2. Single State Methods

Dr. Hendri Sutrisno

Institute of Statistical Science

Academia Sinica, Taiwan

Dr. Chao-Lung Yang

Department of Industrial Management

National Taiwan University of Science and Technology

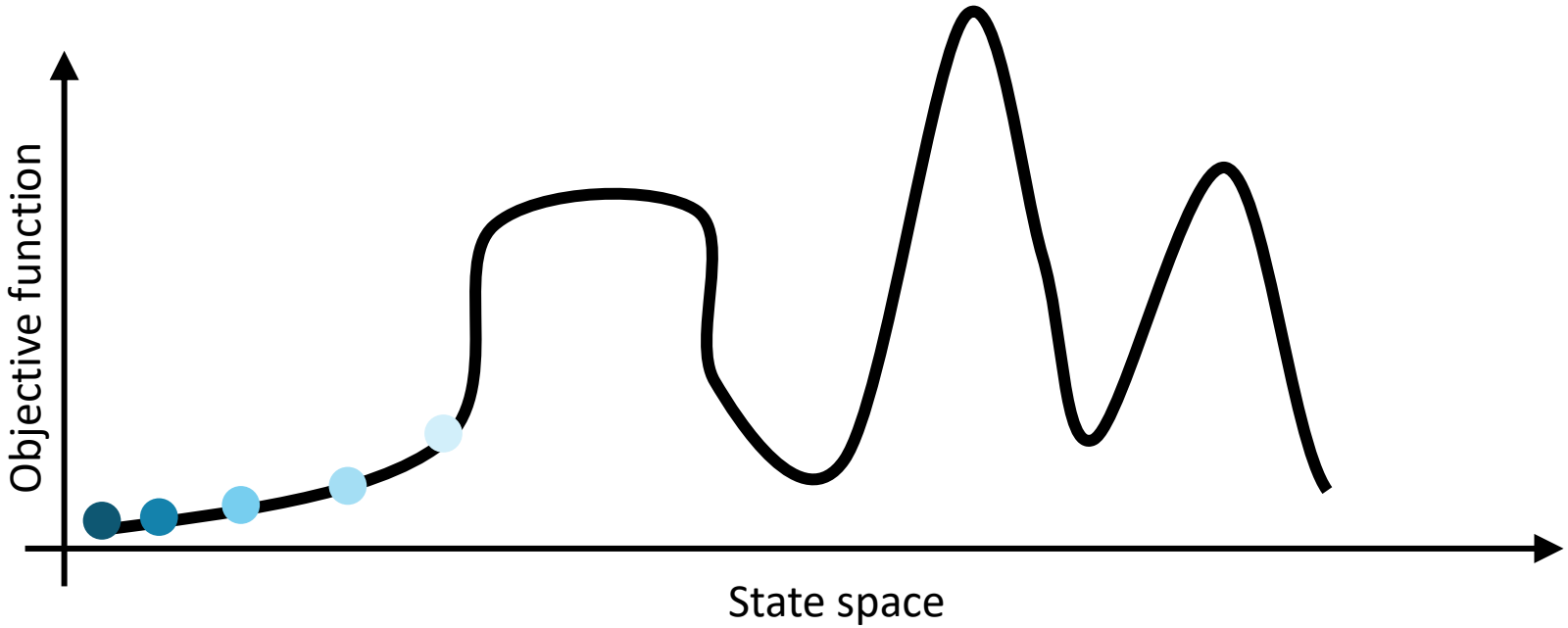
Brute Force, Random Search, and Hill Climbing

Optimization

- To optimize a candidate solution in this scenario, you need to be able to do four things:
  - Provide one or more initial candidate solutions. This is known as the initialization procedure.
  - Assess the Quality of a candidate solution. This is known as the assessment procedure.
  - Make a Copy of a candidate solution.
  - Tweak a candidate solution, which produces a randomly slightly different candidate solution. This, plus the Copy operation, are collectively known as the modification procedure

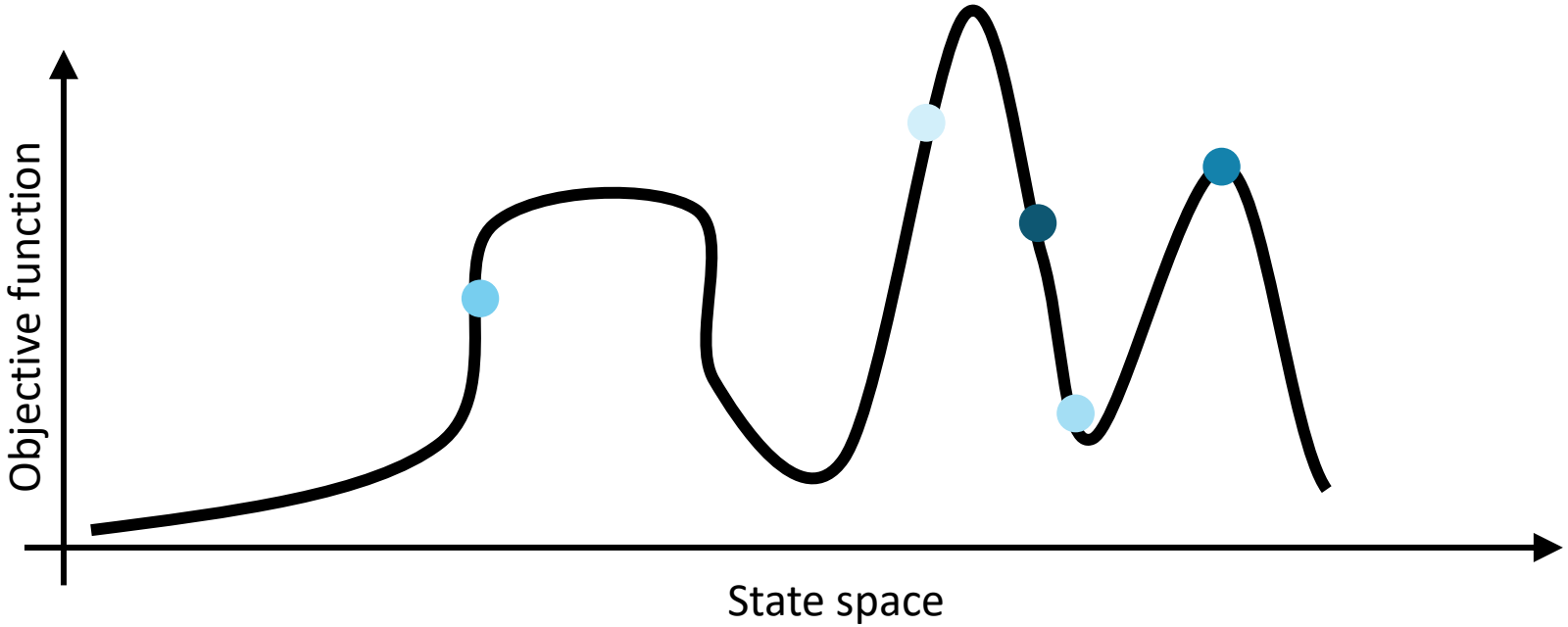
Brute Force: A Naïve approach

- Try every single feasible solution
- Guarantee the global optimum solution, but not computationally feasible



Random Search

- Try other feasible solution randomly
- May not achieve the global optimum solution, but computationally feasible



Simple Hill Climbing

- Hill Climbing (HC) Algorithm
  1. Start at a random point
  2. Evaluate the neighboring solution
  3. If the neighboring solution is better, then move
  4. Repeat (Step 2-3) until no improvement is observed
- The searching performance is highly depended on the starting point
- Fast in computation, but the solution might be sub-optimal

Hill-Climbing

Algorithm 4 *Hill-Climbing*

1: $S \leftarrow$ some initial candidate solution ▷ The Initialization Procedure
2: **repeat**
3: $R \leftarrow \text{Tweak}(\text{Copy}(S))$ ▷ The Modification Procedure
4: **if** $\text{Quality}(R) > \text{Quality}(S)$ **then** ▷ The Assessment and Selection Procedures
5: $S \leftarrow R$
6: **until** S is the ideal solution or we have run out of time
7: **return** S

Steepest Ascent Hill-Climbing

- create n "tweaks" to a candidate solution all at one time, and then possibly adopt the best one

Algorithm 5 *Steepest Ascent Hill-Climbing*

```
1: n ← number of tweaks desired to sample the gradient
2: S ← some initial candidate solution
3: repeat
4:     R ← Tweak(Copy(S))
5:     for n − 1 times do
6:         W ← Tweak(Copy(S))
7:         if Quality(W) > Quality(R) then
8:             R ← W
9:     if Quality(R) > Quality(S) then
10:        S ← R
11: until S is the ideal solution or we have run out of time
12: return S
```

Steepest Ascent Hill-Climbing with Replacement

- keep the best-discovered-so-far solution stashed away

Algorithm 6 *Steepest Ascent Hill-Climbing With Replacement*

```
1: n ← number of tweaks desired to sample the gradient

2: S ← some initial candidate solution
3: Best ← S
4: repeat
5:     R ← Tweak(Copy(S))
6:     for n − 1 times do
7:         W ← Tweak(Copy(S))
8:         if Quality(W) > Quality(R) then
9:             R ← W
10:    S ← R
11:    if Quality(S) > Quality(Best) then
12:        Best ← S
13: until Best is the ideal solution or we have run out of time
14: return Best
```

2.1.1 The Meaning of Tweak

- The initialization, Copy, Tweak, and (to a lesser extent) fitness assessment functions collectively define the representation of your candidate solution
- Fixed-length vector of real-valued numbers

Algorithm 7 *Generate a Random Real-Valued Vector*

1: $min \leftarrow$ minimum desired vector element value
2: $max \leftarrow$ maximum desired vector element value

3: $\vec{v} \leftarrow$ a new vector $\langle v_1, v_2, ...v_l \rangle$
4: **for** i from 1 to l **do**
5: $\quad v_i \leftarrow$ random number chosen uniformly between min and max inclusive
6: **return** $\vec{v}$

Add Randomness

- To Tweak a vector we might (as one of many possibilities) add a small amount of random noise to each number

Algorithm 8 *Bounded Uniform Convolution*

$\vec{v} \leftarrow$ vector $\langle v_1, v_2, ...v_l \rangle$ to be convolved
$p \leftarrow$ probability of adding noise to an element in the vector ▷ Often $p = 1$
$r \leftarrow$ half-range of uniform noise
$min \leftarrow$ minimum desired vector element value
$max \leftarrow$ maximum desired vector element value

for i from 1 to l **do**
    if $p \geq$ random number chosen uniformly from 0.0 to 1.0 **then**
        repeat
            $n \leftarrow$ random number chosen uniformly from $-r$ to r inclusive
        until $min \leq v_i + n \leq max$
        $v_i \leftarrow v_i + n$
return $\vec{v}$

2.2 Single-State Global Optimization Algorithms

- Tweak: to “make a small, bounded, but random change”
- How to escape local optimum?
- Hill-Climbing is for exploitation (local)
- Random Search is for exploration (global)

Random Search

Algorithm 9 *Random Search*

1: $Best \leftarrow$ some initial random candidate solution
2: **repeat**
3: $S \leftarrow$ a random candidate solution
4: **if** Quality(S) $>$ Quality$(Best)$ **then**
5: $Best \leftarrow S$
6: **until** $Best$ is the ideal solution or we have run out of time
7: **return** $Best$

Hill Climbing on Various Search Spaces

- How Hill Climbing algorithm searches the global maximum solution in the following search spaces?

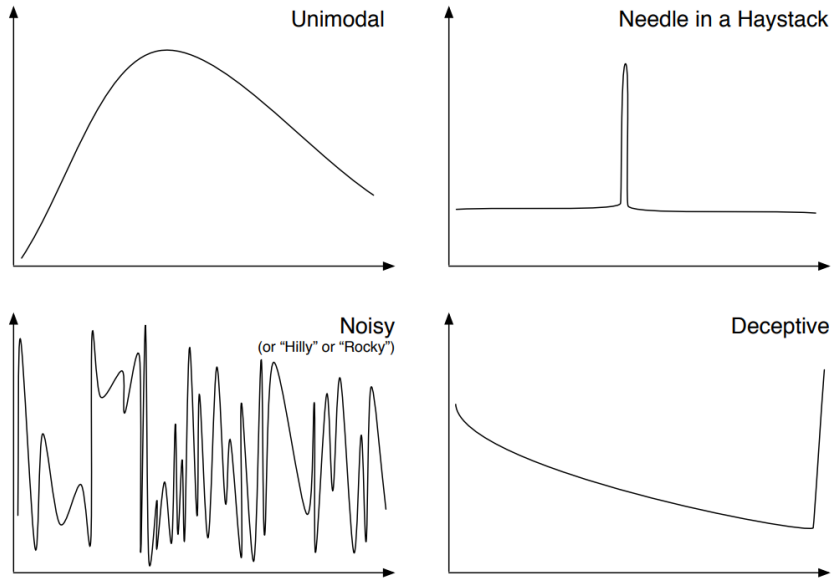


Figure 6 Four example quality functions.

Random Start Hill Climbing

- Algorithm
  1. Start at a random point
  2. Evaluate the neighboring solution
  3. If the neighboring solution is better, then move
  4. Repeat (Step 2-3) until no improvement is observed
  5. Repeat from step 1, stop when the number of restarts r is achieved.
- Higher potential to reach the global optimum solution than the Simple Hill Climbing Algorithm

Hill-Climbing with Random Restarts

Algorithm 10 *Hill-Climbing with Random Restarts*

```
1: T ← distribution of possible time intervals

2: S ← some initial random candidate solution
3: Best ← S
4: repeat
5:     time ← random time in the near future, chosen from T
6:     repeat
7:         R ← Tweak(Copy(S))
8:         if Quality(R) > Quality(S) then
9:             S ← R
10:    until S is the ideal solution, or time is up, or we have run out of total time
11:    if Quality(S) > Quality(Best) then
12:        Best ← S
13:    S ← some random candidate solution
14: until Best is the ideal solution or we have run out of total time
15: return Best
```

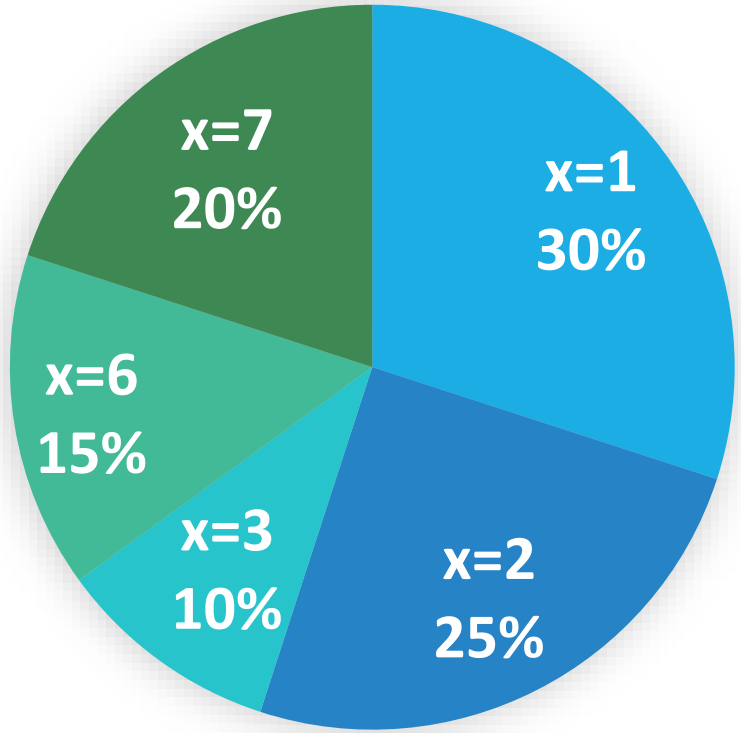
Other Hill Climbing Variants

- Steepest-Ascent Simple Hill Climbing
  - Evaluate all neighboring solutions within a searching window, select the best one
- Stochastic Hill Climbing (SHC)
  - Instead of selecting the best solution in the searching window (Steepest-Ascent), select the better one **randomly** or **based on a certain probability**.
- Hill Climbing with Replacement
  - Instead of selection the best solution in the searching window (Steepest-Ascent), replace the current solution with the last scanned solution, and store the best-found-solution

SHC: Random or probabilistic selection?

- Assume that we start from x=4
- We evaluate three neighborhood solutions (x=1 to x=7)

		Random	Probabilistic Selection		
x	F(x)	Pr	Improvement	∑ Improvement	Pr
1	10	1/5	3	10	3/10
2	9	1/5	2.5	10	2.5/10
3	8	1/5	1	10	1/10
4	7	0	0	10	0
5	6.5	0	-.5	10	0
6	8	1/5	1.5	10	1.5/10
7	8.5	1/5	2	10	2/10



Exploration & Exploitation in Hill Climbing

- Larger step size
  - Higher exploration ability
  - Faster to converge
  - Lower searching quality
- Smaller step size
  - Higher exploitation ability
  - Slower to converge
  - Higher searching quality

Some notes on Hill Climbing Algorithm

- Rely on the gradient information in the objective function (e.g. positive gradient for maximization, negative gradient for minimization)
  - It also means that HC is applicable for optimizing the supervised machine learning models, where the there is strong correlation between the target and features
  - The uglier the search space (weak or no correlation between the target and features), the more likely the performance deteriorated
- Easy and straightforward, but in terms of performance on complex problem, it might not be a good solution

Some notes on Hill Climbing Algorithm

- On unimodal problem
  - Pretty straight forward. An easy job for HC
- On the Needle in a Haystack problem
  - Most of the gradients look uninformative. Applying HC might not wise
- On the Noisy problem
  - Apparently in some noisy cases, Random Search is better than HC?
- On the Deceptive problem
  - If the HC could not find the right starting point, it will not reach the global optimum solution

Gaussian Convolution

Algorithm 11 *Gaussian Convolution*

$\vec{v} \leftarrow$ vector $\langle v_1, v_2, ...v_l \rangle$ to be convolved
$p \leftarrow$ probability of adding noise to an element in the vector ▷ Often $p = 1$
$\sigma^2 \leftarrow$ variance of Normal distribution to convolve with ▷ Normal = Gaussian
$min \leftarrow$ minimum desired vector element value
$max \leftarrow$ maximum desired vector element value

for i from 1 to l **do**
    if $p \geq$ random number chosen uniformly from 0.0 to 1.0 **then**
        repeat
            $n \leftarrow$ random number chosen from the Normal distribution $N(0, \sigma^2)$
        until $min \leq v_i + n \leq max$
        $v_i \leftarrow v_i + n$
return $\vec{v}$

Sample vs Noise in Tweak

		Noise in Tweak: Low	Noise in Tweak: High
Samples	Few		Explorative
	Many	Exploitative	

Table 1 Simplistic description of the interaction of two factors and their effect on exploration versus exploitation. The factors are: degree of noise in the Tweak operation; and the samples taken before adopting a new candidate solution.

Box-Muller-Marsaglia Polar Method

Algorithm 12 *Sample from the Gaussian Distribution (Box-Muller-Marsaglia Polar Method)*

$\mu \leftarrow$ desired mean of the Normal distribution ▷ Normal = Gaussian
$\sigma^2 \leftarrow$ desired variance of the Normal distribution

repeat
    $x \leftarrow$ random number chosen uniformly from -1.0 to 1.0
    $y \leftarrow$ random number chosen uniformly from -1.0 to 1.0 ▷ x and y should be independent
    $w \leftarrow x^2 + y^2$
until $0 < w < 1$ ▷ Else we could divide by zero or take the square root of a negative number!
$g \leftarrow \mu + x\sigma\sqrt{-2\frac{\ln w}{w}}$ ▷ It's σ, that is, $\sqrt{\sigma^2}$. Also, note that ln is $\log_e$
$h \leftarrow \mu + y\sigma\sqrt{-2\frac{\ln w}{w}}$ ▷ Likewise.
return g and h ▷ This method generates two random numbers at once. If you like, just use one.

Create a global search algorithm

- Adjust the Modification procedure
- Adjust the Selection procedure
- Jump to Something New
- Use a Large Sample

Simulated Annealing

Simulated Annealing: The very first work

- Simulated Annealing (SA) algorithm was formalized as an optimization method by Kirkpatrick S., et. al. in 1983
  - Kirkpatrick, S., Gelatt, C.D., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science, 220*, 671 - 680.
- Derived from the Metropolis-Hastings Algorithm proposed in 1953
  - Metropolis, N. et al. “Equation of state calculations by fast computing machines.” *Journal of Chemical Physics* 21 (1953): 1087-1092.
  - Main idea: Accept lower quality solution based on a certain probability

Inspiration from Metallurgy

- Annealing: A technique involving heating and controlled cooling of a material to alter its physical properties

Maxwell-Boltzmann distribution

$$f(v) = \sqrt{[\frac{m}{2\pi kT}]^3 4\pi v^2 e^{(-\frac{mv^2}{2kT})}}$$

$f(v)$: probability density as a function of the speed v

k : Boltzmann's constant

T : Temperature

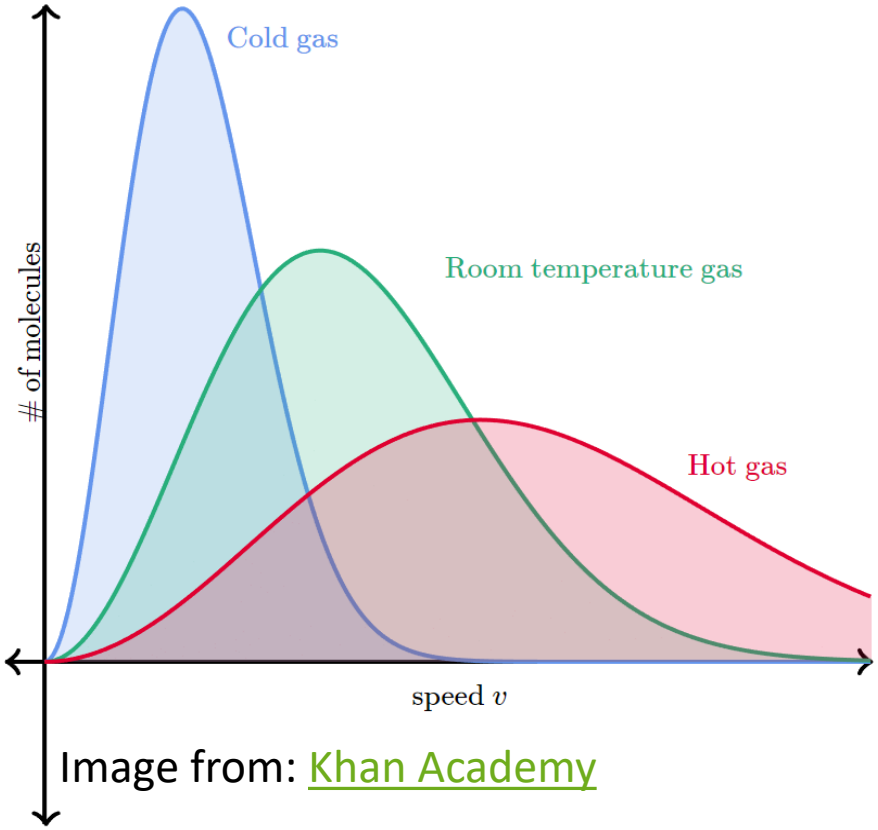


Image from: Khan Academy

The annealing process

- High temperature > more energy to move
- Low temperature > less energy to move

$T = \infty$

Freely to explore

$T \cong 0$

Only to exploit

Metropolis criterion

- Metropolis criterion: the probability of accepting newer solution R to replace the current solution S given temperature T
- Assume that the task is minimization
  - Let $\Delta = f(R) - f(s)$
  - $\Delta < 0$ means the newer solution is more fit
  - $$MC(t, \Delta) = \begin{cases} 1, & \Delta < 0 \\ e^{-\frac{\Delta}{T}}, & otherwise \end{cases}$$

The Pseudocode

- Given an initial solution S with objective value $f(S)$, and the initial temperature T
- For a finite set of iterations N:
  - Sample a new candidate solution R
  - Evaluate R to obtain $f(R)$
  - If $f(R) < f(S)$; Replace S with R
  - Else ;
    - Generate a random number p and calculate MC
    - If $p < MC$ then replace S with R; otherwise keep S
  - Decrease temperature T

Simulated Annealing

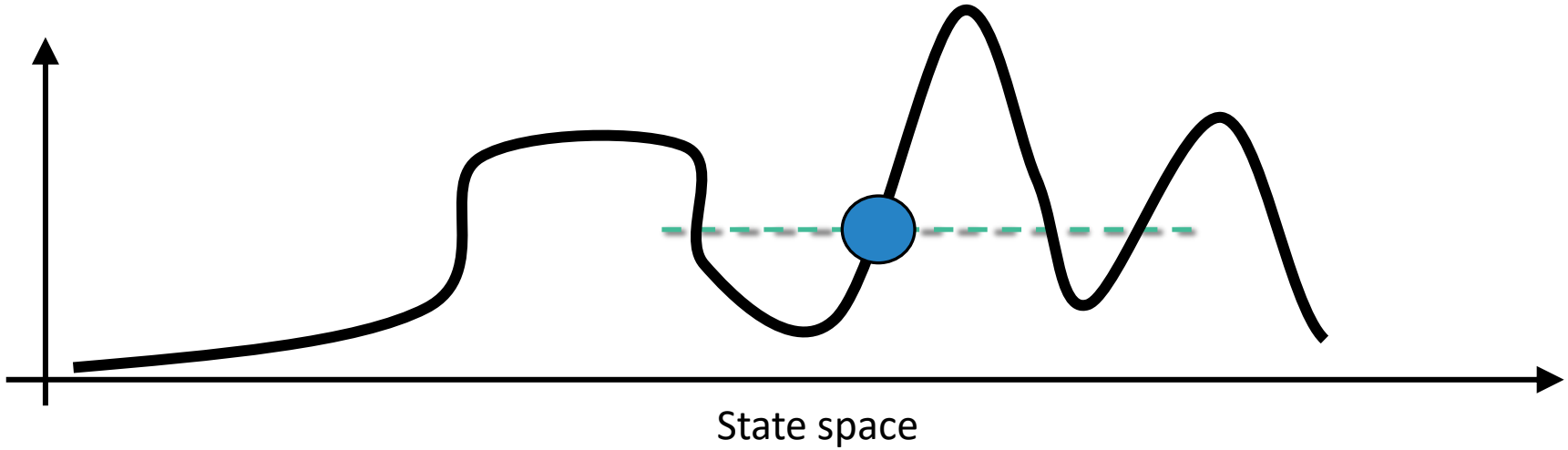
Algorithm 13 *Simulated Annealing*

1: $t \leftarrow$ temperature, initially a high number

2: $S \leftarrow$ some initial candidate solution
3: $Best \leftarrow S$
4: **repeat**
5: $\quad R \leftarrow \text{Tweak}(\text{Copy}(S))$
6: $\quad$ **if** $\text{Quality}(R) > \text{Quality}(S)$ or if a random number chosen from 0 to 1 $< e^{\frac{\text{Quality}(R)-\text{Quality}(S)}{t}}$ **then**
7: $\quad\quad S \leftarrow R$
8: $\quad$ Decrease t
9: $\quad$ **if** $\text{Quality}(S) > \text{Quality}(Best)$ **then**
10: $\quad\quad Best \leftarrow S$
11: **until** $Best$ is the ideal solution, we have run out of time, or $t \leq 0$
12: **return** $Best$

On sampling the candidate solution

- Assume that the solution follow the standard normal distribution
- $R = S + randn(len(S)) * step\_size$



- Reduce $step\_size$ over iterations

On updating temperature and $step\_size$

- The point is to reduce the controlling parameters along with the iterations

Linear

$r = 1 - \frac{i}{N}$

Step

$st = int(\quad) < N$

$if\ i\ mod\ \frac{N}{st} = 0$:

$r = r - \frac{N}{st}\%$

Cosine

- Convert i=[1,N]to [0,90] (radian)

$r = \cos(i)$

1 - Sin

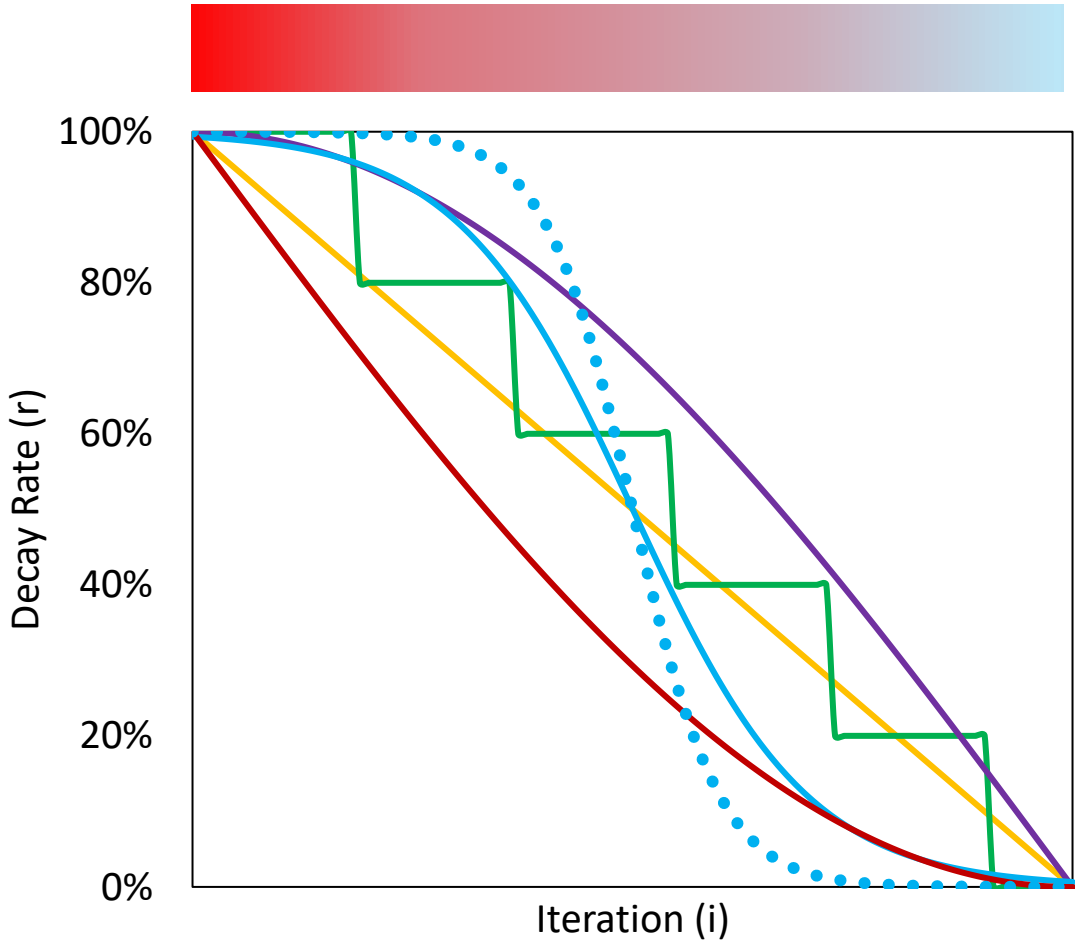
- Convert i=[1,N]to [0,90] (radian)

$r = 1 - sin(i)$

Sigmoid

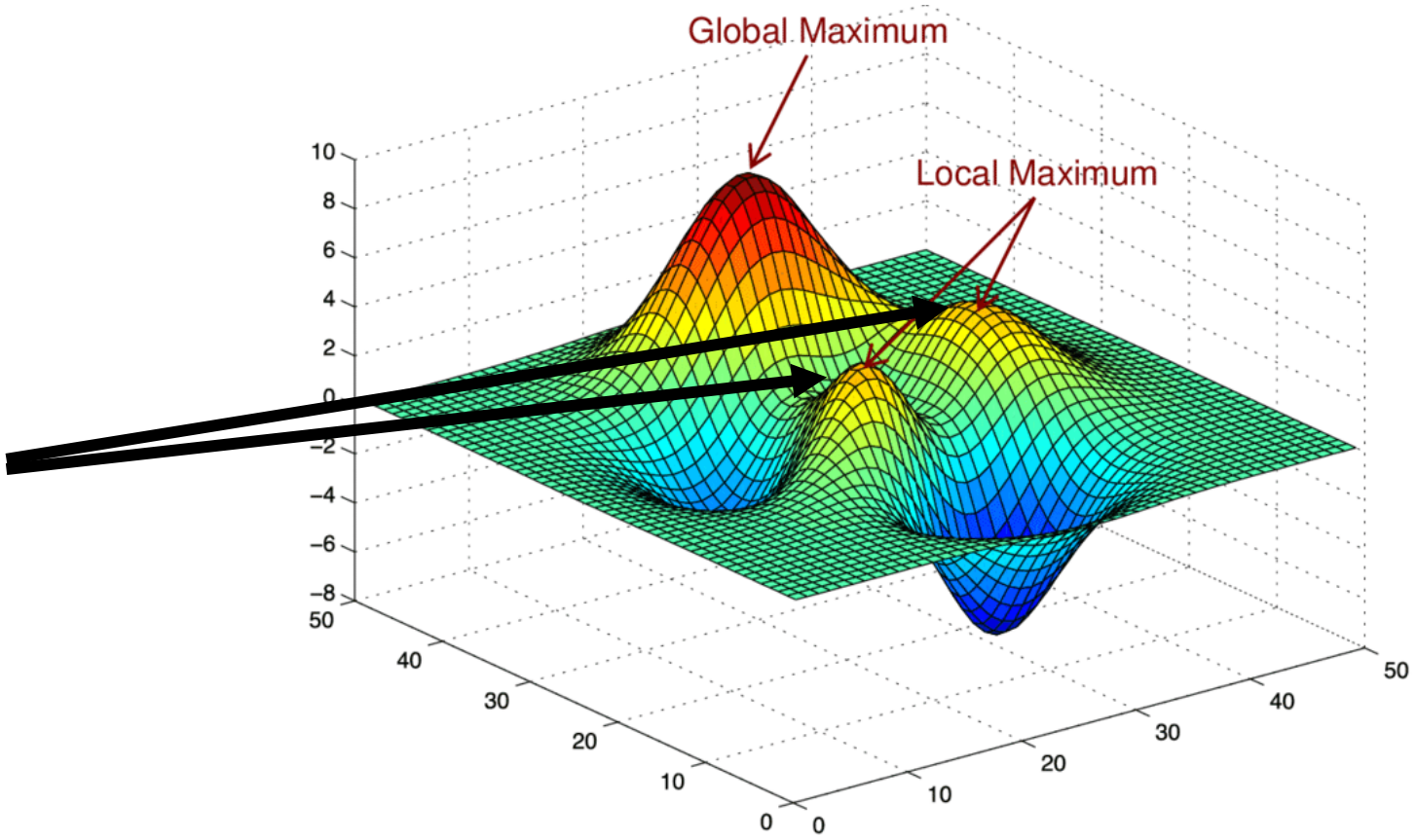
- Convert I [1,N] to some range, [-5,5], [-10,10]

$r = 1 - \frac{1}{1+e^i}$



Take away from SA

- Inspired from annealing process in metallurgy
- Unlike HS, SA allow the acceptance of lower quality solution with a hope to achieve better solution in the long run
- Depending on the problem, it is not necessary to have equal opportunity for exploration and exploitation. Sometimes prioritizing one of them might produce better result


Global Maximum
Local Maximum
10
8
6
4
2
0
−2
−4
−6
−8
50
40
30
20
10
0
0
10
20
30
40
50

Tabu Search

What is Tabu Search?

- Tabu Search (TS) was proposed in 1986 by Fred Glover
  - Glover, F., (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13(5)*, 533 - 549.
- Tabu (Tongan Language), Taboo (English)
- Like prohibiting from doing something “taboo”, TS forbid the searching process to revisit some of the previously visited solutions

Tabu List as the short-term memory

PROHIBITED ITEMS ON FLIGHTS

Image from: Airway Office

TABU LIST

- Lists a set of solutions that are not allowed to be visited again during the searching process

A very simple Tabu Search

- Given an initial solution S with objective value $f(S)$
- Let S^* be the best-found solution. In the beginning, $S^* = S$
- For a finite set of iterations N:
  - Sample a new candidate solution R
  - If R is listed in TabuList; Generate new R, $R \notin \text{TabuList}$; else Continue
  - Add R to TabuList
  - Evaluate R to obtain $f(R)$
  - Replace S with R
  - If $f(R) < f(S)$; update $S^* = R$
- Return S^*

Algorithm 14 *Tabu Search*

$l \leftarrow$ Desired maximum tabu list length
$n \leftarrow$ number of tweaks desired to sample the gradient

$S \leftarrow$ some initial candidate solution
$Best \leftarrow S$
$L \leftarrow \{\}$ a tabu list of maximum length l ▷ Implemented as first in, first-out queue
Enqueue S into L
repeat
    if Length$(L) > l$ **then**
        Remove oldest element from L
    $R \leftarrow$ Tweak(Copy(S))
    for $n - 1$ times **do**
        $W \leftarrow$ Tweak(Copy(S))
        if $W \notin L$ and (Quality$(W) >$ Quality(R) or $R \in L$) **then**
            $R \leftarrow W$
    if $R \notin L$ **then**
        $S \leftarrow R$
        Enqueue R into L
    if Quality$(S) >$ Quality$(Best)$ **then**
        $Best \leftarrow S$
until $Best$ is the ideal solution or we have run out of time
return $Best$

Algorithm 15 *Feature-based Tabu Search*

$l \leftarrow$ desired queue length
$n \leftarrow$ number of tweaks desired to sample the gradient

$S \leftarrow$ some initial candidate solution
$Best \leftarrow S$
$L \leftarrow \{\}$ ▷ L will hold tuples of the form $\langle X, d \rangle$ where X is a feature and d is a timestamp
$c \leftarrow 0$
repeat
    $c \leftarrow c + 1$
    Remove from L all tuples of the form $\langle X, d \rangle$ where $c - d > l$ ▷ The "old" ones
    $R \leftarrow$ Tweak(Copy(S), L) ▷ Tweak will not shift to a feature in L
    for $n - 1$ times **do**
        $W \leftarrow$ Tweak(Copy(S), L)
        if Quality(W) $>$ Quality(R) **then**
            $R \leftarrow W$
    $S \leftarrow R$
    for each feature X modified by Tweak to produce R from S **do**
        $L \leftarrow L \cup \{\langle X, c \rangle\}$
    if Quality(S) $>$ Quality($Best$) **then**
        $Best \leftarrow S$
until $Best$ is the ideal solution or we have run out of time
return $Best$

Some notes on TS

- How long the TabuList should be?
  - Shorter TabuList means less information stored
  - Longer TabuList means longer computation
- Can TS be applied on continuous problem?
- What if the search keep going back to a certain solution (trapped in local optimum)?
- What if a good “stepping stone” solution located in TabuList?

Teaching Note

- More techniques will be discussed in the next few lectures.
- Tabu Search is specialized for discrete optimization.
- We will have homework 1.