Код:

import itertools  
import random  
from numpy.random import choice  
  
MUT\_PROB = 0.01  
  
def route\_length(route, matrix):  
 l = 0  
 for i in range(len(route) - 1):  
 l += matrix[route[i]][route[i + 1]]  
 l += matrix[route[-1]][route[0]]  
 return l  
  
  
def make\_child(p1, p2, splits):  
 child = [None] \* splits[0] + p2[splits[0]:splits[1]] + [None] \* (c - splits[1])  
 i = 0  
 j = splits[0] + 1  
 stop = False  
 while not stop:  
 if child[i] is not None:  
 i += 1  
 if i >= c:  
 stop = True  
 continue  
 while not stop:  
 if p1[j] in child:  
 j += 1  
 if j >= c:  
 j = 0  
 if j == splits[0] + 1:  
 stop = True  
 continue  
 child[i] = p1[j]  
 break  
 return child  
  
  
def mutate\_child(child, prob=MUT\_PROB):  
 if random.random() < prob:  
 splits = list(choice(range(c), size=2, replace=False))  
 child[splits[0]], child[splits[1]] = child[splits[1]], child[splits[0]]  
 return True  
 return False  
  
  
def make\_children(p1, p2):  
 while True:  
 splits = sorted(list(choice(range(c + 1), size=2, replace=False)))  
 if 2 <= splits[1] - splits[0] < c-1:  
 break  
 c1 = make\_child(p1, p2, splits)  
 c2 = make\_child(p2, p1, splits)  
 par1 = ''.join(map(str, p1[:splits[0]])) + '|' + ''.join(map(str, p1[splits[0]:splits[1]])) + '|' + ''.join(map(str, p1[splits[1]:]))  
 par2 = ''.join(map(str, p2[:splits[0]])) + '|' + ''.join(map(str, p2[splits[0]:splits[1]])) + '|' + ''.join(map(str, p2[splits[1]:]))  
 ch1 = ''.join(map(str, c1[:splits[0]])) + '|' + ''.join(map(str, c1[splits[0]:splits[1]])) + '|' + ''.join(map(str, c1[splits[1]:]))  
 ch2 = ''.join(map(str, c2[:splits[0]])) + '|' + ''.join(map(str, c2[splits[0]:splits[1]])) + '|' + ''.join(map(str, c2[splits[1]:]))  
 for i in range(1, 3):  
 print(f"{locals()[f'par{i}']} | {locals()[f'ch{i}']} | {route\_length(locals()[f'c{i}'], matrix)}")  
 if mutate\_child(c1):  
 print("Потомок 1 мутировал: " + ''.join(map(str, c1)))  
 if mutate\_child(c2):  
 print("Потомок 2 мутировал: " + ''.join(map(str, c2)))  
 return c1, c2  
  
  
def generation(c, matrix, p, g):  
 global first\_length, first\_average, first\_sum, final\_length, final\_average, final\_sum  
 og\_cities = list(range(c))  
 population = sorted([random.sample(og\_cities, len(og\_cities)) for \_ in range(p)], key=lambda route: route\_length(route, matrix))  
 for i in range(g):  
 print(f"Поколение {i + 1}")  
 lengths = [route\_length(route, matrix) for route in population]  
 if (i == 0):  
 first\_length = lengths[0]  
 first\_sum = sum(lengths)  
 first\_average = first\_sum / len(lengths)  
 probabilities = [1 / length for length in lengths]  
 total\_probability = sum(probabilities)  
 probabilities = [prob / total\_probability for prob in probabilities]  
 if (i != 0 and i < g - 1):  
 print("Код | Значение целевой функции | Вероятность участия в размножении")  
 for i, (code, length, prob) in enumerate(zip(population, lengths, probabilities), 1):  
 print(f"{''.join(map(str, code))} | {length} | {prob}")  
 else:  
 print("Код | Значение целевой функции")  
 for i, (code, length) in enumerate(zip(population, lengths), 1):  
 print(f"{''.join(map(str, code))} | {length}")  
 print()  
  
 all\_pairs = list(itertools.combinations(range(p), 2))  
 pairs = random.sample(all\_pairs, p // 2)  
 pairs\_str = ", ".join([f"({pair[0]}, {pair[1]})" for pair in pairs])  
 print(f"Пусть выбраны пары: {pairs\_str}")  
 print("Родители | Потомки | Значение целевой функции для потомков")  
 for j, pair in enumerate(pairs):  
 p1 = population[pair[0]]  
 p2 = population[pair[1]]  
 children = make\_children(p1, p2)  
 unique\_children = []  
 for child in children:  
 if child not in population:  
 unique\_children.append(child)  
  
 population += unique\_children  
 print()  
 population.sort(key=lambda route: route\_length(route, matrix))  
 population = population[:p]  
 print()  
 print(f"Финальное поколение")  
 lengths = [route\_length(route, matrix) for route in population]  
 final\_length = lengths[0]  
 final\_sum = sum(lengths)  
 final\_average = final\_sum / len(lengths)  
 print("Код | Значение целевой функции")  
 for i, (code, length) in enumerate(zip(population, lengths), 1):  
 print(f"{''.join(map(str, code))} | {length}")  
 print()  
 return population[0], route\_length(population[0], matrix)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 c = 5  
 matrix = [[0, 4, 5, 3, 8], [4, 0, 7, 6, 8], [5, 7, 0, 7, 9], [3, 6, 7, 0, 9], [8, 8, 9, 9, 0]]  
 p = 4  
 g = 3  
 print()  
 result, length = generation(c, matrix, p, g)  
 print(f"Таким образом после {g} итераций значение целевой функции для лучшего решения изменилось c {first\_length} на {final\_length}, среднее значение изменилось с {first\_average} до {final\_average}, а общее качество с {first\_sum} до {final\_sum}.")

Вывод программы:

Поколение 1

Код | Значение целевой функции

31240 | 33

14032 | 33

04123 | 33

31402 | 34

Пусть выбраны пары: (0, 2), (1, 3)

Родители | Потомки | Значение целевой функции для потомков

31|24|0 | 40|12|3 | 35

04|12|3 | 30|24|1 | 31

|14|032 | |31|402 | 34

|31|402 | |14|023 | 34

Поколение 2

Код | Значение целевой функции | Вероятность участия в размножении

30241 | 31 | 0.2619047619047619

31240 | 33 | 0.24603174603174605

14032 | 33 | 0.24603174603174605

04123 | 33 | 0.24603174603174605

Пусть выбраны пары: (1, 3), (2, 3)

Родители | Потомки | Значение целевой функции для потомков

312|40| | 014|23| | 31

041|23| | 312|40| | 33

140|32| | 140|23| | 34

041|23| | 041|32| | 34

Поколение 3

Код | Значение целевой функции

30241 | 31

01423 | 31

31240 | 33

14032 | 33

Пусть выбраны пары: (1, 3), (0, 1)

Родители | Потомки | Значение целевой функции для потомков

01|423| | 14|032| | 33

14|032| | 10|423| | 34

3|024|1 | 3|142|0 | 31

0|142|3 | 3|024|1 | 31

Финальное поколение

Код | Значение целевой функции

30241 | 31

01423 | 31

31420 | 31

31240 | 33

Таким образом после 3 итераций значение целевой функции для лучшего решения изменилось c 33 на 31, среднее значение изменилось с 33.25 до 31.5, а общее качество с 133 до 126.