|  |  |
| --- | --- |
| **Оюутны код: B190900003** | **Лаборатори №: 3** |
| **Оюутны нэр: Д. Батсүх** |  |

# Даалгавар 1: Travelling sells person бодлогыг Simulated annealing algorithm ашиглан бодно уу.

# • Хавтгай дээр дурын давхцаагүй 6 цэгийг тодорхойлно

# • Тухайн 6 цэгийг А, В, С, Д, Г, Е хотууд гэж үзэн тэдгээрийн хооронд зорчих хамгийн бага зайг Simulated annealing ашиглаж тооцолно

# • Дараах үр дүнг хэвлэж харуулна

# • 6 хотын байрлалыг

# • 6 хотын хоорондын замыг Simulated annealing ашиглан зурсан үр дүн

Main code:

from anneal import SimAnneal

import matplotlib.pyplot as plt

import random

def read\_coords(*path*):

    coords = []

    with open(path, "r") as f:

        for line in f.readlines():

            line = [*float*(x.replace("\n", "")) for x in line.split(" ")]

            coords.append(line)

    return coords

def generate\_random\_coords(*num\_nodes*):

    return [[random.uniform(-1000, 1000), random.uniform(-1000, 1000)] for i in range(num\_nodes)]

if \_\_name\_\_ == "\_\_main\_\_":

    coords = read\_coords("coord.txt")

    sa = SimAnneal(coords, *stopping\_iter*=5000)

    sa.anneal()

    sa.visualize\_routes()

    sa.plot\_learning()

Simulated annealing class code:

import math

import random

import visualize\_tsp

import matplotlib.pyplot as plt

class SimAnneal(*object*):

    def \_\_init\_\_(*self*, *coords*, *T*=-1, *alpha*=-1, *stopping\_T*=-1, *stopping\_iter*=-1):

*self*.coords = coords

*self*.N = len(coords)

*self*.T = math.sqrt(*self*.N) if T == -1 else T

*self*.T\_save = *self*.T

*self*.alpha = 0.995 if alpha == -1 else alpha

*self*.stopping\_temperature = 1e-8 if stopping\_T == -1 else stopping\_T

*self*.stopping\_iter = 100000 if stopping\_iter == -1 else stopping\_iter

*self*.iteration = 1

*self*.nodes = [i for i in range(*self*.N)]

*self*.best\_solution = None

*self*.best\_fitness = *float*("Inf")

*self*.fitness\_list = []

    def initial\_solution(*self*):

        cur\_node = random.choice(*self*.nodes)

        solution = [cur\_node]

        free\_nodes = *set*(*self*.nodes)

        free\_nodes.remove(cur\_node)

        while free\_nodes:

            next\_node = min(free\_nodes, *key*=lambda *x*: *self*.dist(cur\_node, x))

            free\_nodes.remove(next\_node)

            solution.append(next\_node)

            cur\_node = next\_node

        cur\_fit = *self*.fitness(solution)

        if cur\_fit < *self*.best\_fitness:

*self*.best\_fitness = cur\_fit

*self*.best\_solution = solution

*self*.fitness\_list.append(cur\_fit)

        return solution, cur\_fit

    def dist(*self*, *node\_0*, *node\_1*):

        coord\_0, coord\_1 = *self*.coords[node\_0], *self*.coords[node\_1]

        return math.sqrt((coord\_0[0] - coord\_1[0]) \*\* 2 + (coord\_0[1] - coord\_1[1]) \*\* 2)

    def fitness(*self*, *solution*):

        cur\_fit = 0

        for i in range(*self*.N):

            cur\_fit += *self*.dist(solution[i % *self*.N], solution[(i + 1) % *self*.N])

        return cur\_fit

    def p\_accept(*self*, *candidate\_fitness*):

        return math.exp(-abs(candidate\_fitness - *self*.cur\_fitness) / *self*.T)

    def accept(*self*, *candidate*):

        candidate\_fitness = *self*.fitness(candidate)

        if candidate\_fitness < *self*.cur\_fitness:

*self*.cur\_fitness, *self*.cur\_solution = candidate\_fitness, candidate

            if candidate\_fitness < *self*.best\_fitness:

*self*.best\_fitness, *self*.best\_solution = candidate\_fitness, candidate

        else:

            if random.random() < *self*.p\_accept(candidate\_fitness):

*self*.cur\_fitness, *self*.cur\_solution = candidate\_fitness, candidate

    def anneal(*self*):

*self*.cur\_solution, *self*.cur\_fitness = *self*.initial\_solution()

        print("Starting annealing.")

        while *self*.T >= *self*.stopping\_temperature and *self*.iteration < *self*.stopping\_iter:

            candidate = *list*(*self*.cur\_solution)

            l = random.randint(2, *self*.N - 1)

            i = random.randint(0, *self*.N - l)

            candidate[i : (i + l)] = reversed(candidate[i : (i + l)])

*self*.accept(candidate)

*self*.T \*= *self*.alpha

*self*.iteration += 1

*self*.fitness\_list.append(*self*.cur\_fitness)

        print("Best fitness obtained: ", *self*.best\_fitness)

        improvement = 100 \* (*self*.fitness\_list[0] - *self*.best\_fitness) / (*self*.fitness\_list[0])

        print(f"Improvement over greedy heuristic: {improvement : .2f}%")

    def batch\_anneal(*self*, *times*=10):

        for i in range(1, times + 1):

            print(f"Iteration {i}/{times} -------------------------------")

*self*.T = *self*.T\_save

*self*.iteration = 1

*self*.cur\_solution, *self*.cur\_fitness = *self*.initial\_solution()

*self*.anneal()

    def visualize\_routes(*self*):

        visualize\_tsp.plotTSP([*self*.best\_solution], *self*.coords)

    def plot\_learning(*self*):

        plt.plot([i for i in range(len(*self*.fitness\_list))], *self*.fitness\_list)

        plt.ylabel("Fitness")

        plt.xlabel("Iteration")

        plt.show()

Visualize\_tcp code

import matplotlib.pyplot as plt

def plotTSP(*paths*, *points*, *num\_iters*=1):

    x = []; y = []

    for i in paths[0]:

        x.append(points[i][0])

        y.append(points[i][1])

    plt.plot(x, y, 'co')

    a\_scale = *float*(max(x))/*float*(100)

    if num\_iters > 1:

        for i in range(1, num\_iters):

            xi = []

            yi = []

            for j in paths[i]:

                xi.append(points[j][0])

                yi.append(points[j][1])

            plt.arrow(xi[-1], yi[-1], (xi[0] - xi[-1]), (yi[0] - yi[-1]),

*head\_width* = a\_scale, *color* = 'r',

*length\_includes\_head* = True, *ls* = 'dashed',

*width* = 0.001/*float*(num\_iters))

            for i in range(0, len(x) - 1):

                plt.arrow(xi[i], yi[i], (xi[i+1] - xi[i]), (yi[i+1] - yi[i]),

*head\_width* = a\_scale, *color* = 'r', *length\_includes\_head* = True,

*ls* = 'dashed', *width* = 0.001/*float*(num\_iters))

    plt.arrow(x[-1], y[-1], (x[0] - x[-1]), (y[0] - y[-1]), *head\_width* = a\_scale,

*color* ='g', *length\_includes\_head*=True)

    for i in range(0,len(x)-1):

        plt.arrow(x[i], y[i], (x[i+1] - x[i]), (y[i+1] - y[i]), *head\_width* = a\_scale,

*color* = 'g', *length\_includes\_head* = True)

    plt.xlim(min(x)\*1.1, max(x)\*1.1)

    plt.ylim(min(y)\*1.1, max(y)\*1.1)

    plt.show()

Оролт:

A. -4.5 -9

B. -52 -36

C. -53 -36

D. 0 0.01

G. -30 -18

E. -51 -35

Гаралт:  
 129.95413006555745

Visualize:

