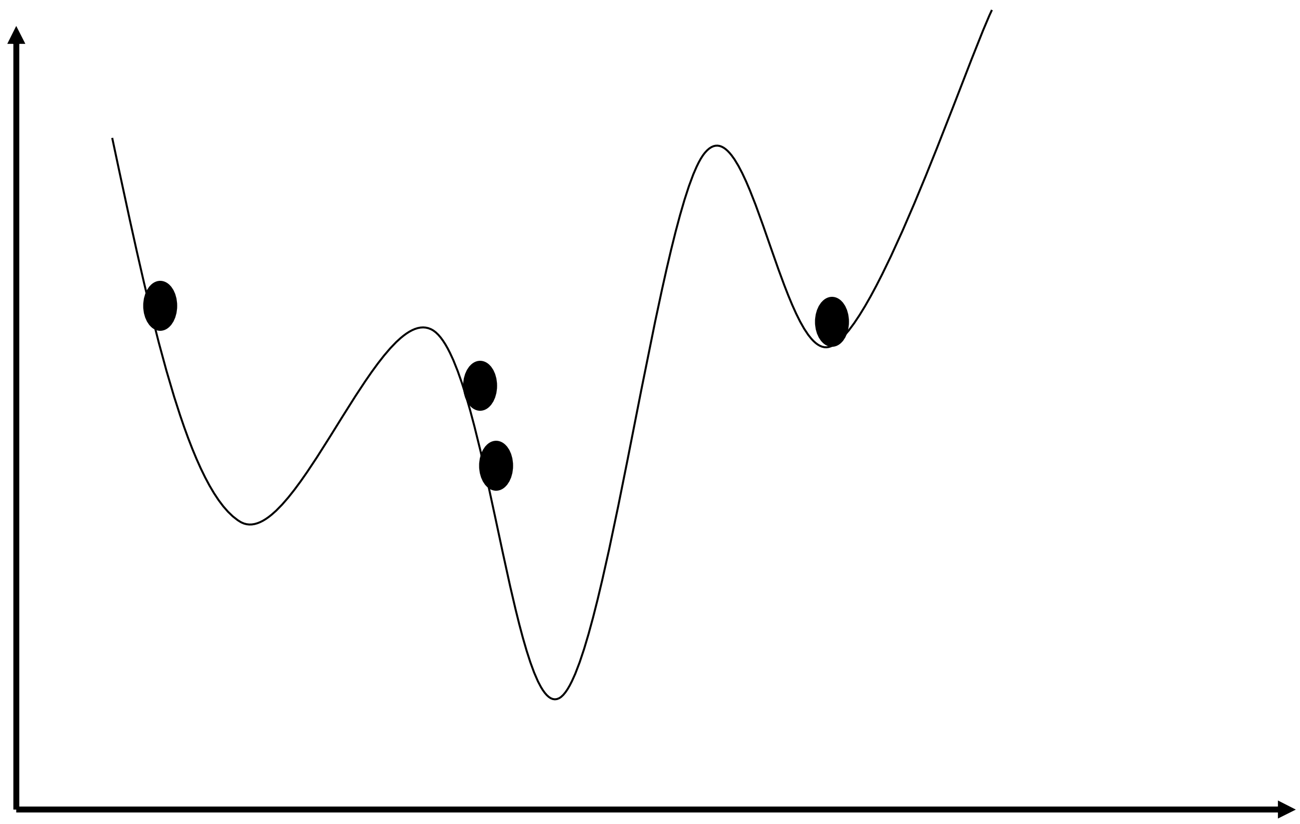
Local Beam Search



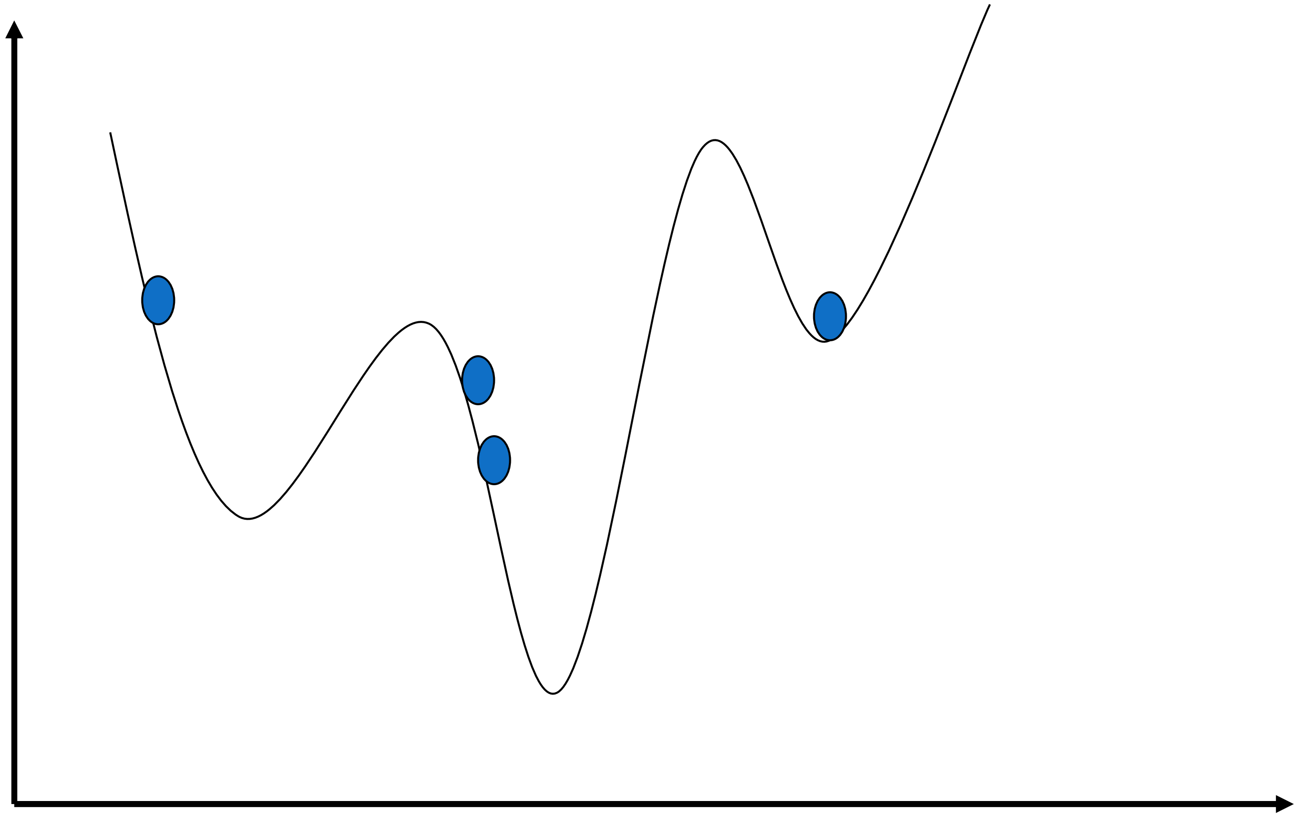
Local Beam Search



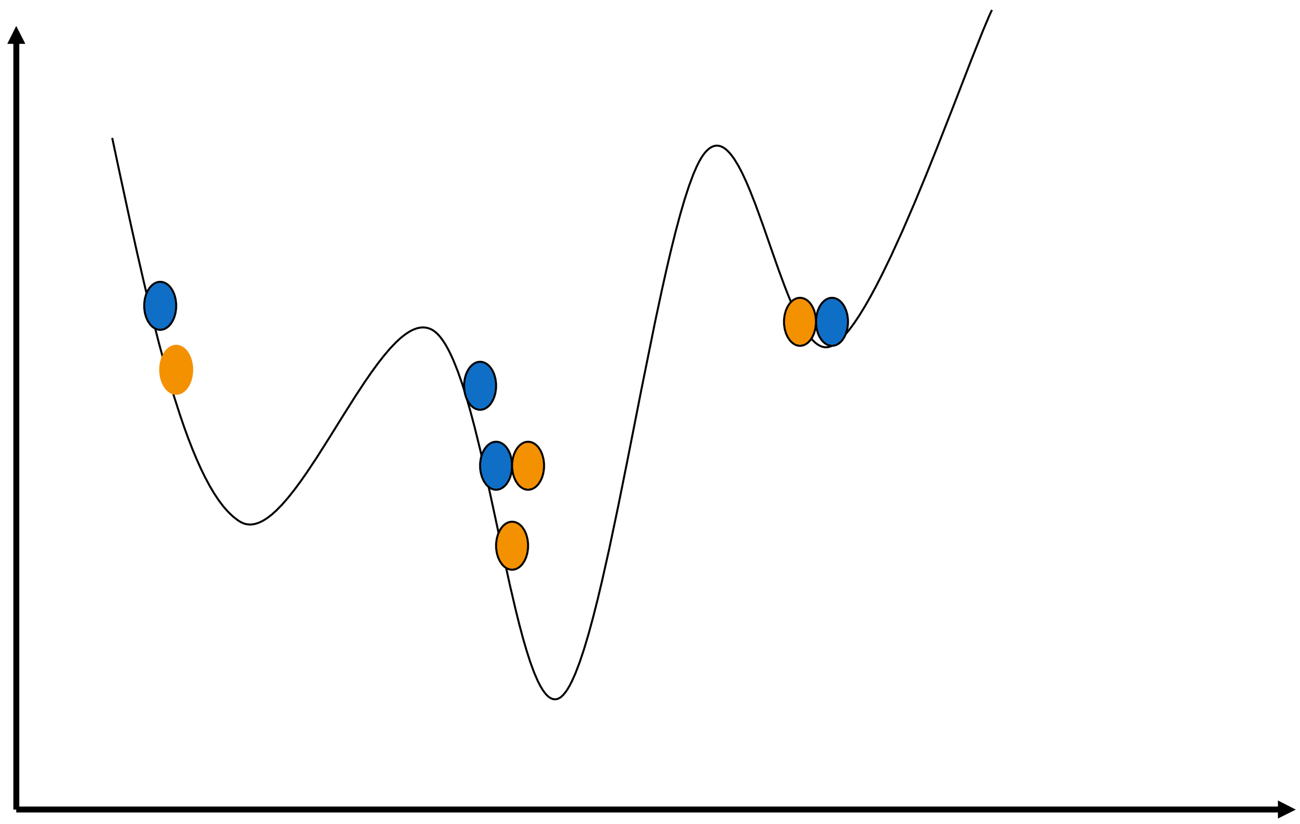
Local Beam Search



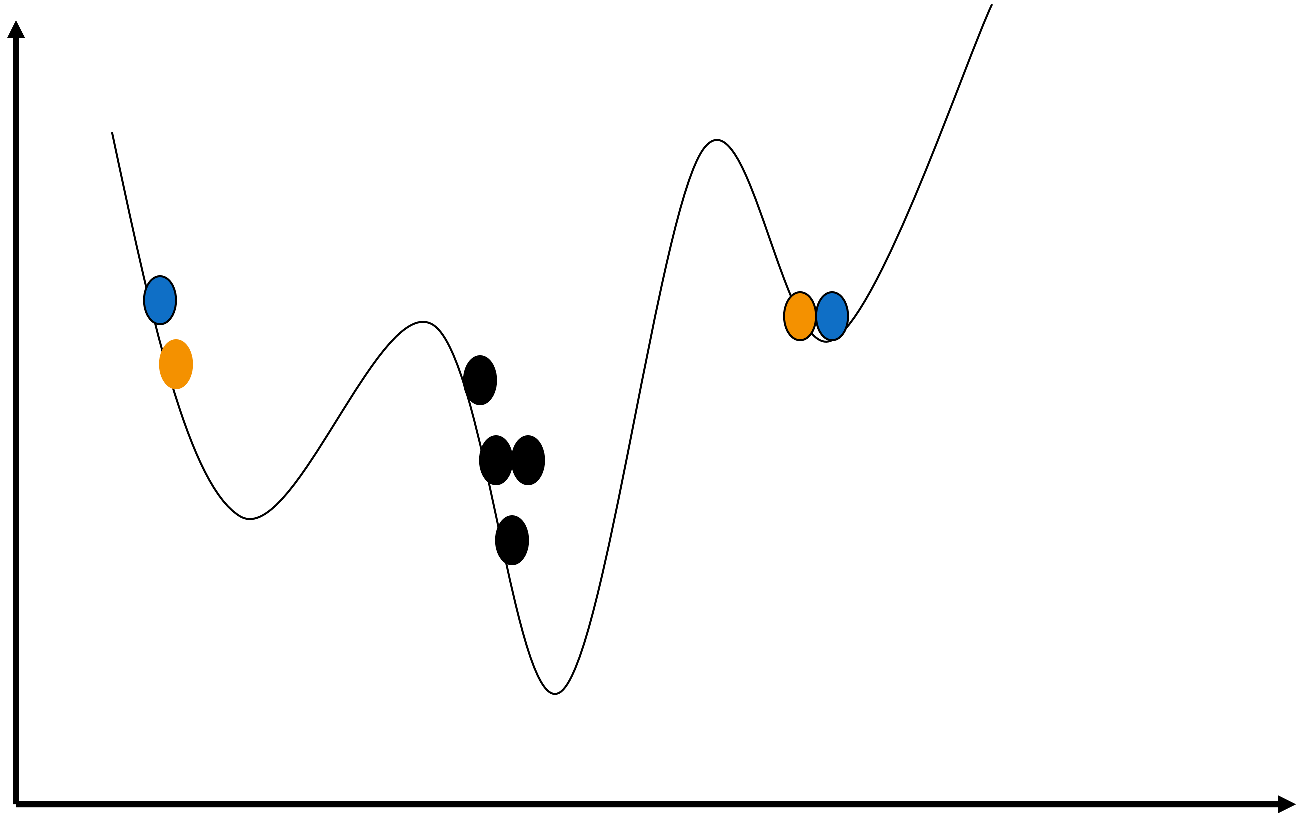
Local Beam Search



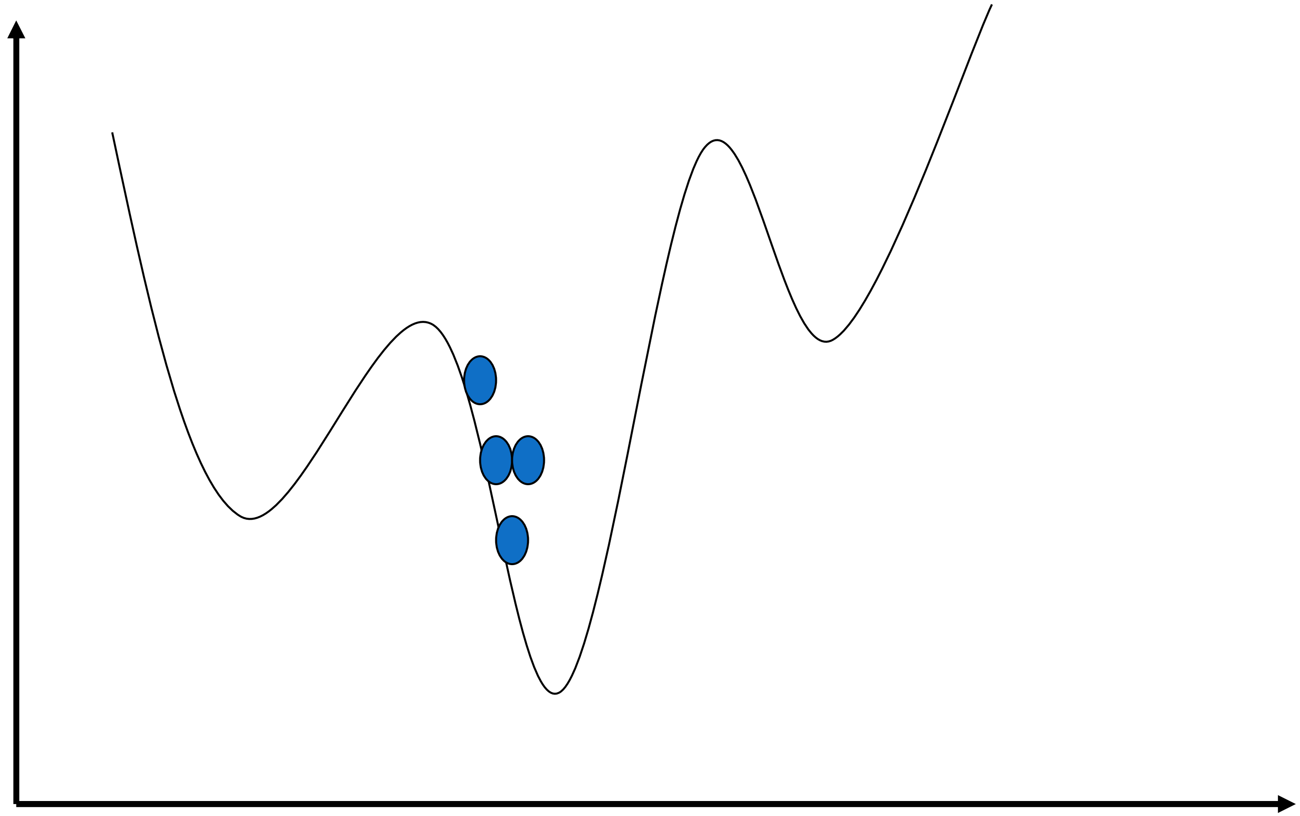
Local Beam Search



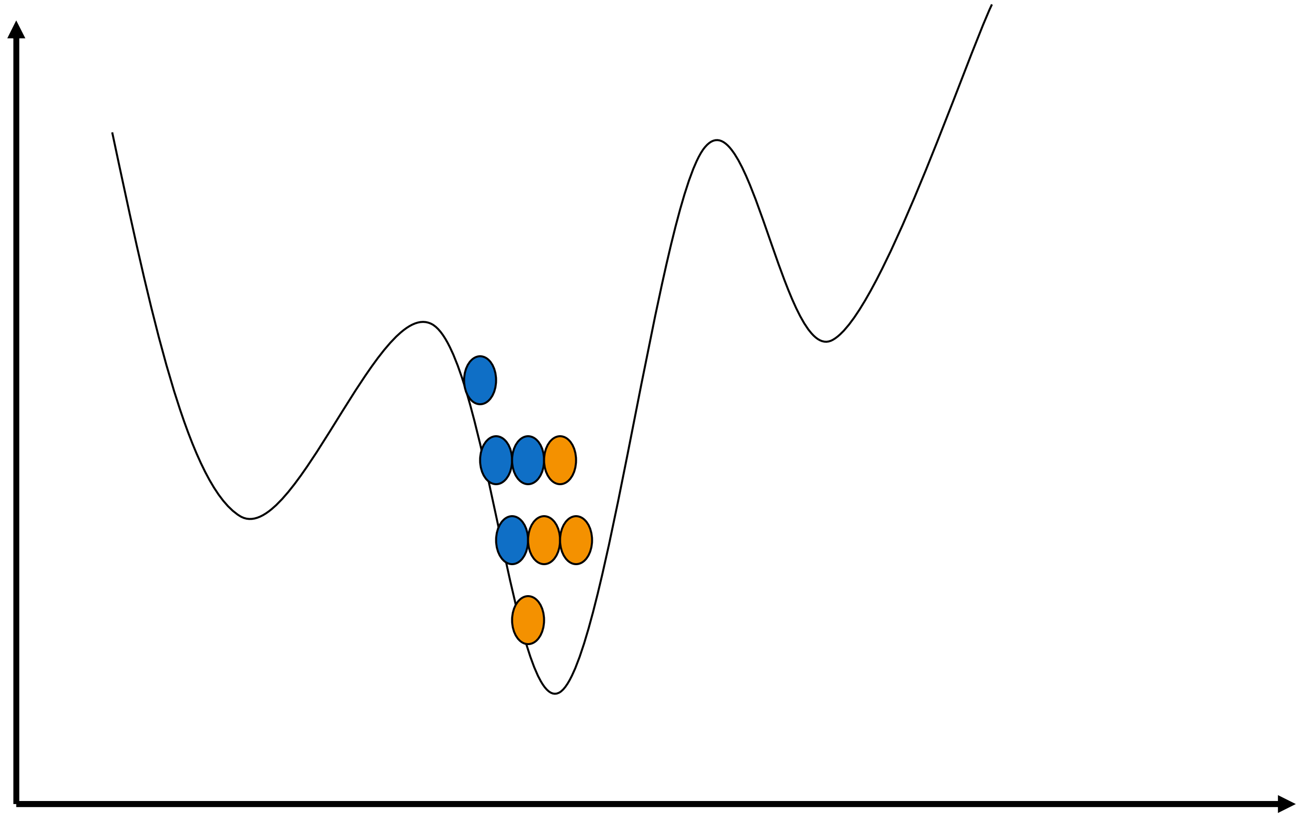
Local Beam Search



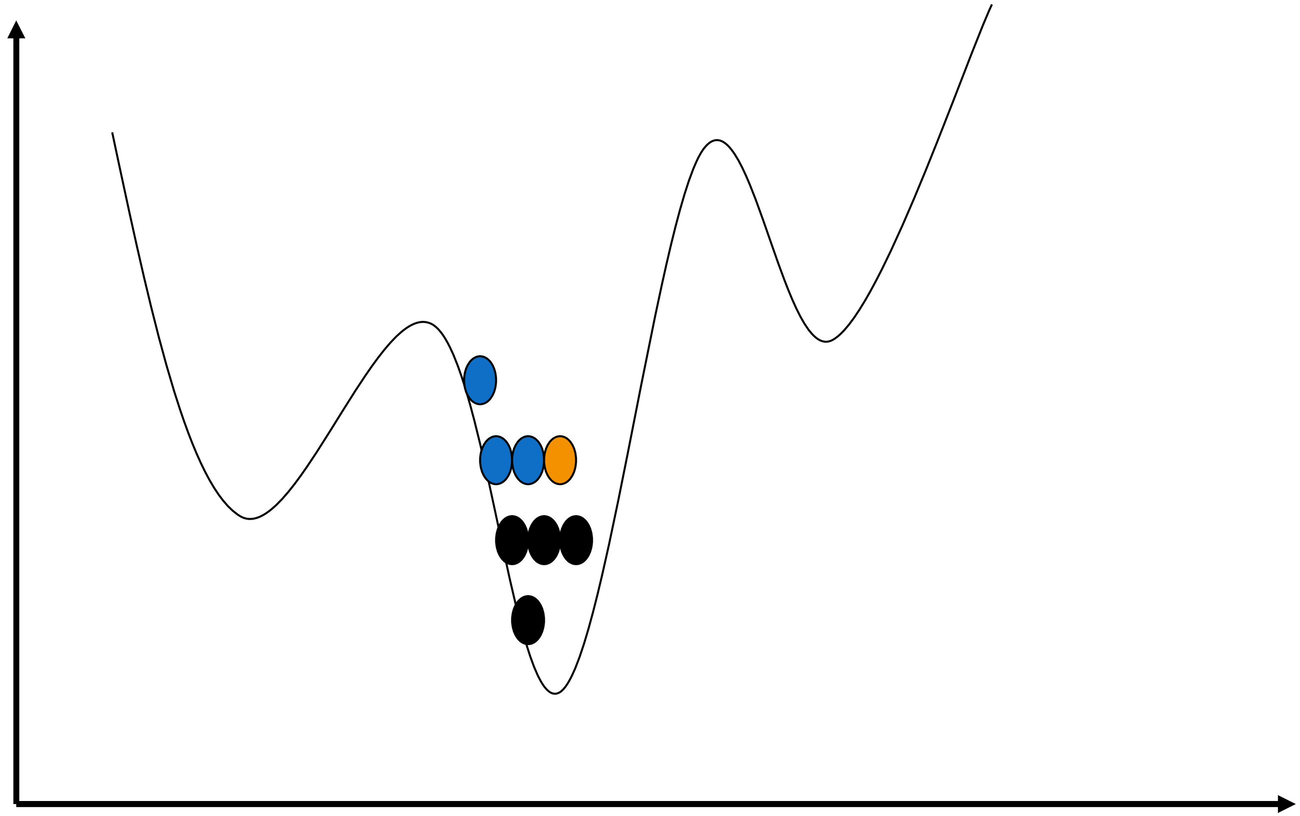
Local Beam Search



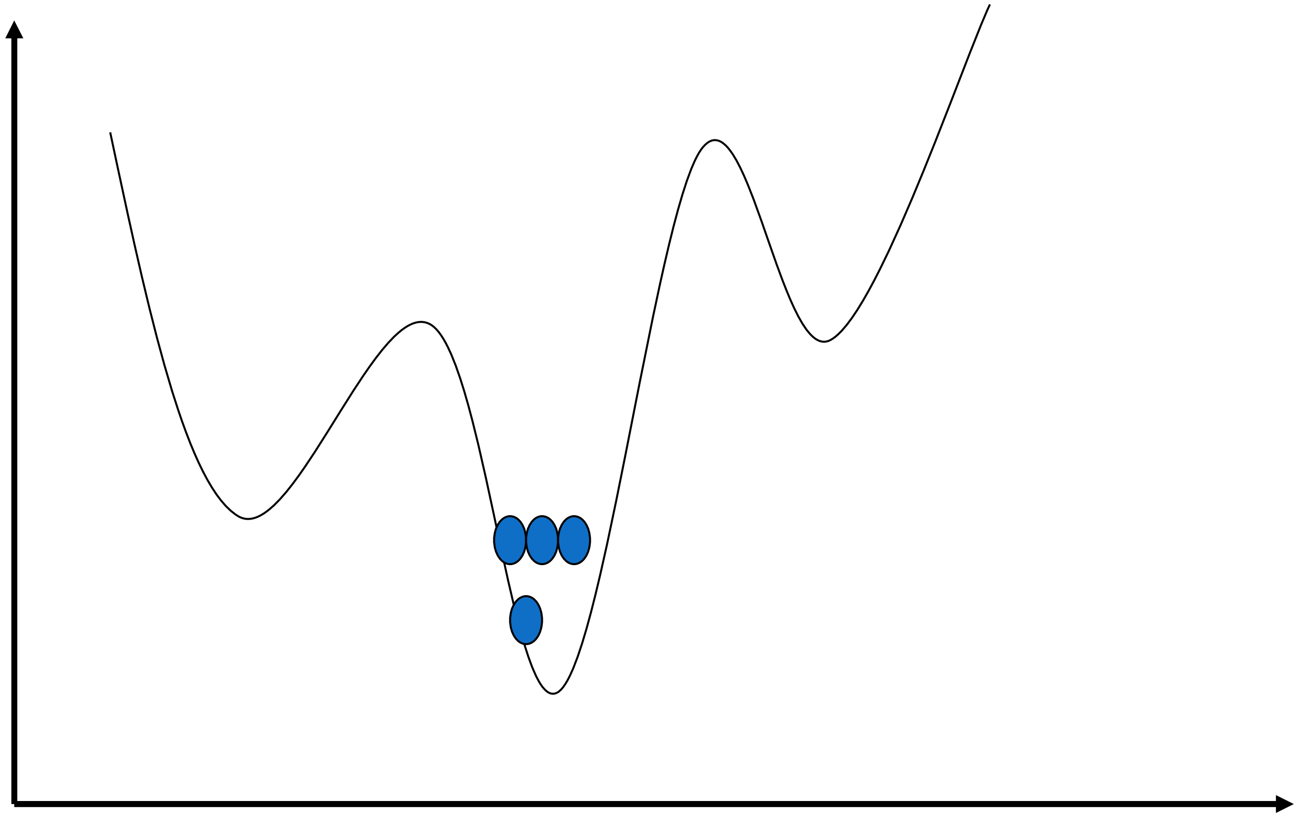
Local Beam Search



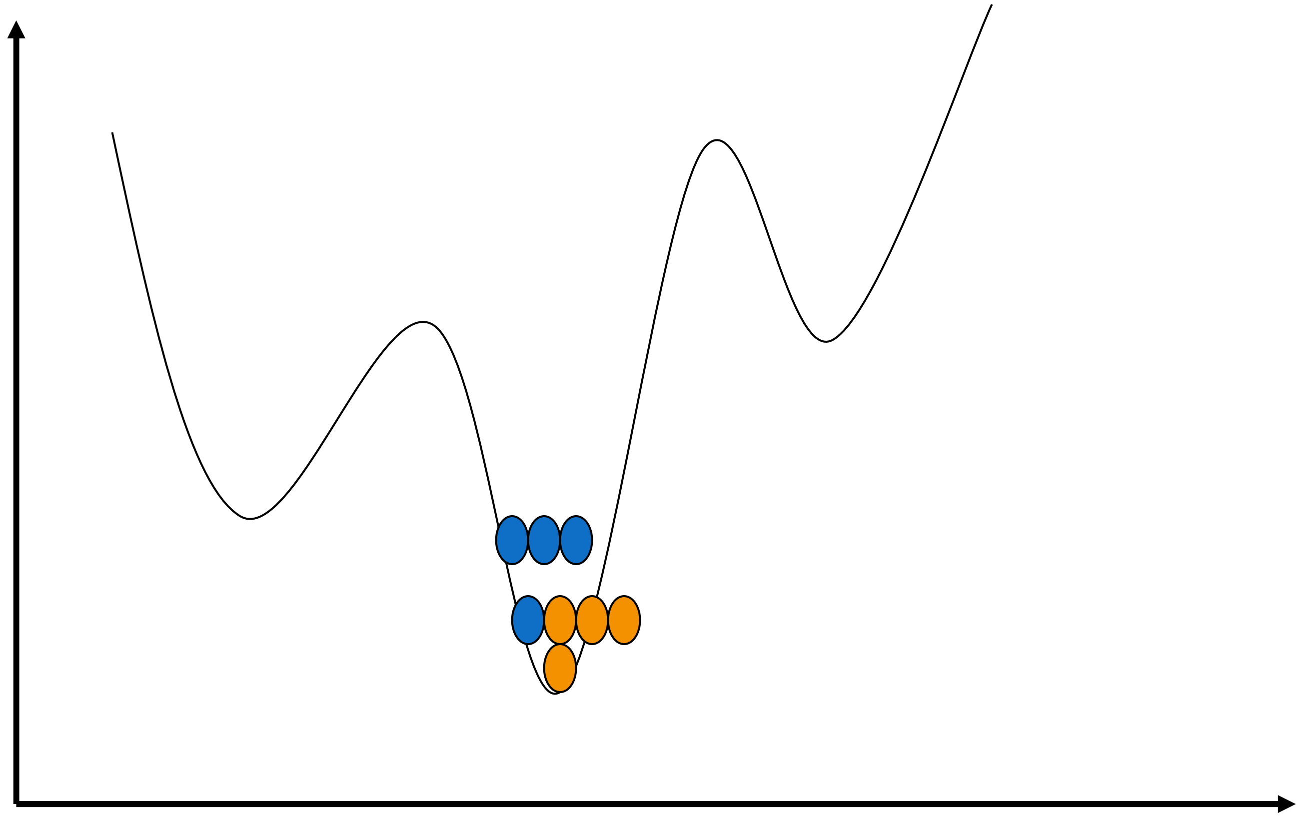
Local Beam Search



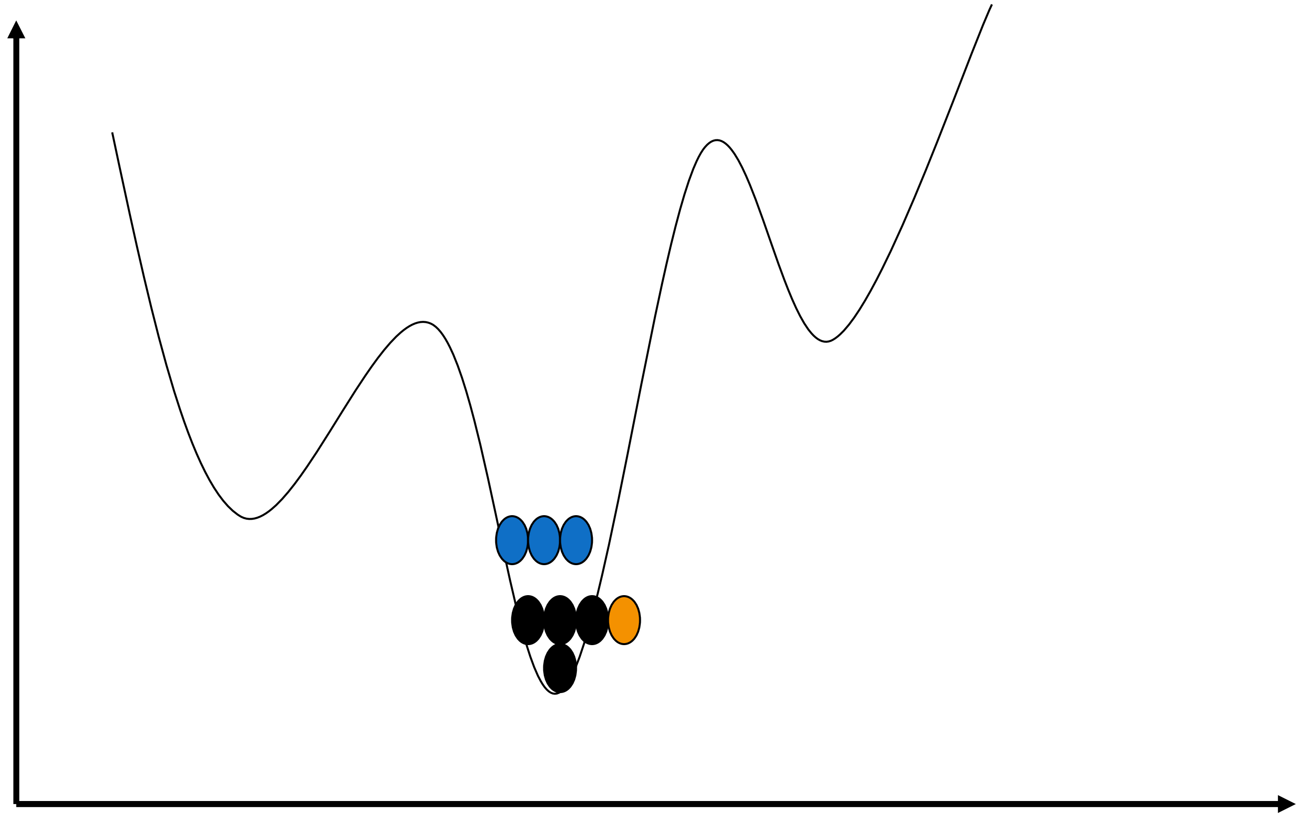
Local Beam Search



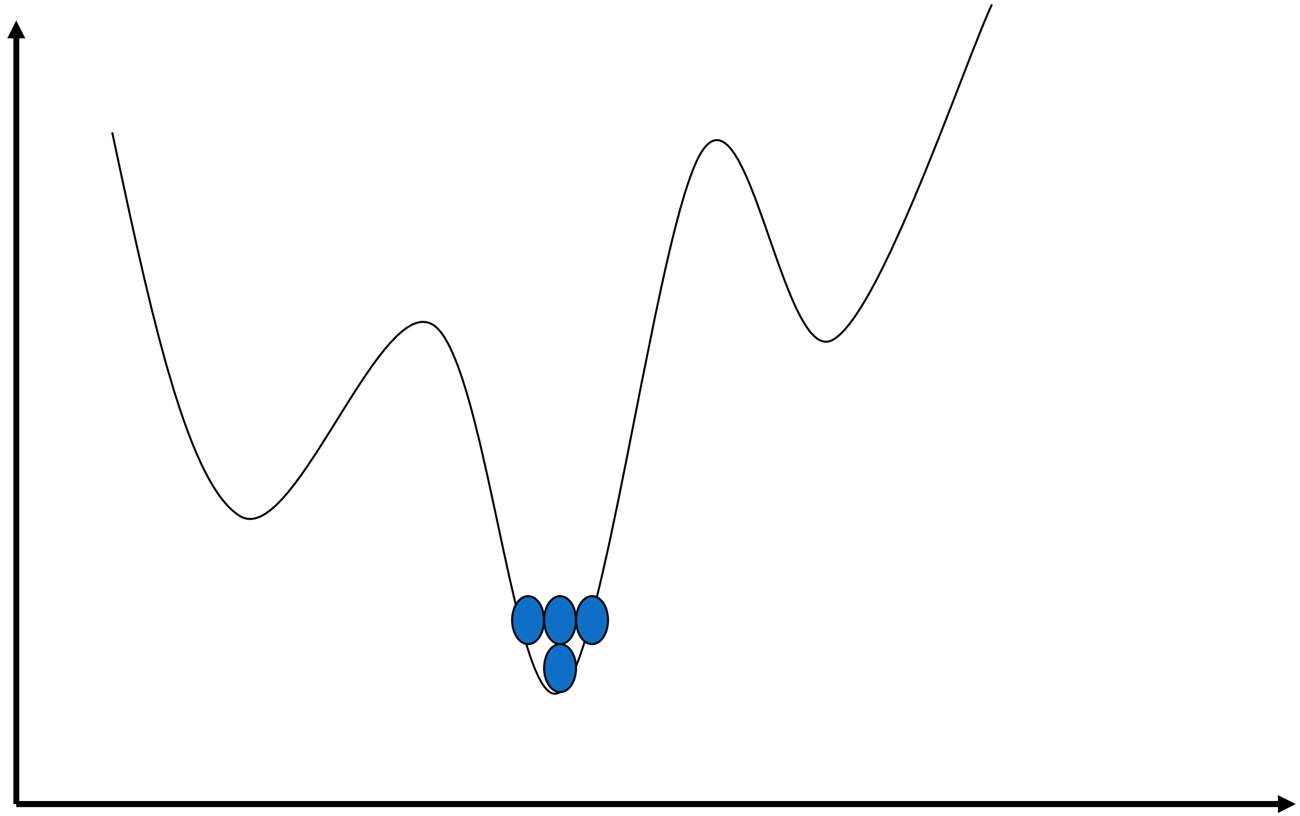
Local Beam Search



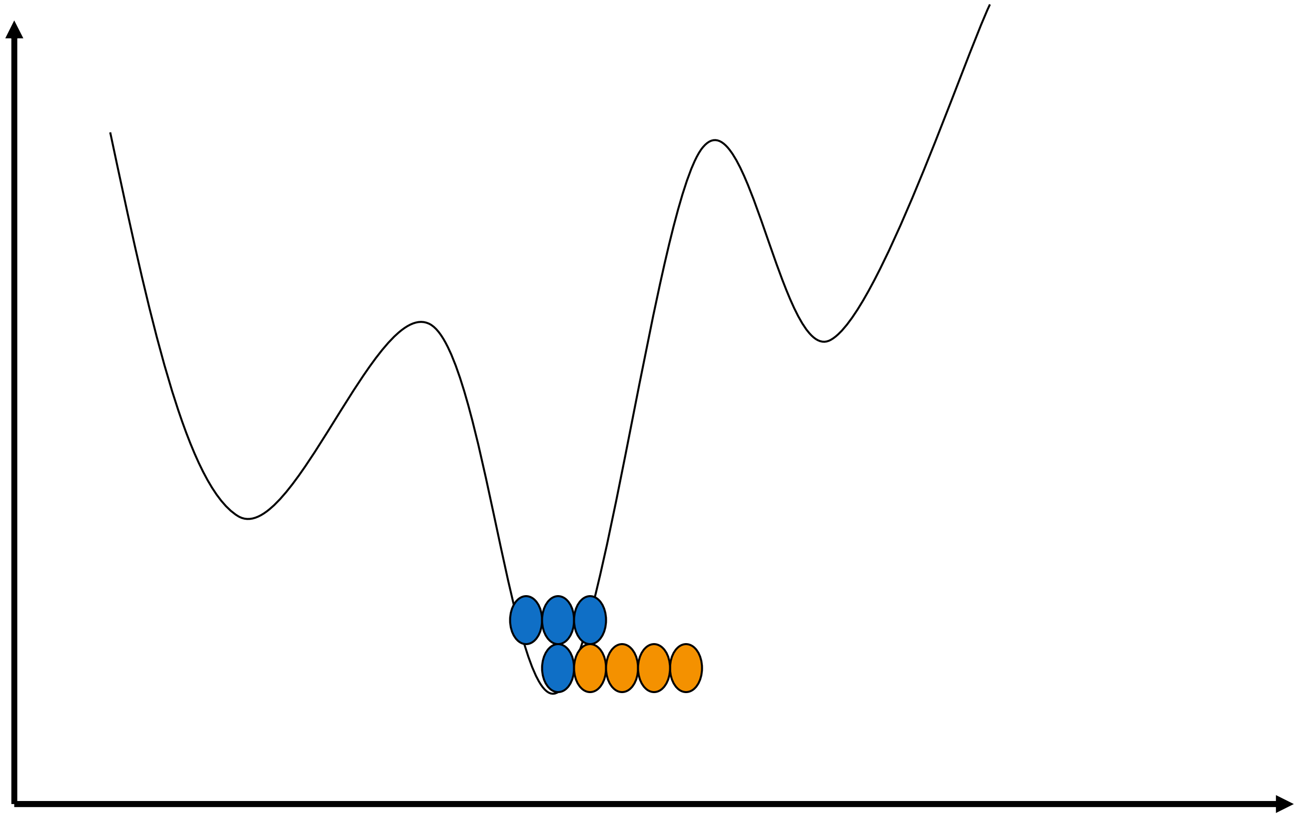
Local Beam Search



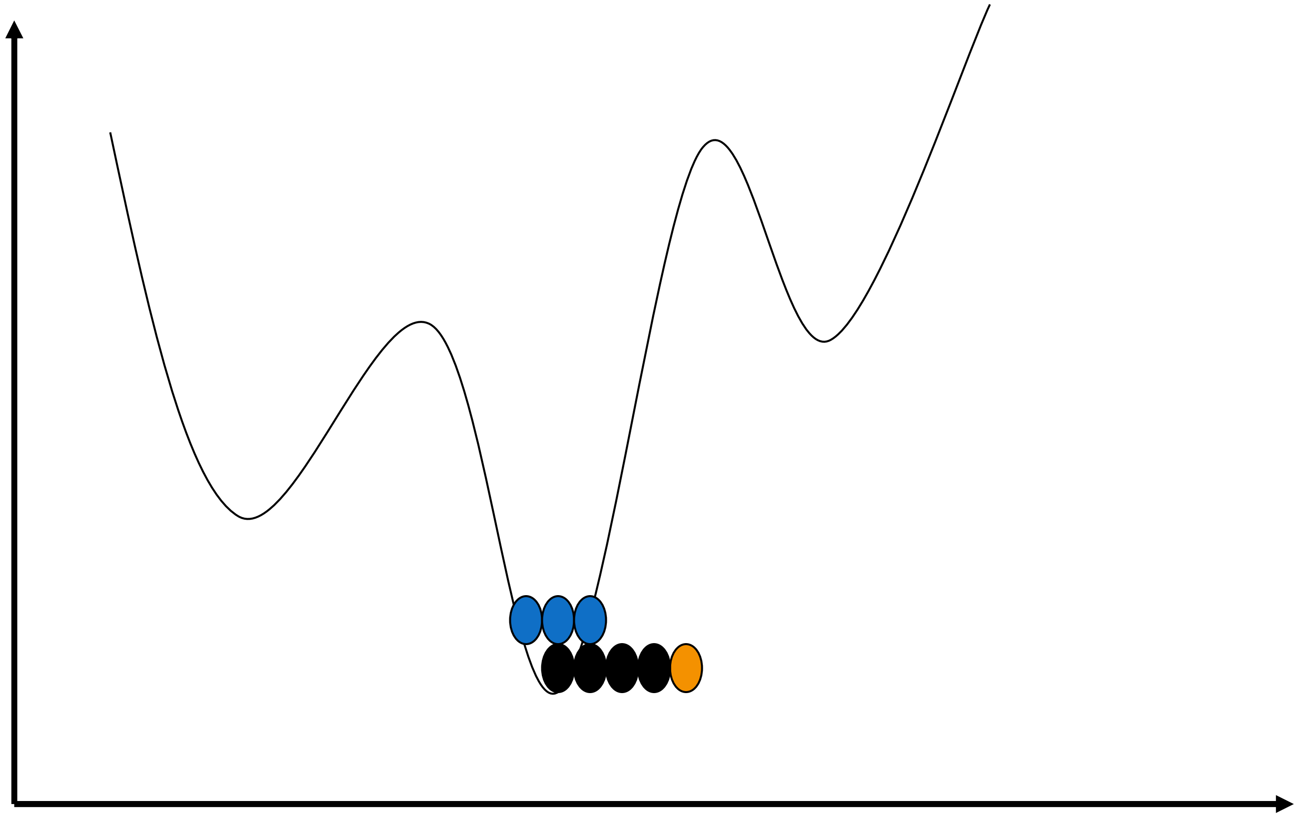
Local Beam Search



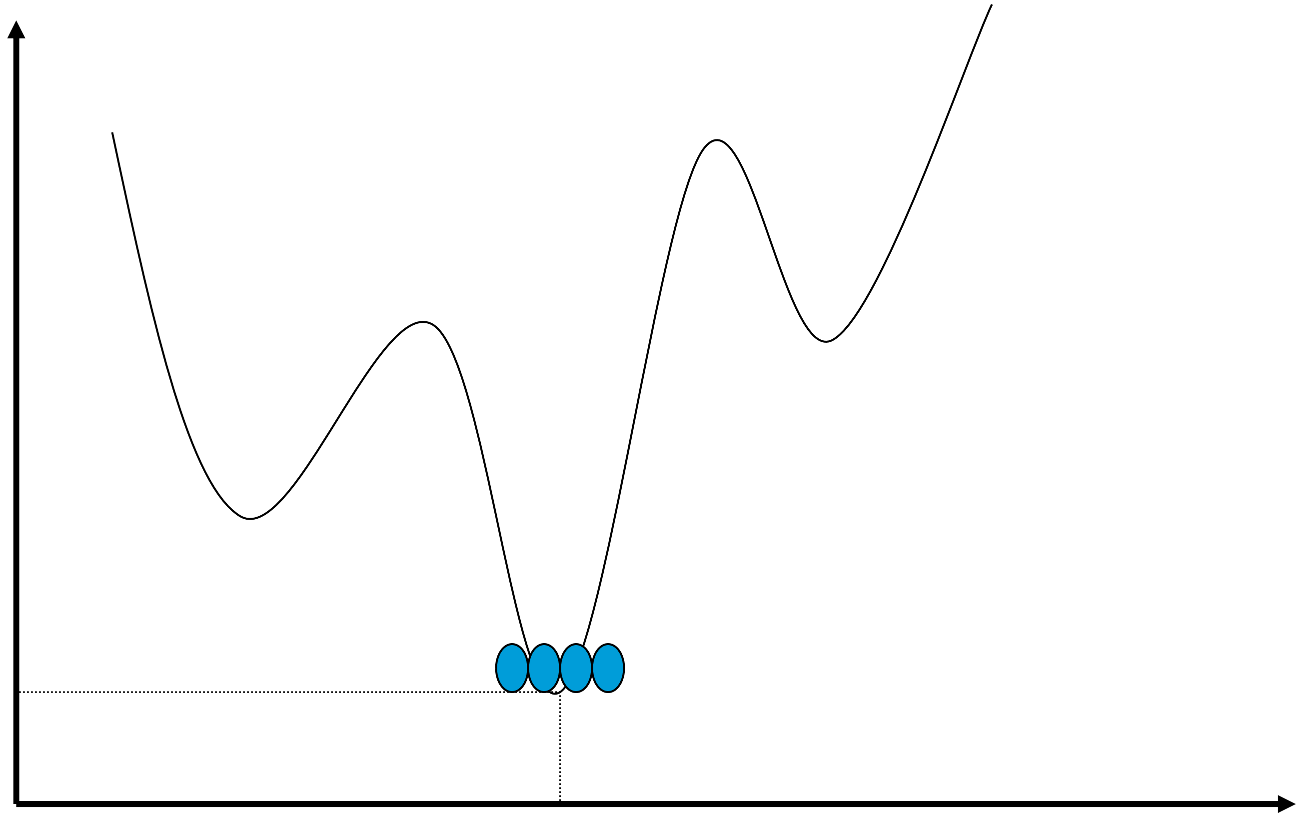
Local Beam Search



Local Beam Search



Local Beam Search



Thank you

End of Chapter 4-1