Description: **Simulated annealing** (**SA**) is a probabilistic technique for approximating the global optimum of a given function E(s) (called energy function). Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem. For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to exact algorithms such as gradient descent or branch and bound.

Explaination:

Basic iteration: at each step, the simulated annealing will:

* Heuristic considers some neighboring state s\* from current state s:
* Probabilistically decides between moving the system to state s\* or staying in-state s:

These probabilities lead the system to move to states of lower energy.

This step will repeat until the system reaches a state that is good enough, or until reaches maximum iteration.

1. The state and the neighbours of a state:

For TSP problem, a ***state***is defined as a permutation of customers to be visited.

***neighbours*** *of a state* are a set of permutation by swapping 2 customers in s

1. Acceptance probabilities:

The probability of changing current state s to s\* is based on an ***acceptance probability*** function:

P( E, Enew, T) in which E = E(s)

Enew = E(s\*)

T = a global time-vary parameter called “temperature”

State with smaller energy ( for TSP problem it’s time taken, total distance, …) are better than those with higher energy. However the P function must be a positive value even when Enew is better than E so that the method won’t stuck at a local minimum worse than the global one.

When T �T tends to zero, the probability P �(�,�new,�)Pmust tend to zero if Enew < E �new>�dand to a positive value. For sufficiently small values of T�, the system will then increasingly favor moves that go ‘downhill’ (to lower energy values), and avoid those that go ‘uphill’. If T reaches zero, we got greedy algorithm.

The function P we decided to use is: P = 1 if E(s\*) < E(s)

exp( (E(s) – E(s\*)) / T) otherwise

After each iteration, we reduce T by ( 1 – α ) to lower the value P which mean we tend to favor ‘downhill’ value more and more as number of iteration performed increases.

If a better state is found, we check for time window constrain to decide whether accept this state or wait for next better state that not violate the constrain. In short, we let the algorithm keep exploring the best value while only keep the best-and-valid one.