Search Algo. (= 탐색) is the foundation of AI algorithm!

Search Algorithms: Object-Oriented Implementation (Part A)

In Machine Learning and Deep Learning problems, we perform optimization, and search techniques are used for this. From this lecture onward, we will study the theory of basic search algorithms and how to implement them using OOP.

Contents

- Conventional vs. AI Algorithms
- Local Search Algorithms
- Implementing Hill-Climbing Algorithms
- Defining 'Problem' Class
- Adding Gradient Descent
- Defining 'HillClimbing' Class
- Adding More Algorithms and Classes
- Experiments

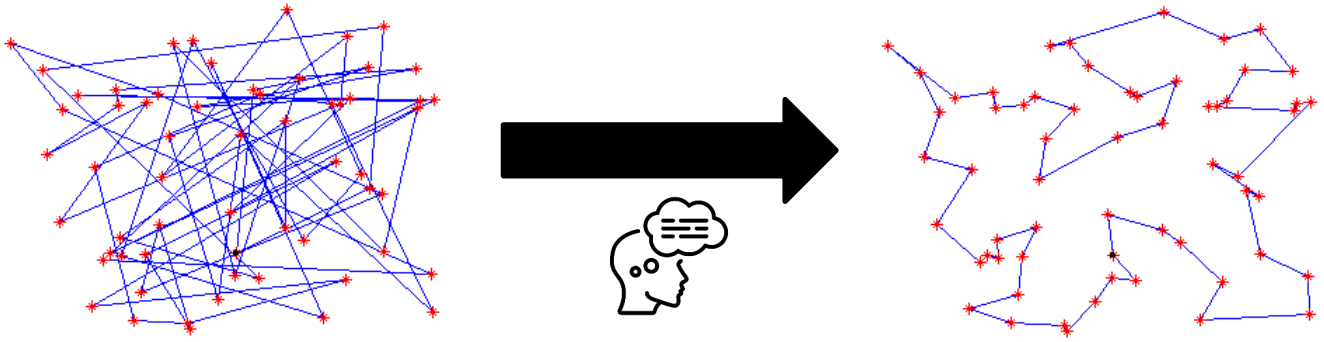
This Part A cover the following topics

Conventional vs. AI Algorithms

- Intractable problems (Problems that are difficult to handle in a mathematical sense...)
  - There are many optimization problems (최적화 문제) that require a lot of time to solve, but no efficient algorithms have been identified
  - Exponential (time complexity) algorithms are useless except for very small problems

Example. Traveling Salesperson Problem (외판원 문제)

The salesperson wants to minimize the total traveling cost (예 : 이동 시간 / 거리) required to visit all the cities in a territory, and return to the starting



Visits *n* cities randomly without a plan
➔ Travel path is longer (= higher cost)

Optimal travel path

Conventional vs. AI Algorithms

- Combinatorial Explosion: there are $n!$ routes to investigate

  → Efficiency is the issue! (= Time to calculate the optimal path and order among all routes is too long.)

  Time-efficient algorithms designed to find an approximate solution

  Nearest neighbor heuristic algo. : always goes to the nearest city

  – Given a starting city, there are $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2$ cases to consider at most

    Starting from a city, visit the nearest among the remaining n-1 cities, then the nearest of n-2, and so on….

  – Since there are n different ways to start, the total number of cases is $n \times n(n-1)/2$, which is much less than $n!$

  – One can find a near-optimal (최적 성능에 근접한) soln. in a much shorter time

- Conventional algorithms (e.g., sorting algorithms) are often called exact algorithms because they always find a correct or an optimal solution, but they can be inefficient in time
- AI algorithms use heuristics or randomized strategies to find a near optimal solution quickly

Local Search Algorithms

- Local Search Algorithm is a category of search methods.
  - Not by checking **all** cases at once, but by iteratively improving the solution through **local search** of the neighborhood.
- Iterative improvement algorithms: (Algorithms that iteratively improve the solution)
  - State space := set of "complete" configurations/solutions
    - A complete configuration : Arbitrary solution that satisfies the conditions.
      - ✓ An arbitrary solution is valid, but not necessarily optimal (best).
      - ✓ Ex: In TSP, any tour visiting all cities (satisfies conditions) is a complete configuration.
    - State space : set of complete configuration
      - ✓ Ex: in TSP, complete tour = state space

CAUTION! Search aims for the optimal solution among many/infinite complete configurations. A "solution" is just a valid state, not necessarily the most optimal one.

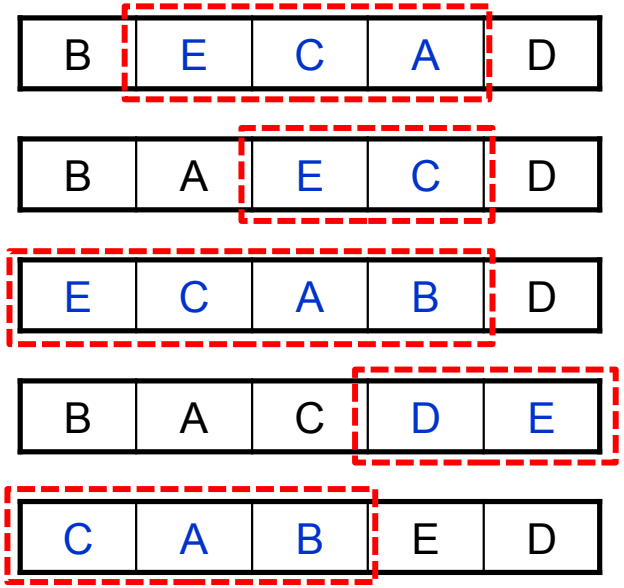
  - Among state space, find optimal configuration according to the objective function (목적 함수)
    - Ex: Optimal complete tour minimizing total path length (Objective/Goal function)?
  - Configurations should satisfy constraints
    - Constraints may exist (Ex: Must go via C to visit A).
  - Start with a complete configuration and make modifications to improve its quality
    - Local Search starts with an arbitrary config, selects a neighbor, and iteratively improves (to reduce complexity)

Local Search Algorithms

Example: TSP (Visiting 5 cities, A-B-C-D-E)

- [Step 1] Current configuration: (Select an arbitrary complete tour and calculate cost.)

- [Step 2] Find a slightly different (neighbor) configuration, and if valid, select the one with the minimum cost.
  - Candidate neighborhood configurations (of the **current** configuration):



Mutation by inversion: // One of the techniques for selecting candidates (neighbors)

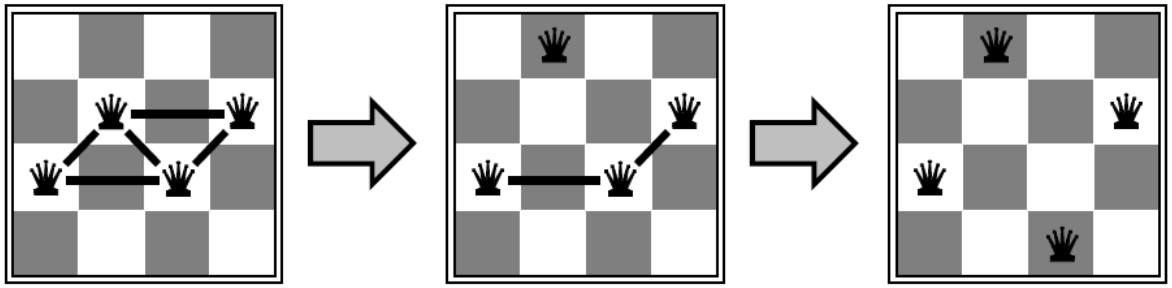
- A random subsequence is inverted (Select a random range and reverse the values within that range.)
- Select the best complete tour among candidates and repeat the inversion process to gradually find a better tour.

Performance:
Variants of this approach get within 1% of optimal very quickly with thousands of cities

Local Search Algorithms

Example: n-queens

- Move a queen each time to reduce number of conflicts (서로 공격 또는 대치하는 queen의 쌍을 최소화 하기위해, 퀸 의 위치를 하나씩 이동시키기)
  - Queens connected by a solid line are in an attacking position. (h = #lines = #conflicts)



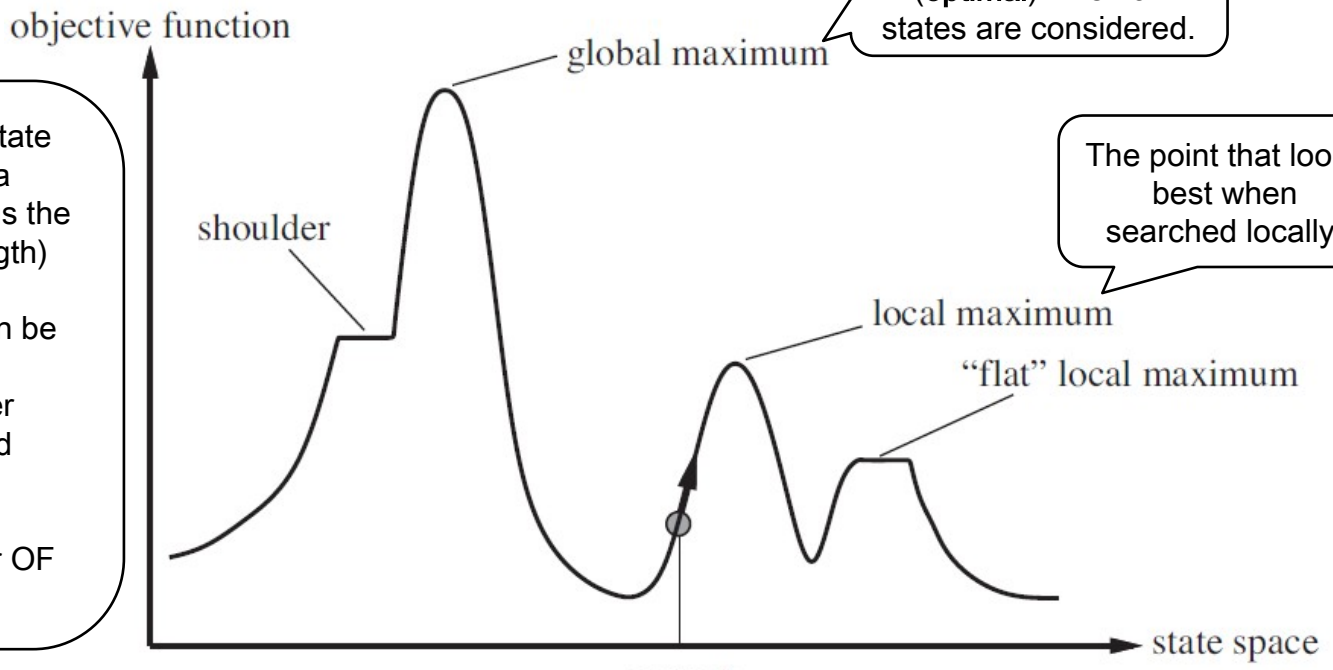
- Applying the Local Search algorithm allows us to gradually move the queens to minimize conflicts.
  - Almost always solves n-queens problems almost instantaneously (= Good time efficiency) for very large n, e.g., $n = 1\,million$

Local Search Algorithms

- State space landscape (The overall shape of the state space distribution)
  - x-axis: state (= configuration = solution)
  - y-axis: objective function (OF) or cost function
  - CAUTION! Local search generally reaches a local maximum, not the **optimal** global maximum, despite its best efforts.

CAUTION! A "solution" is just a valid state satisfying conditions. A problem can have many solutions, but finding the optimal one is key.

The best point (**optimal**) when all states are considered.



The point that looks best when searched locally.

- OF changes when the state changes (Ex: choosing a different TSP route), thus the OF value (total path length) changes.
- CAUTION! OF value can be better when high (Ex: Maximize profit) or better when low (Ex: Minimized TSP distance).
- The right figure is an example where a higher OF value is better.

current state = current configuration = the best solution found so far.

Hill-Climbing Search

- "Like climbing Everest in thick fog with amnesia"
  - (Fog) ➞ Cannot see far (i.e., doesn't consider all possibilities) and only inspects the vicinity (local search) ➞ Limits search scope, reducing computational complexity.
  - (Amnesia) ➞ Does not remember the path taken so far ➞ Reduces memory usage.
  - Continually moves in the increasing direction
    - A strategy of repeatedly moving in the direction of increasing altitude toward the mountain peak.
    - The strategy of slightly changing the configuration (solution) in a direction that improves the target value
      - ✓ Ex: The strategy of slightly adjusting a stock portfolio to maximize profit.
      - ✓ Ex: On the graph $y = -x\text{^}2$, the strategy of moving x slightly left or right to maximize y value.
      - ✓ Note: For some problems, it may be better to move in the direction of *lowering* the target value (Ex: Gra dually changing city order in TSP to minimize total path length)

Hill-Climbing Search

- "Like climbing Everest in thick fog with amnesia"
  - Also called gradient ascent/descent search (in case of continuous space)
    - Maximize value using ascent, minimize value using descent.
    - However, for minimization problems, you can solve it as a maximization problem by setting the goal value = goal value * -1

[Steepest ascent version]

Details on next page...

```
1: function HILL-CLIMBING( problem ) returns a state that is a local maximum
2:
3: current ← MAKE-NODE(problem.INITIAL-STATE)
4: loop do
5:     neighbor ← a highest-valued successor of current
6:     if neighbor.VALUE ≤ current.VALUE then return current.STATE
7:     else current ← neighbor
```

Hill-Climbing Search

- "Like climbing Everest in thick fog with amnesia"

[Steepest ascent version]

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

HILL-CLIMBING function that takes the problem as input (Ex: moving to the mountain peak) and returns the best result (**local maximum**)

3: current ← MAKE-NODE(*problem*.INITIAL-STATE)

Uses the initial state as input (Ex: location of the trail entrance) to create the *current* state (current position = an arbitrary solution)

Among algorithms that gradually improve a solution, one solution must be given to begin improvement, so an initial solution is randomly selected.

4: loop do

Repeats until the best result(local max) is found

5: *neighbor* ← a highest-valued successor of *current*

Selects the highest-valued state among the neighbors (*neighbor*, *successor*) of the *current* state (Ex: Select the highest elevation point among the spots immediately accessible from the trail entrance).

6: **if** *neighbor*.VALUE ≤ *current*.VALUE **then return** *current*.STATE

If the selected *neighbor* is not better than the current state (Ex: if elevation is lower), it assumes no better solution exists and returns the current state.

7: **else** *current* ← *neighbor*

If the *neighbor* is better than the current state (Ex: elevation is higher), move to that point, save it as *current*, and repeat the loop.

Hill-Climbing Search

Example: Finding optimal locations for 3 airports in Romania

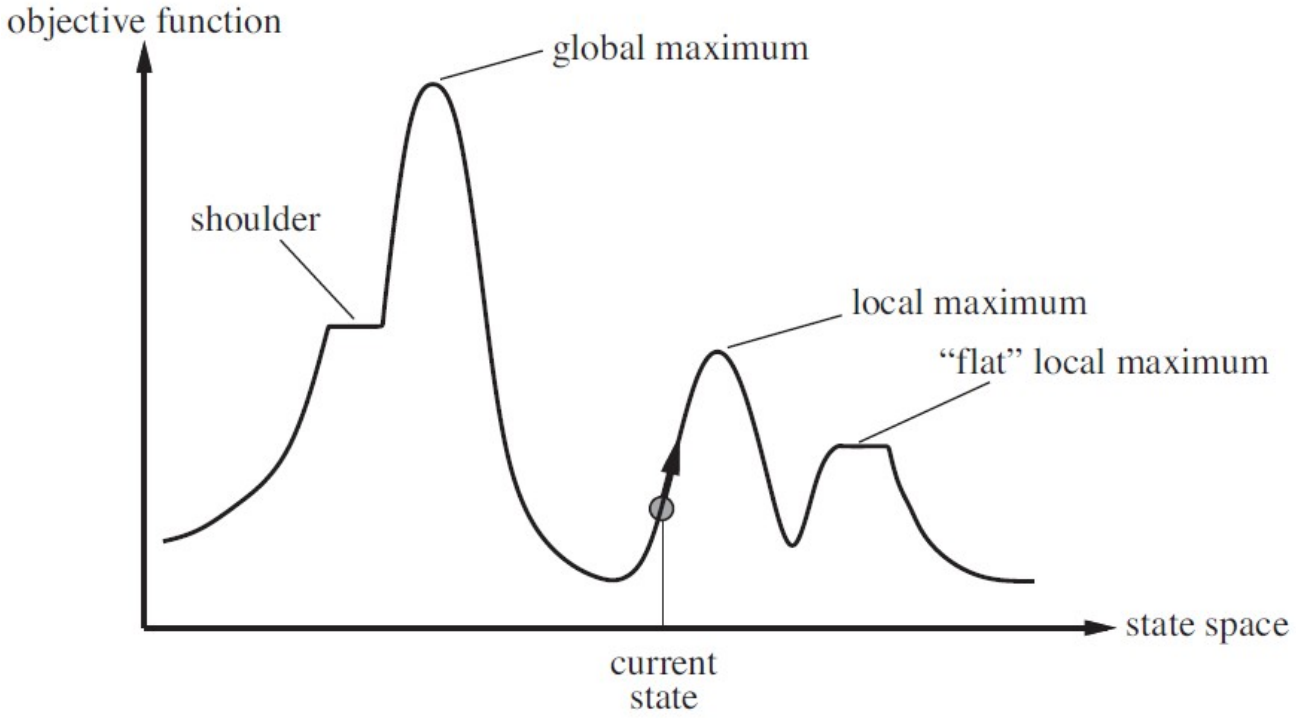
- state space: x,y coordinates of 3 candidate airport locations
  (x_1, y_1), (x_2, y_2), (x_3, y_3)

- Objective function:
  $f(x_1, y_1, x_2, y_2, x_3, y_3)$ = Sum of distances from all cities to the nearest airport ➔ Minimize f (minimization problem)

- Hill-Climbing Search Method
  - Randomly select initial locations for the 3 airports
  - Slightly move airport locations to select candidate areas (= find neighbor/adjacent solutions) ➔ Check if objective function increases or decreases ➔ Repeat process of moving to better airport location

Hill-Climbing Search

- Drawback: often gets stuck to local maxima due to greediness
  - Moves to the seemingly best state by only considering nearby neighboring points (= greedy approach)
  - Applying Hill-Climbing to the graph below, it immediately returns after reaching a local maximum (no better solutions nearby), failing to reach the actual optimal point (global maximum)



Hill-Climbing Search

- Drawback: often gets stuck to local maxima due to greediness
- Possible solutions (Ways to overcome the drawback of getting stuck at *local max*):
  - key idea: Don't necessarily choose a point just because it looks good immediately!
  - Stochastic hill climbing:
    - Chooses at random among the uphill moves with probability proportional to steepness
    - Assigns probability to each candidate state/solution proportional to its steepness (gradient), and selects one based on this probability.
  - First-choice (simple) hill climbing:
    - Generates successors randomly until one is found that is better than the current state
    - Selects candidate states randomly and chooses the first one that is better than the current state (simple calculation)
  - Random-restart hill climbing:
    - Conducts a series of hill-climbing searches from randomly generated initial states
    - Repeats hill-climbing multiple times by varying the starting point (initial state), and selects the best solution among them
  - Note: Even using the above techniques may not guaranteed to find the global maximum.

Hill-Climbing Search

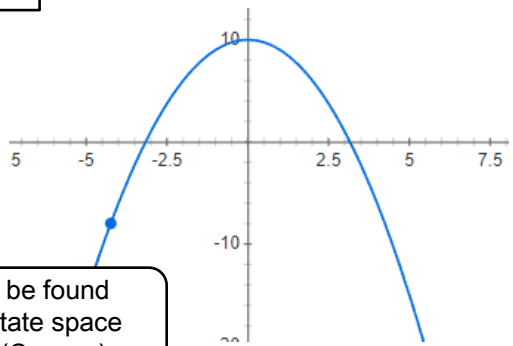
- Complexity (시간 복잡도 , time complexity):

Note: The concept of space complexity is also present

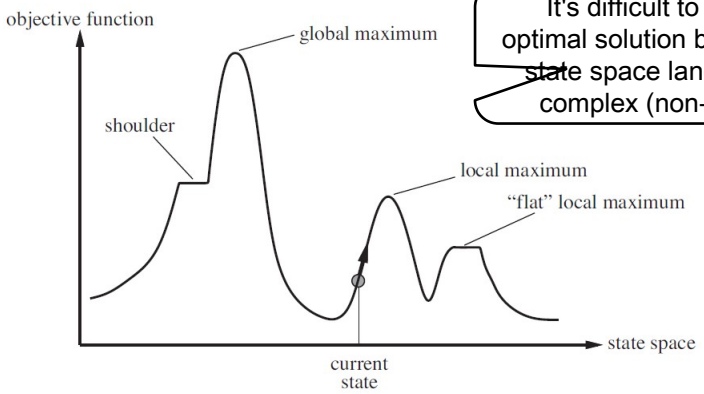
  - A standard for evaluating the time required to execute an algorithm (specifically, counting the number of basic operations needed to complete the algorithm)
  - The success of hill climbing depends on the shape of the state-space landscape
  - NP-hard problems typically have an exponential number of local maxima to get stuck on
  - A reasonably good local maximum can often be found after a small number of restarts with different starting point

Since the final result may vary depending on the randomly selected initial state, calculate optimal solutions for various starting points and select the best one among them

Reason why Local search techniques are widely used



Optimal solution can be found quickly because the state space landscape is simple (Convex)

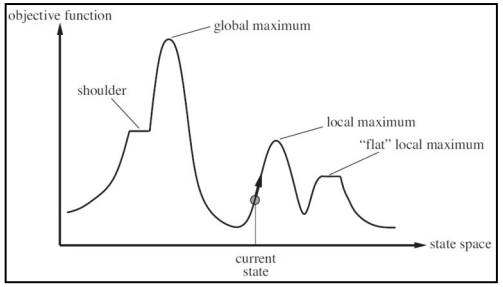


It's difficult to find the optimal solution because the state space landscape is complex (non-convex)

Convex vs Non-Convex

연속 상태 공간

Continuous State Spaces



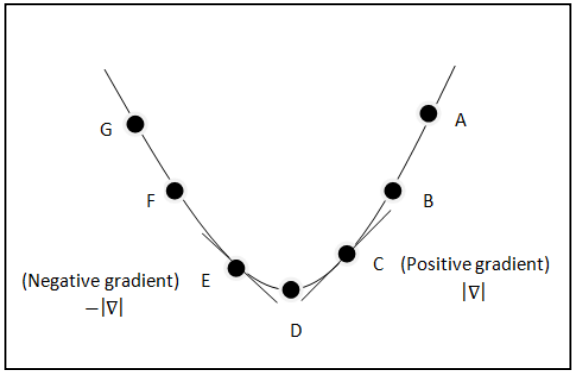
- Terminology Summary
  - Solution: A single answer (complete configuration) that solves the given problem
    - Ex: In a problem planning a route from Seoul visiting (Chuncheon/C, Daegu/D, Busan/B), one solution is Seoul > Busan > Chuncheon > Daegu
  - Optimal Solution: A solution that maximizes or minimizes the objective function
    - Ex: In the route planning problem above, the optimal solution minimizes the total travel distance (e.g., Seoul > Chuncheon > Daegu > Busan)
  - state space (SS): The set of all solutions that satisfy the given constraints
    - Ex: In the route planning problem above, the set of all possible combinations like S>C>D>B, S>D>B>C, S>B>C>D, ...
  - Optimal Solution (Cont.) // Refer to the top right figure
    - **Global** Optimal Solution: The best choice among all options within the SS set
    - **Local** Optimal Solution: The best among states adjacent to the current state; a state where no better-looking solution can be found by moving into a neighboring state
  - discrete state space : state space가 discrete (이산) 이다
    - The number of elements (= number of solutions) in the SS set can be counted
    - Ex: given x={1,2,3,4,5}, what is the solution that maximizes x^2? Number of elements in SS set: 5
    - Ex: with {서울(start),춘천,대구,부산}, what is the optimal route? Number of elements in SS set: 3*2*1
  - Continuous state space : state space 가 continuous (연속적) 이다
    - The number of elements in the SS set cannot be counted.
    - Ex: given range of [-3, 3] what is the solution that minimizes x^2? The number of selectable x values

Continuous State Spaces & Gradient Method

- Gradient (미분, 기울기) method
  - Given an objective function f, gradient (∇f) indicates the direction that increases the objective function

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  - The Gradient method incrementally updates the state/solution $(= x)$ in the direction that increases the objective function f, using the gradient
    - If the Gradient = 0, it is assumed the optimal solution has been found, as there is no longer a direction to increase the objective function (Note: The optimal solution found may not always be the global optimal solution)
    - It is an effective technique for finding the x that maximizes the objective function f in a continuous state space



    - Comparison: Hill-Climbing must select the best state among neighboring states based on the current state $(= x)$, while the gradient method uses the gradient to find the best neighboring state in one step.

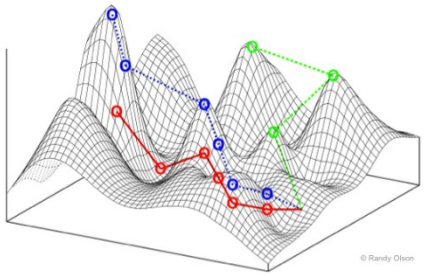
Continuous State Spaces & Gradient Method

- Gradient (미분 , 기울기) methods attempt to use the gradient of the landscape to maximize/minimize f by the following update rule:

$$\mathbf{x} \leftarrow \mathbf{x} + / - \alpha \nabla f(\mathbf{x}) \quad (\alpha : \text{update rate})$$

Rule for incrementally updating state x

where $\nabla f(\boldsymbol{x})$ is the gradient vector (containing all of the partial derivatives) of f that gives the magnitude and direction of the steepest slope

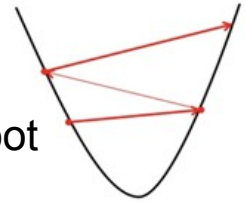


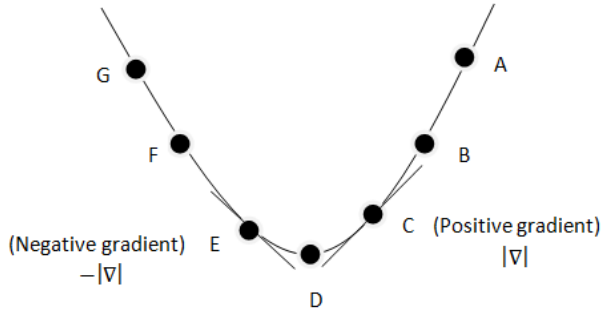
state space landscape example
x, y axis : state space,
Z axis: Objective function magnitude

- In this case, α (step=size) must be chosen carefully
  - Too small α : too many steps are needed
  - Too large α : the search could overshoot the target
  - Points where $\nabla f(\mathbf{x}) = 0$ are known as critical points

If reached, no better state can be found nearby

Overshoot





Continuous State Spaces & Gradient Method

Example: Gradient descent (Goal is to minimize the objective function)

- If $f(x) = x^2 + 1$, then $f'(x) = 2x$

  Starting from an initial value $w = 4$, with the step size of 0.1:

  - $4 - (0.1 \times 2 \times 4) = 3.2$
  - $3.2 - (0.1 \times 2 \times 3.2) = 2.56$
  - $2.56 - (0.1 \times 2 \times 2.56) = 2.048$
  - . . . . . .

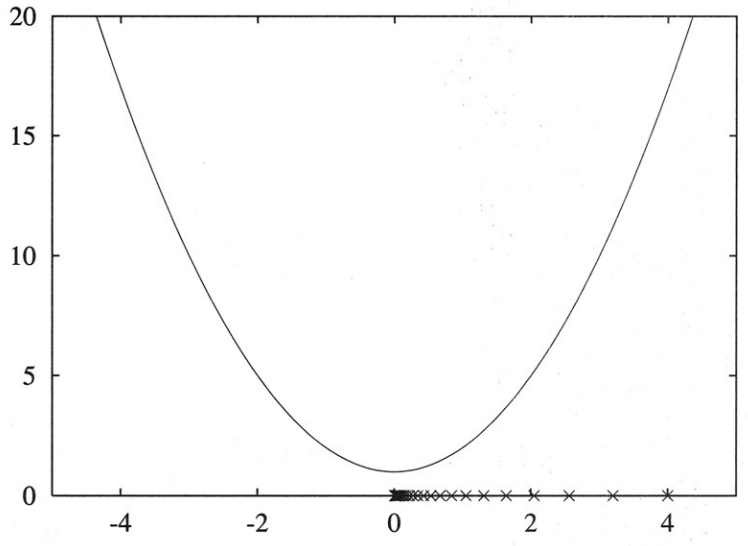
"To minimize", update the state in the negative (-) direction

$$x[t+1] = x[t] - \alpha f'(x[t])$$

state x update rule

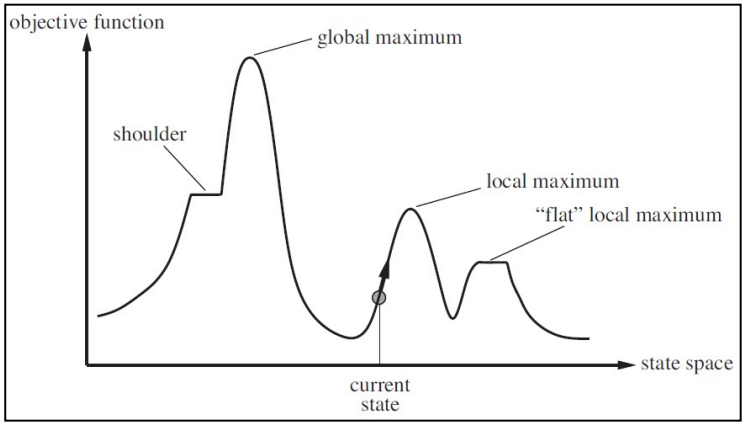
- Stops when the change in parameter value becomes too small
  - Ex: `stop if` $|x[t+1] - x[t]| \leq 1e-5$
  - or, `stop if` $|f'(x[t])| \leq 1e-5$

Because reaching the Critical point *exactly* is probabilistically almost impossible



Simulated Annealing Search

- Idea:
  - Hill-Climbing/Gradient method examines the surroundings and makes the best choice from them (i.e., makes a locally optimal choice)
    - HC : Selects the best state among neighboring states
    - Grad. : Calculates the derivative (gradient) based on the current position, and moves in that direction to select a new state
    - Both techniques stop when there is no better state
  - However, if the starting position is poor, it may reach and terminate at a local optimal solution.
  - Furthermore, always moving in the seemingly best direction after only checking the vicinity may be a poor decision



Simulated Annealing Search

- Idea:
  - Efficiency of valley-descending + completeness of random walk
    - That is, select a better next state in a time-efficient manner, but add a "random" selection process when choosing the next state
  - Escape local minima by allowing some “bad” moves. But gradually decrease their step size and frequency
    - Randomly allows bad moves, which helps to find better solutions
    - However, bad moves should be performed restrictively (i.e., gradually reduce the frequency of bad moves and the extent to which they change the state)

Simulated Annealing Search

- Origin of the Algorithm's Name (annealing: metal tempering/heat treatment)
  - Originates from the method of heating a metal material and then slowly cooling it (changing the temperature) to eliminate internal defects and encourage optimal physical properties (i.e., cool quickly at first, then slowly later to achieve stable properties!)
    - Devised by Metropolis *et al.*, 1953, for physical process modeling
    - Widely used in VLSI layout, airline scheduling, etc.
  - At fixed temperature T, state occupation probability reaches Boltzman distribution $p(x) = \alpha\, e^{-E(x)\,/\,kT}$
  - T decreased slowly enough → always reach the best state

Simulated Annealing Search

[Temperature change schedule] Passed as input, a schedule that changes the temperature from high to low (close to 0). It's stored as a list of what the temperature will be at each time-step.

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)

Calculate the current state from the initial state

for $t \leftarrow 1$ **to** ∞ **do**

$T \leftarrow schedule[t]$

if $T = 0$ **then return** *current*

next ← a randomly selected successor of *current*

Randomly select the next state

$\Delta E \leftarrow next$.VALUE – *current*.VALUE

Calculate the difference in objective function between the two states (= energy difference)

if $\Delta E < 0$ **then** *current* ← *next*

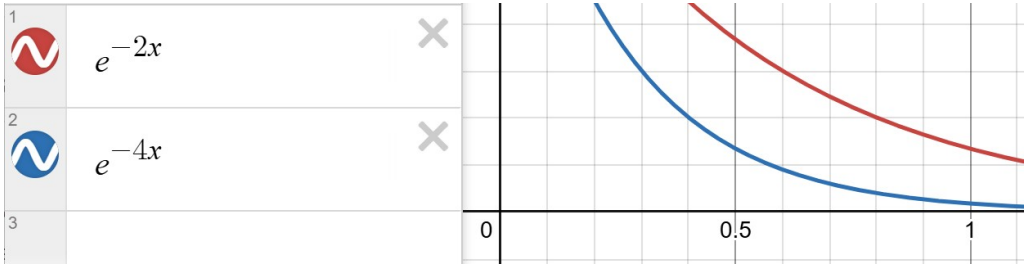
Assumes a minimization problem ➔ Lower value is better

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

Selects a bad state probabilistically. Here, the probability varies based on the difference between the two solutions and the temperature (T)

Simulated Annealing Search

- A random move is picked (with decreasing probability) instead of the best move
- If the move improves the situation, it is always accepted
- Otherwise, the move is accepted with probability $e^{-\Delta E / T}$
  - ΔE : the amount by which the evaluation is worsened
    - The acceptance probability decreases exponentially with the "badness" of the move
    - i.e., The worse the next state is, the lower the probability of selecting it.
  - T : temperature, determined by the annealing schedule (controls the randomness)
    - Bad moves are more likely at the start when T is high
      - ✓ When the initial T value is high, the probability of accepting a random bad move is high.
    - They become less likely as T decreases
      - ✓ Since it's common to set the schedule so that T decreases as the iteration progresses, the probability of selecting a bad move decreases with each iteration.
  - In short, random moves are highly permitted in the early stages, and restricted in the later stages



Simulated Annealing Search

- $T \to 0$: simple hill-climbing (first-choice hill-climbing)
  - If T approaches 0, the probability of accepting any move that worsens the solution ($\Delta E > 0$) drops to zero. The algorithm is then forced to only accept improving moves ($\Delta E < 0$), which is the defining characteristic of a greedy Hill-Climbing search.

- If the annealing schedule reduces T slowly enough, a global optimum will be found with probability approaching 1
  - In practice, following this theoretically slow schedule would require an **impractically large number of iterations** (often exponential time)!

- The initial temperature is often heuristically set to a value so that the probability of accepting bad moves is 0.5
  - This high initial probability ensures the algorithm is highly exploratory at the start.