LAB - 05

22-10-2024

Date __/__/__
Page ____

Stimulated Annealing.

the following algorithm presents the stimulated annealing heuristic:

① initialse parameters
- set the initial solutions
- set the initial temperature T
- Define coding rate α $(0 < \alpha < 1)$
- Set the stopping criterion

② Iterate:
- Repeat until a stopping condition (like a low temperature or a certain no of iteration) is met
- Generate a neighboring state: slightly modify the current state to explore new solutions
- Evalute energy: calculate the energy (objective function) of the new state.

Acceptance Decision
- If the new state has lower energy than the previous accept it
- If the higher energy is found accepts it with probability that it is and how much worse the now state it

metropolis creteria

④ Cool down: ~~Gradually reduce the temperature according to the cooling schedule.~~

⑤ Stop: once the temperature is low or after a set number of iteration stop & retur the best solution found.

Code :-

```
import numpy as np
import matplotlib pyplot as plt

def rastrigin (x)
    A = 10
    return A * len(x) + sum([(xi **2 - A * np.cos(
            2* np.pi * xi))
         for xi in x])

def simulated annealing (start, initial_temp,
                cooling_state, max_iter)
    current_solution = start
    current_energy = rastrigen (current solution)

    best_solution = current_Solution
    best_energy = current_energy
    temp - initial_temp

    energies = [current_energy]

    for i in range (max_iter):
        candidate_solution = current_Solution
            +np. random.uniform (-1, 1, size =
                    len(start)
    candidate_solution = np.clip (candidate
          solution, -5.12, 5.12)
    candidate_energy = rastragein (candidate
                    _solution)
    data energy = candidate_energy - current_energy
    if delta_energy > 0 :
        current_solution = candidate solution
        current_energy = candidate_energy
```

Date __/__/__
Page ______

```
else:
    acceptance_prob = np.exp(delta_energ / temp)

if np.random.rand() < acceptance.prob
    current_solution = candidate_solution
    best_energy = current_energy

temp *= cooling_rate
energies.append(current_energy)

return best_solution, best energy, energies
```

output :-

```
Best solution : [-4.540...  4.519...]

Best energy : 80.642.
```