

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет ПИ и КТ

Лабораторная работа №6

по дисциплине: «Методы оптимизации»

«Генетическое программирование»

Вариант 1

Выполнил:

**Болорболд Аригуун**,

группа P3211

Преподаватель:

**Селина Елена Георгиевна**

Санкт-Петербург

2024



**Задание:**

Дано множество из *n* городов и матрица расстояний между ними. Требуется объехать все города по кратчайшему пути, причем в каждом городе необходимо побывать один раз и вернуться в город, из которого был начат маршрут. Задачу необходимо решить с помощью генетического алгоритма.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Город 1 | Город 2 | Город 3 | Город 4 | Город 5 |
| Город 1 | 0 | 4 | 5 | 3 | 8 |
| Город 2 | 4 | 0 | 7 | 6 | 8 |
| Город 3 | 5 | 7 | 0 | 7 | 9 |
| Город 4 | 3 | 6 | 7 | 0 | 9 |
| Город 5 | 8 | 8 | 9 | 9 | 0 |

За целевую функцию следует принять сумму расстояний между городами.

Размер популяции N = 4.

Оператор мутации представляет собой случайную перестановку двух чисел в геноме, которые выбираются случайно.

Вероятность мутации 0.01.

**Решение:**

Программная реализация задачи (Python):

from random import (random, randint)  
from bisect import bisect  
from itertools import permutations  
  
graph = [  
 [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]  
]  
cities\_count = len(graph)  
population\_size = 4  
mutation\_probability = 0.01  
  
  
class Path:  
 def \_\_init\_\_(self, sequence):  
 self.sequence = sequence  
  
 def f(self):  
 distance = 0  
 for i in range(1, cities\_count):  
 distance += graph[self.sequence[i - 1]][self.sequence[i]]  
 distance += graph[self.sequence[-1]][self.sequence[0]]  
 return distance  
  
  
def choose\_parents(paths):  
 parents = []  
  
 non\_chosen\_paths = [path for path in paths]  
 for i in range(population\_size):  
 sum\_of\_distances = 0  
 for path in non\_chosen\_paths:  
 sum\_of\_distances += path.f()  
 probabilities = []  
 for path in non\_chosen\_paths:  
 probabilities.append(path.f() / sum\_of\_distances)  
  
 cumulative\_probabilities = []  
 cumulative\_sum = 0  
 for probability in probabilities:  
 cumulative\_sum += probability  
 cumulative\_probabilities.append(cumulative\_sum)  
  
 r = random()  
  
 chosen\_path\_index = bisect(cumulative\_probabilities, int(r))  
 parents.append(non\_chosen\_paths[chosen\_path\_index])  
 non\_chosen\_paths.remove(non\_chosen\_paths[chosen\_path\_index])  
 return parents  
  
  
def get\_sequence\_with\_break\_points(sequence, break\_point\_1, break\_point\_2):  
 res = ""  
 for i in range(cities\_count):  
 if i in (break\_point\_1, break\_point\_2):  
 res += "|"  
 res += str(sequence[i])  
 return res  
  
  
def create\_next\_population(parents):  
 pairs = []  
 for i in range(0, len(parents), 2):  
 pairs.append([parents[i], parents[i + 1]])  
  
 next\_population = []  
  
 for pair in pairs:  
 parent\_1\_sequence = pair[0].sequence  
 parent\_2\_sequence = pair[1].sequence  
  
 break\_point\_1 = randint(1, cities\_count - 1)  
 break\_point\_2 = randint(1, cities\_count - 1)  
  
 print(f"Скрешиваем: {get\_sequence\_with\_break\_points(parent\_1\_sequence, break\_point\_1, break\_point\_2)} {get\_sequence\_with\_break\_points(parent\_2\_sequence, break\_point\_1, break\_point\_2)}")  
  
 while break\_point\_2 == break\_point\_1:  
 break\_point\_2 = randint(1, cities\_count - 1)  
  
 if break\_point\_2 < break\_point\_1:  
 break\_point\_1, break\_point\_2 = break\_point\_2, break\_point\_1  
  
 taken\_cities = []  
  
 child\_1 = [0 for i in range(cities\_count)]  
 for i in range(break\_point\_1, break\_point\_2):  
 child\_1[i] = parent\_2\_sequence[i]  
 taken\_cities.append(parent\_2\_sequence[i])  
 i = 0  
 j = break\_point\_1  
 while i < break\_point\_1:  
 if parent\_1\_sequence[j] not in taken\_cities:  
 child\_1[i] = parent\_1\_sequence[j]  
 taken\_cities.append(parent\_1\_sequence[j])  
 i += 1  
 if j == cities\_count - 1:  
 j = 0  
 else:  
 j += 1  
 i = break\_point\_2  
 while i < cities\_count:  
 if parent\_1\_sequence[j] not in taken\_cities:  
 child\_1[i] = parent\_1\_sequence[j]  
 taken\_cities.append(parent\_1\_sequence[j])  
 i += 1  
 if j == cities\_count - 1:  
 j = 0  
 else:  
 j += 1  
  
 taken\_cities.clear()  
  
 child\_2 = [0 for i in range(cities\_count)]  
 for i in range(break\_point\_1, break\_point\_2):  
 child\_2[i] = parent\_1\_sequence[i]  
 taken\_cities.append(parent\_1\_sequence[i])  
 i = 0  
 j = break\_point\_1  
 while i < break\_point\_1:  
 if parent\_2\_sequence[j] not in taken\_cities:  
 child\_2[i] = parent\_2\_sequence[j]  
 taken\_cities.append(parent\_2\_sequence[j])  
 i += 1  
 if j == cities\_count - 1:  
 j = 0  
 else:  
 j += 1  
 i = break\_point\_2  
 while i < cities\_count:  
 if parent\_2\_sequence[j] not in taken\_cities:  
 child\_2[i] = parent\_2\_sequence[j]  
 taken\_cities.append(parent\_2\_sequence[j])  
 i += 1  
 if j == cities\_count - 1:  
 j = 0  
 else:  
 j += 1  
  
 print(f"Получены потомки: {get\_sequence\_with\_break\_points(child\_1, break\_point\_1, break\_point\_2)} {get\_sequence\_with\_break\_points(child\_2, break\_point\_1, break\_point\_2)}")  
 print("="\*77)  
  
 next\_population.append(Path(child\_1))  
 next\_population.append(Path(child\_2))  
  
 return next\_population  
  
  
def mutation(path):  
 random\_number = random()  
 if random\_number <= mutation\_probability:  
 index\_1 = randint(0, cities\_count - 1)  
 index\_2 = index\_1  
 while index\_2 == index\_1:  
 index\_2 = randint(0, cities\_count - 1)  
  
 mutated\_path\_sequence = path.sequence  
  
 mutated\_path\_sequence[index\_1], mutated\_path\_sequence[index\_2] = mutated\_path\_sequence[index\_2], \  
 mutated\_path\_sequence[index\_1]  
  
 print("Произошла мутация:", "".join(map(str, path.sequence)), "-->", "".join(map(str, mutated\_path\_sequence)))  
  
 return Path(mutated\_path\_sequence)  
  
 return None  
  
  
population\_count = 3  
cities = range(cities\_count)  
  
population = []  
for sequence in permutations(cities):  
 if len(population) < population\_size:  
 population.append(Path(sequence))  
 else:  
 break  
  
current\_population\_number = 1  
  
while True:  
 print(f"Популяция №{current\_population\_number}:")  
 print(" Путь | Значение целевой функции | Вероятность размножения")  
 sum\_of\_distances = 0  
 for path in population:  
 sum\_of\_distances += path.f()  
 for path in population:  
 print(f'{"".join(map(str, path.sequence))} | {path.f()} | {path.f()}/{sum\_of\_distances}')  
  
 if current\_population\_number == population\_count:  
 break  
  
 print("="\*77)  
  
 parents = choose\_parents(population)  
 print("В качестве родителей выбраны следующие особи:")  
 for parent in parents:  
 print("".join(map(str, parent.sequence)))  
 print("="\*77)  
  
 next\_population = create\_next\_population(parents)  
  
 for i in range(len(next\_population)):  
 child = mutation(next\_population[i])  
 if child is not None:  
 next\_population[i] = child  
  
 population += next\_population  
 print("Расширенная популяция:")  
 print(" Путь | Значение целевой функции")  
 for path in population:  
 print(f'{"".join(map(str, path.sequence))} | {path.f()}')  
 population.sort(key=lambda x: x.f())  
 population = population[:population\_size]  
  
 current\_population\_number += 1  
 print()  
  
print()  
  
optimal\_path = population[0]  
print("Оптимальный путь:", "".join(map(str, optimal\_path.sequence)), "\nРасстояние:", optimal\_path.f())

**Результат работы программы:**

Популяция №1:

Путь | Значение целевой функции | Вероятность размножения

01234 | 35 | 35/134

01243 | 32 | 32/134

01324 | 34 | 34/134

01342 | 33 | 33/134

=========================================================

В качестве родителей выбраны следующие особи:

01234

01243

01324

01342

=========================================================

Скрешиваем: 0|123|4 0|124|3

Получены потомки: 3|124|0 4|123|0

=========================================================

Скрешиваем: 01|324 01|342

Получены потомки: 24|3|01 42|3|01

=========================================================

Произошла мутация: 23401 --> 23401

Расширенная популяция:

Путь | Значение целевой функции

01234 | 35

01243 | 32

01324 | 34

01342 | 33

31240 | 33

41230 | 33

23401 | 35

42301 | 31

Популяция №2:

Путь | Значение целевой функции | Вероятность размножения

42301 | 31 | 31/129

01243 | 32 | 32/129

01342 | 33 | 33/129

31240 | 33 | 33/129

==========================================================

В качестве родителей выбраны следующие особи:

42301

01243

01342

31240

==========================================================

Скрешиваем: 42|301 01|243

Получены потомки: 2|1|304 1|2|430

==========================================================

Скрешиваем: 0|1342 3|1240

Получены потомки: 3|124|0 2|134|0

==========================================================

Расширенная популяция:

Путь | Значение целевой функции

42301 | 31

01243 | 32

01342 | 33

31240 | 33

21304 | 33

12430 | 32

31240 | 33

21340 | 35

Популяция №3:

Путь | Значение целевой функции | Вероятность размножения

42301 | 31 | 31/128

01243 | 32 | 32/128

12430 | 32 | 32/128

01342 | 33 | 33/128

Оптимальный путь: 42301

Расстояние: 31

**Выводы о работе:**

На этом и заканчивается курс методов оптимизации. В этом работе я поработал и освоил генетический алгоритм решения задачи. Задачи о коммивояжере — классическая задача, где решение обычным алгоритмическим методом приведет к факториальном времени, что никак не является практичным в любой сфере деятельности, поэтому знакомство с хотя бы одной разновидностью таких методов, который решает данную задачу (хотя бы немного) быстрее — полезная навыка.