

Simulated Annealing Algorithm

tham drapin
mahant drapin

Step-1 Initialisation

Choose initial soln 'S'

(x) niching - sithijis job

Set initial temp 'T'

Set a stopping criteria (target achieved)

much + (x) nel * 0) nichant

([x is ix ref

Step-2: Iteration

while (stopping criteria not met)

(1.0 = sith - feth, x) niching - daf job

(:] x = nichingant

(1 - (x) nel, 0) nichant - nichant = nichant

Generate neighbour - Create neighbouring solution 'S' from 'S'

Calculate energy difference:

Compute cost of $E(S)$ and neighbouring $E(S')$

(nichant ni nichant of ([1] nichant, [0] nichant) niching - nichant = nichant
(nichant) nichant = nichant - nichant

Step-3: Acceptance criteria:

nichant - nichant, nichant = nichant - nichant, nichant

If $E(S') < E(S)$ → accept 'S' as new solution

else accept 'S' with probability

(nichant - n) nichant ni j feth

$$P = \exp\left(\frac{E(S) - E(S')}{T}\right)$$

(1 + i) nichant feth = feth

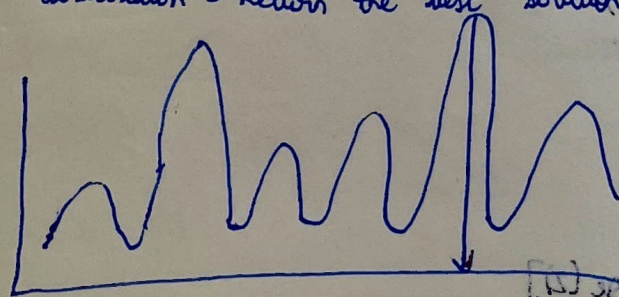
(nichant - nichant, nichant) nichant daf = nichant

update S to S' as new solution if accepted

(nichant) nichant = nichant - nichant, nichant = nichant - nichant, nichant
([1] (nichant - nichant - nichant

Step-4

Termination - Return the best solution found



Proceed
(nichant)

([1] nichant ni feth (0.2, 0.2 -) = nichant
nichant = nichant - n

Code -

```
import math
import random
```

```
def objective - function (x)
```

```
    return 10 * len(x) + sum([xi**2 - 10 * math.cos(2 * math.pi * xi) for xi in x])
```

```
def get - neighbour (x, step - size = 0.1):
```

```
    neighbour = x[:]
```

```
    index = random.randint(0, len(x) - 1)
```

```
    neighbour[index] += random.uniform(-step-size, step-size)
```

```
    return neighbour
```

```
def simulated (objective, bound, n - iteration, temp)
```

```
    best = [random.uniform(bound[0], bound[1]) for bound in bound]
```

```
    best - eval = objective(best)
```

```
    current, current - eval = best, best - eval
```

```
    scores = [best - eval]
```

```
    for i in range(n - iteration)
```

```
        t = temp / float(i + 1)
```

```
        candidate = get - neighbour (current, step - size)
```

```
        if eval < best - eval or random.random() < math.exp((best - eval - eval - candidate - eval) / t):
```

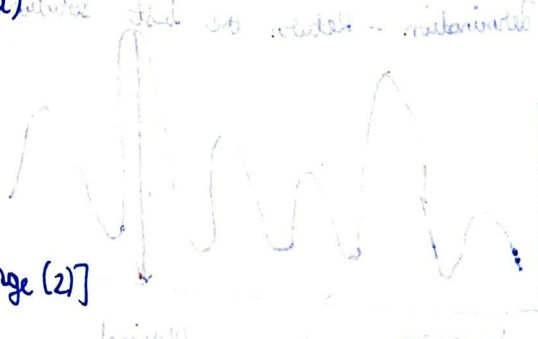
```
            scores.append(best - eval)
```

```
        if i % 100 == 0:
```

```
            print()
```

```
bound = [(-5.0, 5.0) for _ in range(2)]
```

```
n - iteration = 1000
```



step-size = 0.1

temp = 10

print()

print()

01/05

2-8A1

- group 8

mitigolo *A

1-126

OUTPUT -

Generation 0

Temperature 10.0

Best evaluation

41.275

100

0.099

16.954

200

0.050

16.926

300

0.033

16.916

400

0.025

16.916

500

0.020

16.915

600

0.017

16.915

700

0.014

16.914

800

0.012

16.914

900

0.011

Best Solution: [3.98054, 0.9962213]

Best score: 16.914632

8/22/024

(1) group - fitness = best - score
(fitness, fitness) best - score

(0 = 1 best - score) fitness

(1) min - fit - best - score = fitness

(0 = 1 fitness) fitness

fitness fitness

best score

(fitness) fitness is fitness

(1) fitness, fitness) best - score

"fitness" fitness