Hill Climbing And Simulated Annealing

YAO ZHAO

Hill Climbing

```
hillClimbing(startNode){
    x ← startNode
    while (halt condition is not satisfied ) {
        Generate solution x' based on x
        Evaluate the f (x')
        If f(x')>f(x) {
            x = x'
        }
    }
}
```

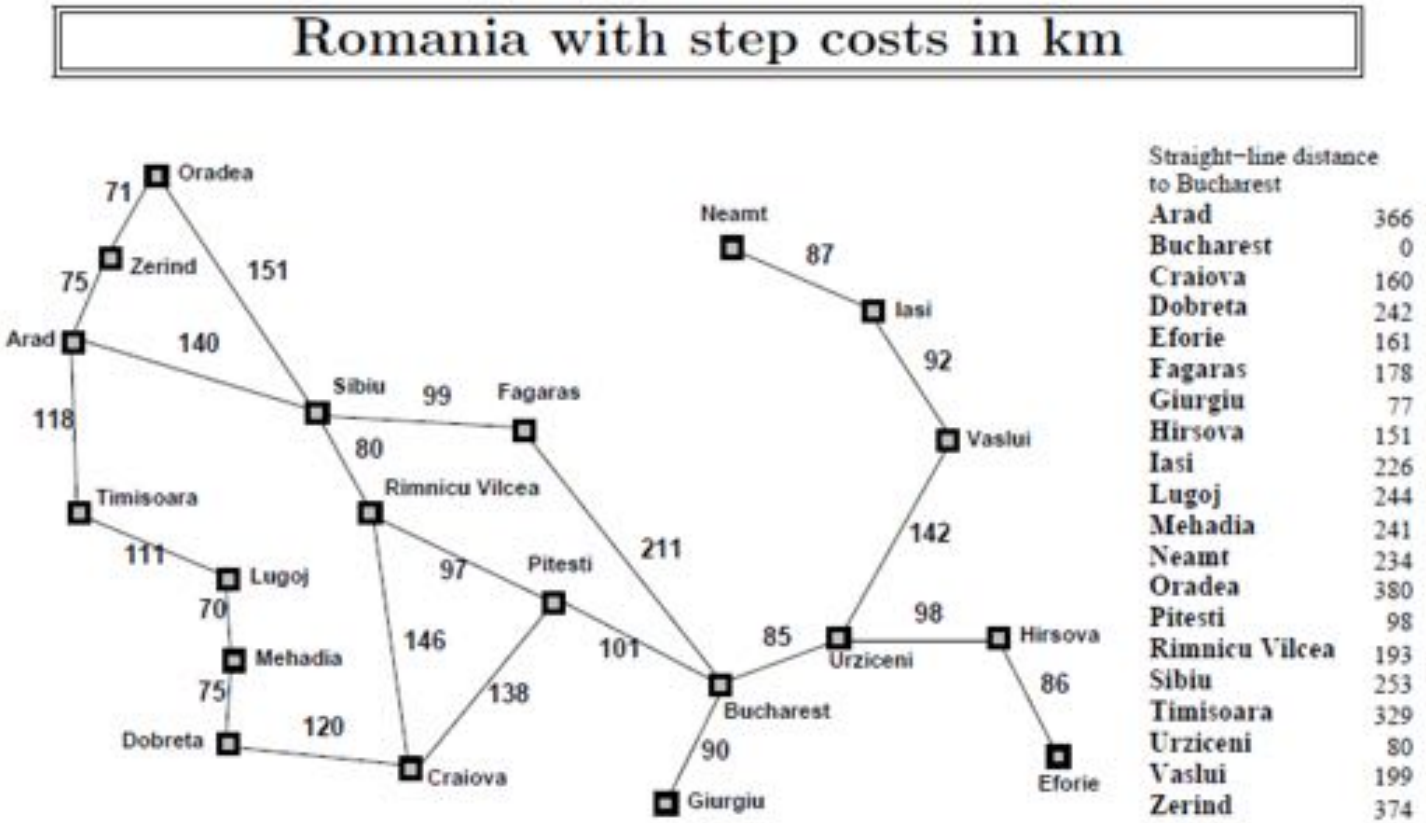
Hill Climbing For Route Search

Start: Arad
Target: Bucharest

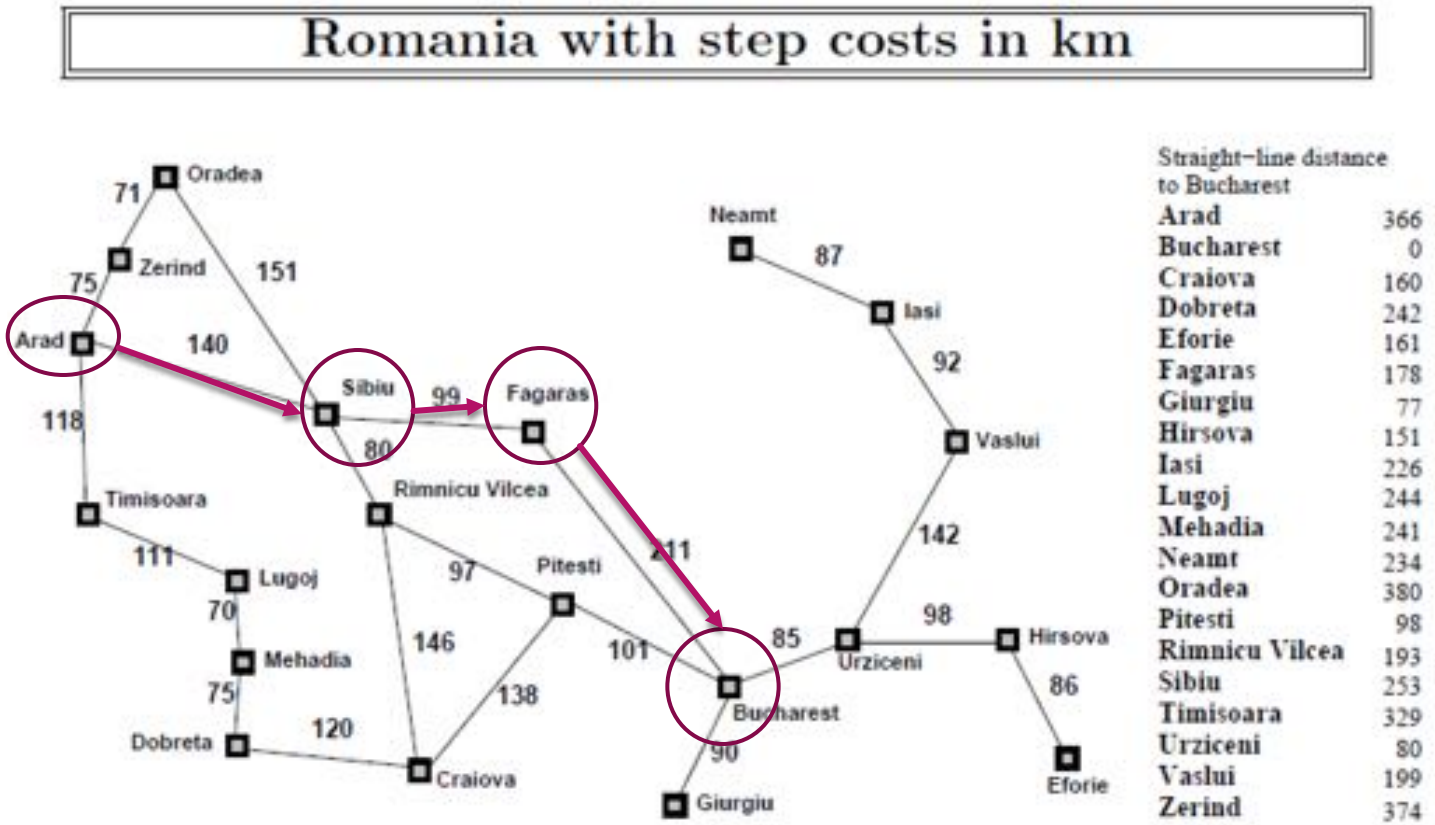
f(x) = straight-line distance to Bucharest

How to generate x' based on x?

Randomly select the adjacency node of x

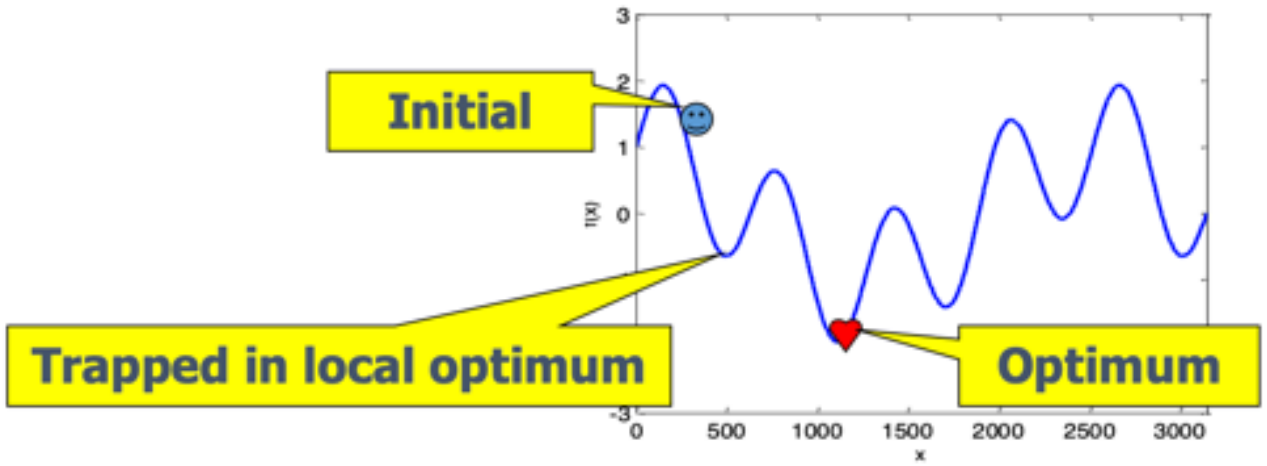


One possible execution process of a hill climbing algorithm for route search



Hill Climbing

- Hill-climbing inherits the advantage and disadvantage of greedy algorithms:
  - advantage: simple and easy implementation
  - disadvantage: depends on the initial point, always find a local optimal point and can not find global optimum



Simulated annealing

Simulated annealing is a probabilistic method proposed in Kirkpatrick, Gelett and Vecchi (1983) and Cerny (1985) for finding the global minimum of a cost function that may possess several local minima. It works by emulating the physical process whereby a solid is slowly cooled so that when eventually its structure is "frozen," this happens at a minimum energy configuration.

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** only depends on the current state (x) and new state (x'), and is controlled by temperature:

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

The temperature is controlled by **schedule function**.

- Simulated annealing combines random search and greedy search

Simulated annealing VS Hill Climbing

```
hillClimbing(startNode){
    x ← startNode
    while (halt condition is not satisfied ) {
        Generate solution x' based on x
        Evaluate the f (x')
        If f(x')>f(x) {
            x = x'
        }
    }
}
```

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

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: $t = \frac{t}{1+\beta t}$

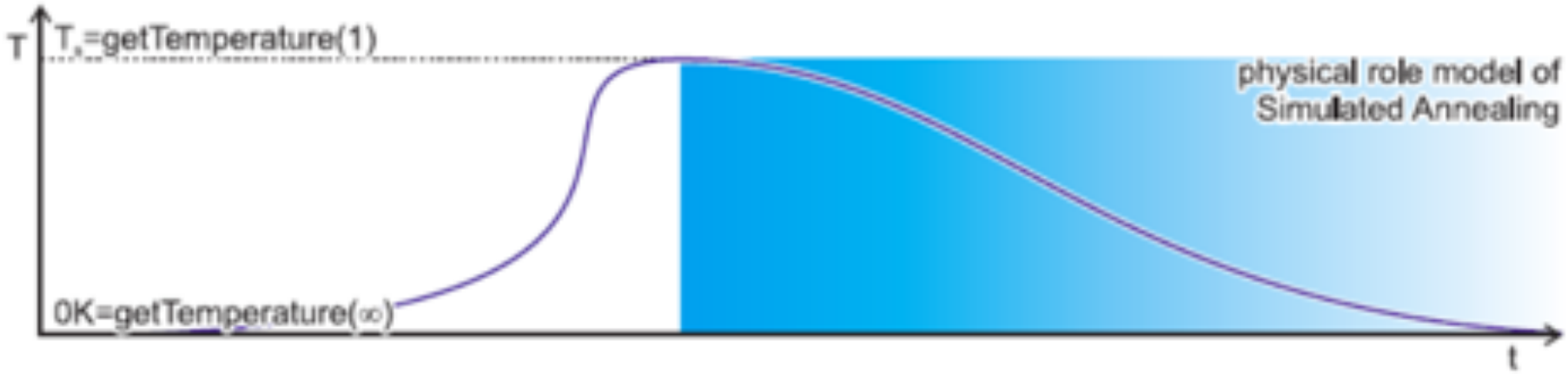
```
simulatedAnnealing(startNode, t0, schedule=expSchedule()){
    x ← startNode
    initial k, lam, limit
    t ← t0
    while (halt condition is not satisfied ) {
          T = schedule(k, lam, limit, t)
          Generate solution x' based on x
           Evaluate the f (x')
           If (f(x')>f(x)) or (Math.exp((f(x')-f(x))/T)>random(0.0, 1.0)) {
                    x = x'
           }//end if
           update(t)
      }//end whhile
}//end simulatedAnnealing
```

Initial temperature

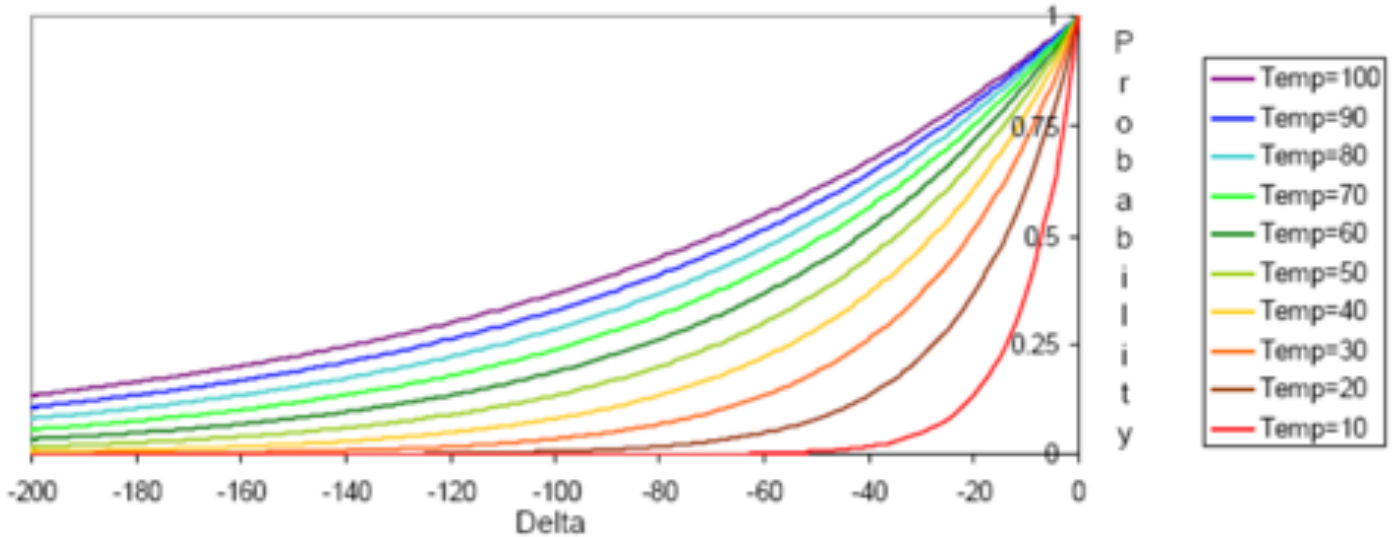
Cooling schedule function

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.

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



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

```
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
```

Lam become larger

```
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
```

K become smaller

```
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
```

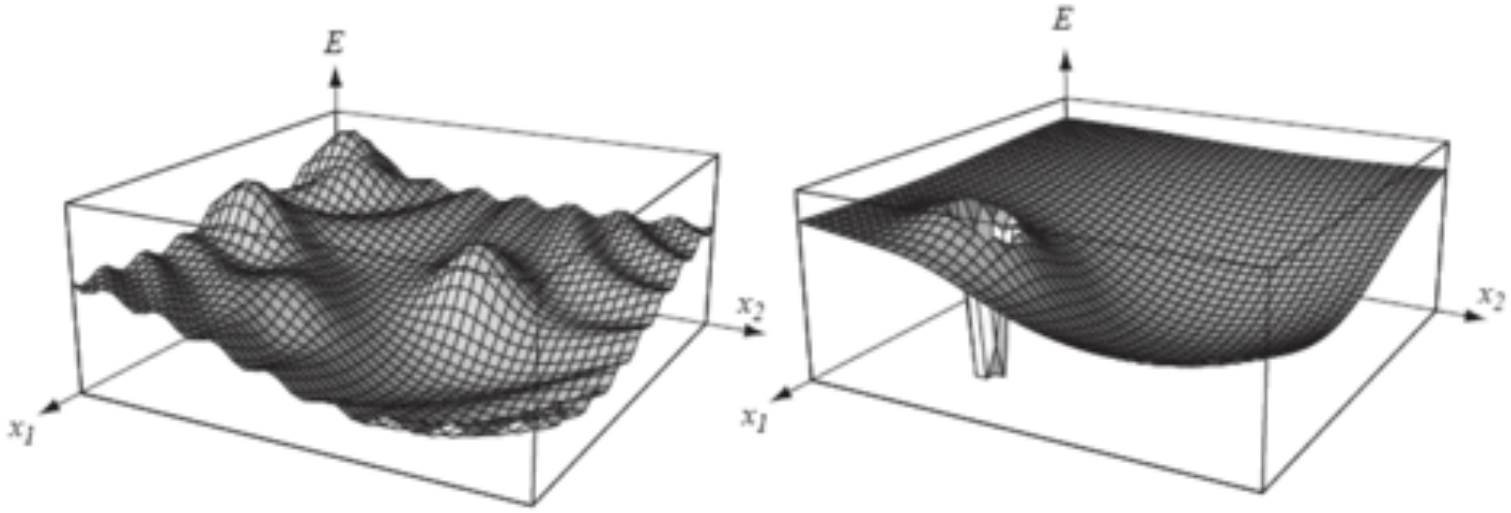
Limit become smaller

```
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
```

Practical Issues with simulated annealing

Cost function must be carefully developed, it has to 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 suggestion about the cooling schedule, but it stills requires a lot of testing and it 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

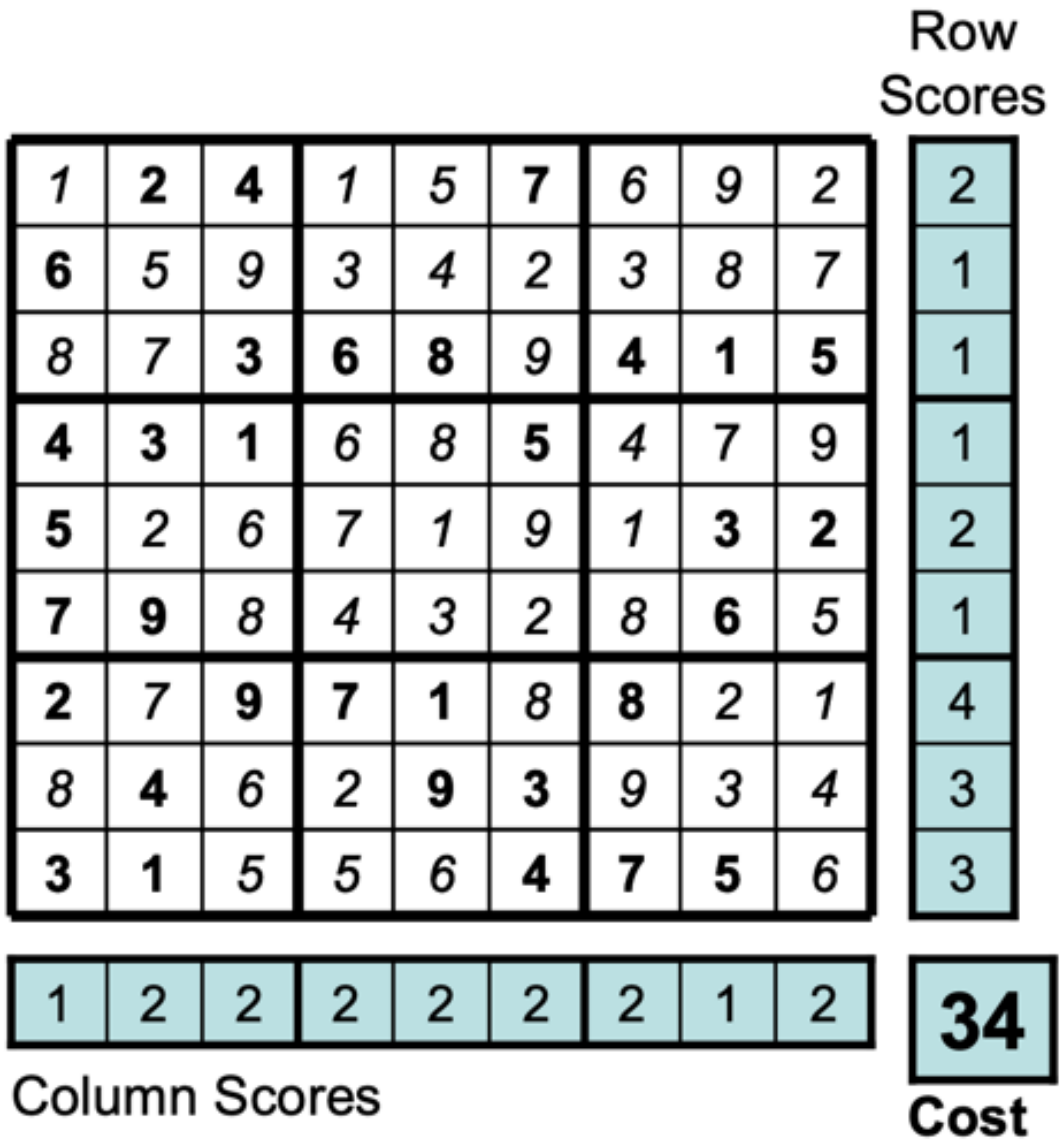
Fill the grid by assigning each non-fixed cell in the grid a value. This is done randomly, but make sure every square contains the values 1 to n^2 exactly once when the grid is full.

Simulated annealing: Steps for solving Sudoku

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

Scan each row individually and calculates 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 random states
2. Design a cost function
3. How to generate next states
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