Simulated annealing

Review: Hill Climbing

```
hillClimbing(){
    t ← 0
    x ← generate an initial solution
    while (halt condition is not satisfied ) {
        Generate solution x' based on x
        Evaluate the f (x')
        If f(x')>f(x) {
            x ← x'
        }
        t ← t+1
    }
}
```

Hill-climbing algorithm

- Hill-climbing algorithm use the idea of greedy algorithm:
  - Each time it selects the best node in the neighborhood of the current node to compare
  - If the best neighbor is better than the current point, it will move to the neighbor. Otherwise, the process will be ended, and it will return to the current point.
- So, it has all the advantages and disadvantages of greedy algorithms:
  - Advantages: Simple and easy to implement
  - Disadvantages: depends on the initial point, in most cases, can only find **a local optimal solution**, and can not get the global optimal solution.

Hill-climbing algorithm: construction of neighbors

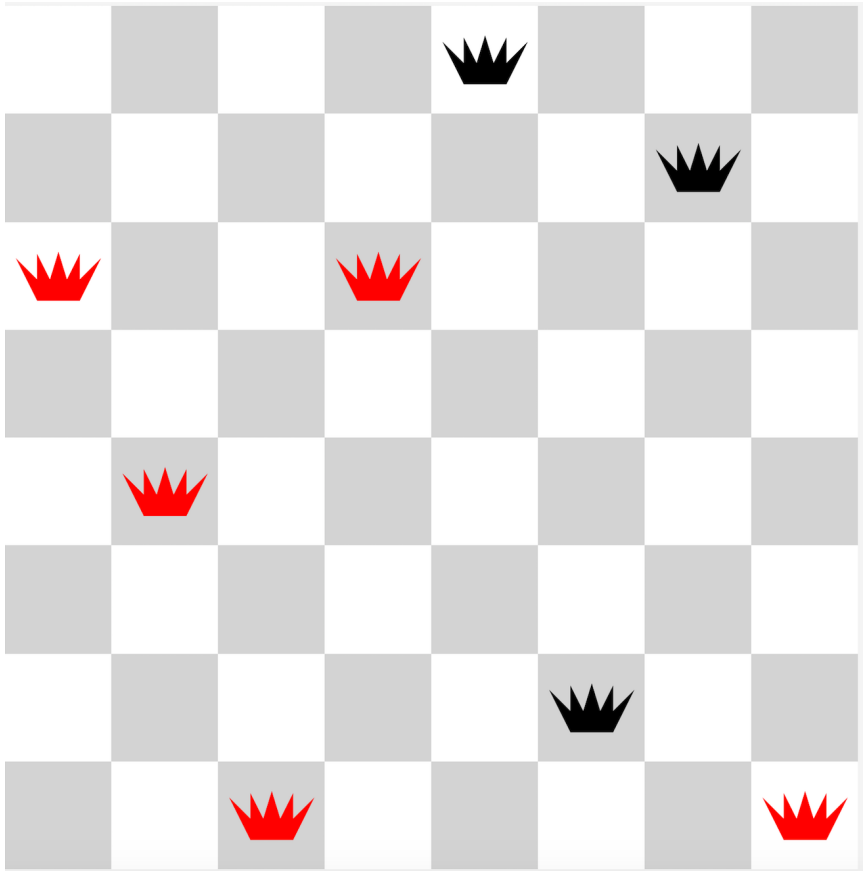
- Neighbor construction for eight-queen problem:

If the initial position is [5, 3, 0, 5, 7, 1, 6, 0], the neighbor can be constructed as the positions differing from the current position by a value on a single bit

1、[5,3,0,5,7,1,6,0]---→[0,3,0,5,7,1,6,0], [1,3,0,5,7,1,6,0]
[2,3,0,5,7,1,6,0], [3,3,0,5,7,1,6,0]
[4,3,0,5,7,1,6,0], [6,3,0,5,7,1,6,0]
[7,3,0,5,7,1,6,0]

2、[5,3,0,5,7,1,6,0]---→[5,0,0,5,7,1,6,0], [5,1,0,5,7,1,6,0]
[5,2,0,5,7,1,6,0], [5,4,0,5,7,1,6,0]
[5,5,0,5,7,1,6,0], [5,6,0,5,7,1,6,0]
[5,7,0,5,7,1,6,0]

And so on.



Hill-climbing algorithm: continuous optimization problem

- Neighbor construction for continuous optimization problem:

$$\max\ f(x_1, x_2, x_3, x_4, x_5) = \sum_{i=1}^{5} x_i^2\,, \qquad x_1 \in [0, 10]$$

- If the initial position is [0, 0, 0, 0, 0], the neighbor can be constructed by adding a gaussian noise to each variable.

$[0, 0, 0, 0, 0]$

$\rightarrow\ [0+\delta, 0, 0, 0, 0], [0, 0+\delta, 0, 0, 0], [0, 0, 0+\delta, 0, 0], [0, 0, 0, 0+\delta, 0], [0, 0, 0, 0, 0+\delta]$

Review: Simulated Annealing

- It is an **improvement** over Hill-climbing
  - From the initial state, every state is generated randomly from the neighborhood of the current state, and is **accepted with some probability**
  - The **acceptance probability** $\boldsymbol{P}$ only depends on the current state (x_i) and new state (x_i'), and is controlled by **temperature** $\boldsymbol{T}$:

$$p = \begin{cases} 1 & if \quad f(x_i) < f(x_i') \\ \exp(-\frac{f(x_i) - f(x_i')}{T}) & if \quad f(x_i) \geq f(x_i') \end{cases}$$

- Simulated annealing combines random search and greedy search

Simulated annealing VS Hill Climbing

```
hillClimbing(){
    t ← 0
    x ← generate an initial solution
    while (halt condition is not satisfied ) {
        Generate solution x' based on x
        Evaluate the f (x')
        If f(x')>f(x) {
            x ← x'
        }
        t ← t+1
    }
}
```

Replace this line with a probability p

$$p = \begin{cases} 1 & if \quad f(x_i) < f(x_i') \\ \exp(-\frac{f(x_i) - f(x_i')}{T}) & if \quad f(x_i) \geq f(x_i') \end{cases}$$

```
//one possible cooling schedule function for simulated   annealing
expSchedule(k, lam, limit, t){
        if (t < limit){
              return k*Math.exp(-lam*t)
        }
         return   0
}
```

Cooling schedule function

Each reduction rule reduces the temperature at a different rate and each method is better at optimizing a different type of model. For the 3rd rule, *beta* is an arbitrary constant.

1. **Linear Reduction Rule:** $t = t - \alpha$
2. **Geometric Reduction Rule:** $t = t * \alpha$
3. **Slow-Decrease Rule:** $t = \frac{t}{1+\beta t}$

```
simulatedAnnealing( schedule=expSchedule()){
    x ← generate an initial solution
    t ← 0
    while (halt condition is not satisfied ) {
          T = schedule(t)
          Generate solution x' based on x
           Evaluate the f (x')
           If (f(x')>f(x)) or (Math.exp((f(x')-f(x))/T)>random(0.0, 1.0)) {
                      x = x'
           }//end if
           update(t) // t = t+1
       }//end while
}//end simulatedAnnealing
```

Metropolis:

At high temperatures, a new state that is much worse than the current state can be accepted; at low temperatures, only a new state that is little bit worse than the current state can be accepted.

$$p = \begin{cases} 1 & if \quad f(x_i) < f(x_i') \\ \exp(-\frac{f(x_i) - f(x_i')}{T}) & if \quad f(x_i) \geq f(x_i') \end{cases}$$

Cooling schedule function: parameter controlling annealing process

```
schedule=exp_schedule(20,0.005,100)
for t in range(5):
    T = schedule(t)
    print(T)
```

```
20.0
19.900249583853647
19.800996674983363
19.70223879206125
19.603973466135105
```

Increase lam value

```
schedule=exp_schedule(20,0.1,100)
for t in range(5):
    T = schedule(t)
    print(T)
```

```
20.0
18.09674836071919
16.374615061559638
14.816364413634357
13.406400920712787
```

Decrease k value

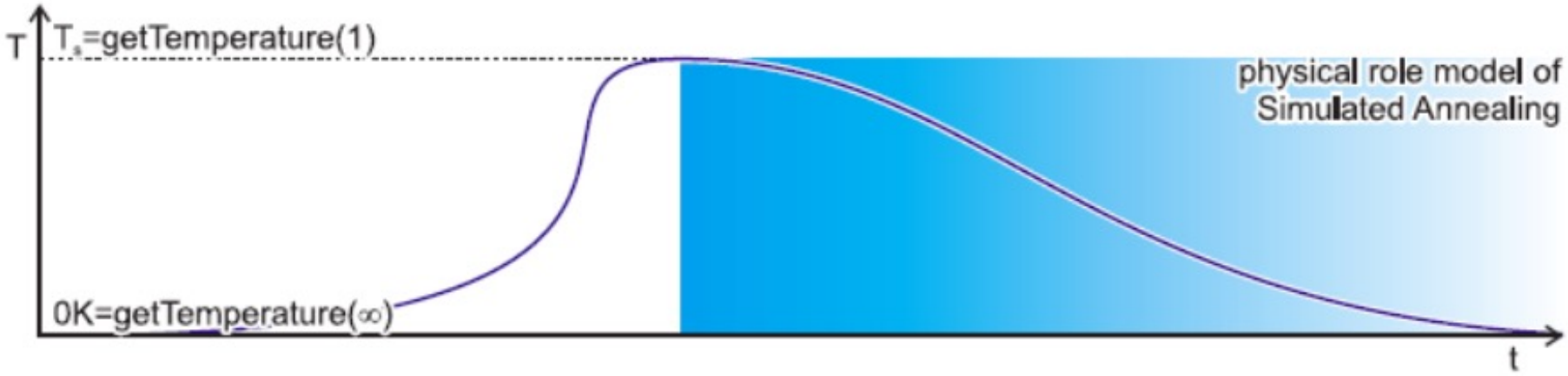
```
schedule=exp_schedule(10,0.1,100)
for t in range(5):
    T = schedule(t)
    print(T)
```

```
10.0
9.048374180359595
8.187307530779819
7.4081822068171785
6.703200460356394
```

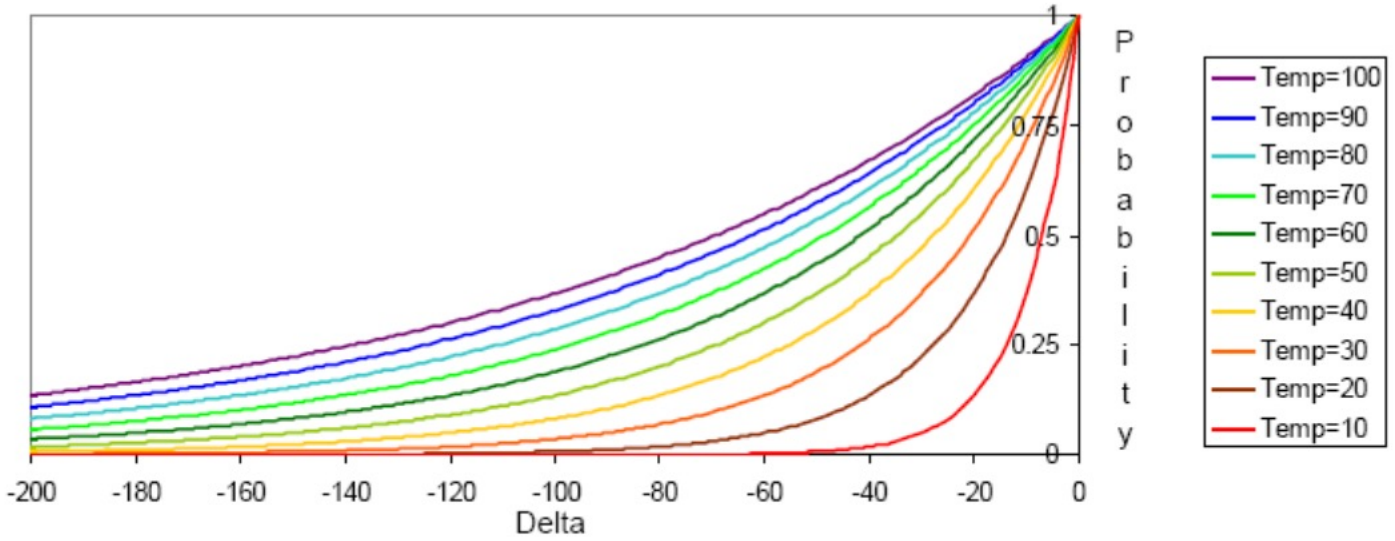
Decrease limit value

```
schedule=exp_schedule(10,0.1,5)
for t in range(6):
    T = schedule(t)
    print(T)
```

```
10.0
9.048374180359595
8.187307530779819
7.4081822068171785
6.703200460356394
0
```



Acceptance criterion and cooling schedule



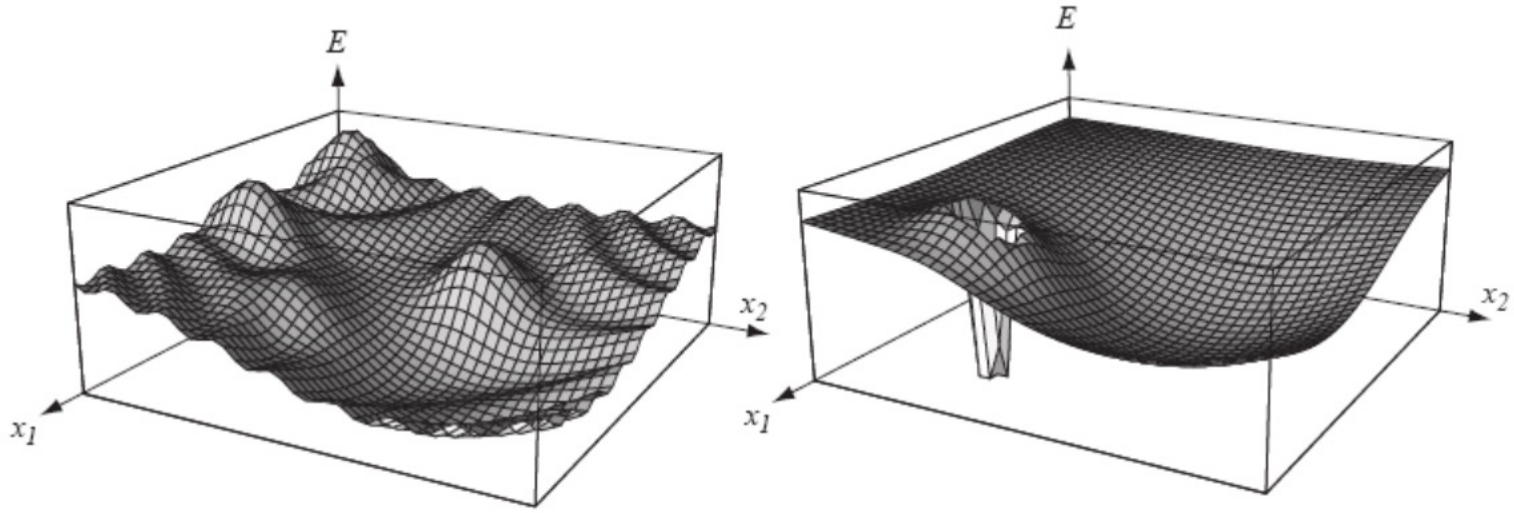
Initially temperature is very high (most bad moves accepted)

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

Practical Issues with simulated annealing

The cost function must be carefully developed, it must be "fractal and smooth". The energy function of the left would work with SA while the one of the right would fail.



Practical Issues with simulated annealing

- The cost function should be fast. It is going to be called “millions” of times.

- The best is if we just have to calculate the deltas produced by the modification instead of traversing through all the state.

- This is dependent on the application.

Practical Issues with simulated annealing

- In asymptotic convergence, simulated annealing converges to globally optimal solutions.
- In practice, the convergence of the algorithm depends on the cooling schedule.
- There are some suggestions about the cooling schedule, but it still requires a lot of testing and usually depends on the application.
  - Start at a temperature where 50%(or higher) of bad moves are accepted.
  - Each cooling step reduces the temperature by 10%(or smaller).
  - The number of iterations at each temperature should attempt to move between 1-10 times each "element" of the state.
  - The final temperature should not accept bad moves; this step is known as the quenching step.

Simulated annealing: Steps for solving Sudoku

1. Generate initial state

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

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

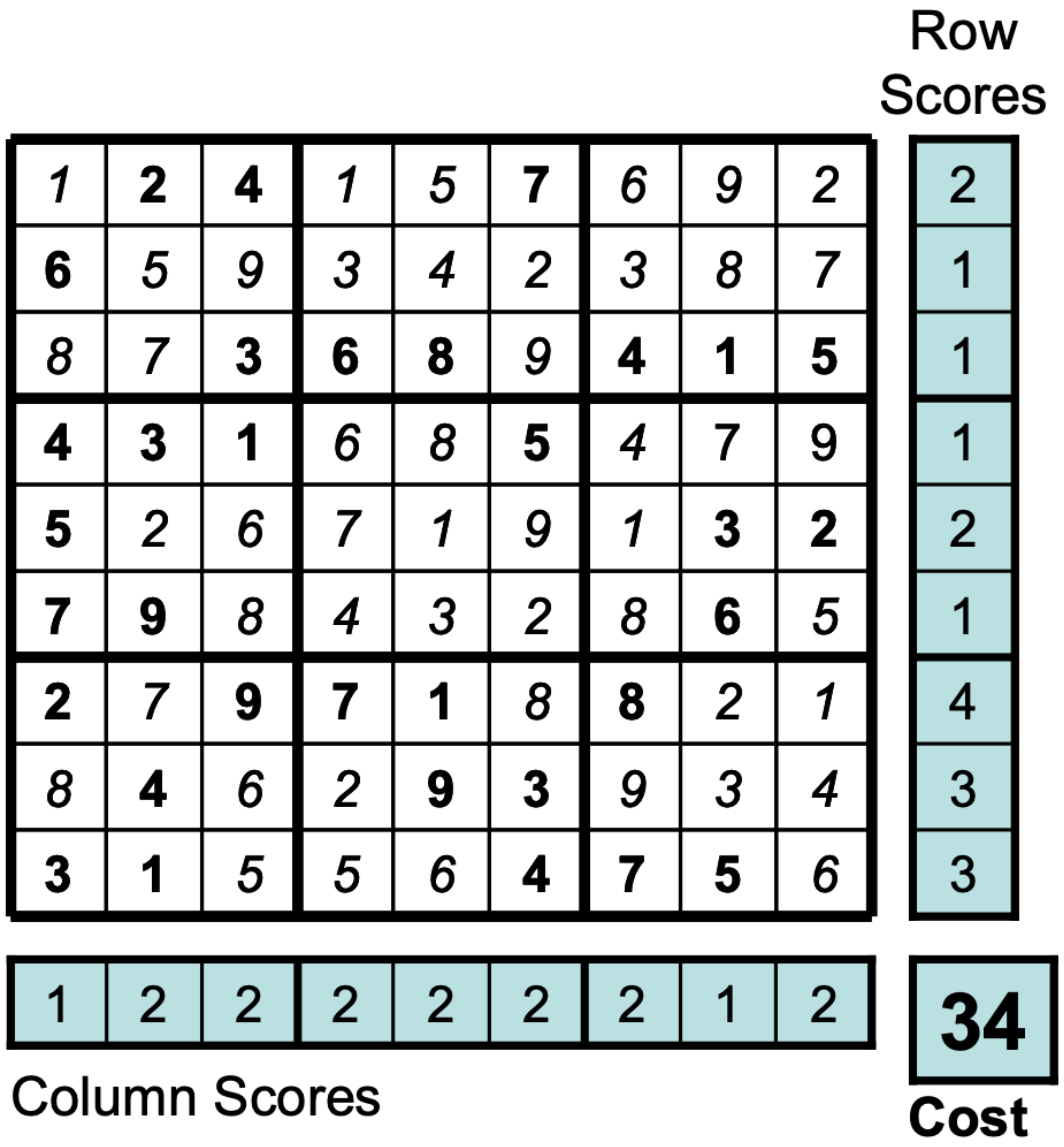
Fill the grid by assigning each non-fixed cell in the grid a value. This is done randomly, but make sure every square contains the values 1 to n^2 **exactly once** when the grid is full.

Simulated annealing: Steps for solving Sudoku

1. **Generate random states**
2. **Design a cost function:**

Scan each row individually and calculate the number of values, 1 through to n^2 that are not present.

The same is then done for each column, and the cost is simply the total of these values.



Simulated annealing: Steps for solving Sudoku

1. **Generate a random state**
2. **Design a cost function**
3. **How to generate the next state**
4. **Determine the initial temperature**
5. **Determine the cooling schedule function**

You can read the paper below for more detailed information:

https://www.researchgate.net/publication/220403361_Metaheuristics_can_solve_Sudoku_puzzles