Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №6
по дисциплине
«Методы оптимизации»

Выполнил:
Студент группы P3207
Разинкин А.В.

Преподаватель:
Селина Е.Г.

г. Санкт-Петербург
2024

Условие задания

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	1	1	5	3
Город 2	1	0	3	1	5
Город 3	1	3	0	11	1
Город 4	5	1	11	0	1
Город 5	3	5	1	1	0

Реализация кодом (Python)

```python
from random import (random, randint)
from bisect import bisect
from itertools import permutations

graph = [
    [0, 1, 1, 5, 3],
    [1, 0, 3, 1, 5],
    [1, 3, 0, 11, 1],
    [5, 1, 11, 0, 1],
    [3, 5, 1, 1, 0]
]
cities_count = len(graph)
population_size = 4
mutation_probability = 0.01


class Way:
    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(ways):
    parents = []

    non_chosen_ways = [way for way in ways]
    for i in range(population_size):
        sum_of_distances = 0
        for way in non_chosen_ways:
            sum_of_distances += way.f()
        probabilities = []
        for way in non_chosen_ways:
            probabilities.append(way.f() / sum_of_distances)

        cumulative_probabilities = []
        cumulative_sum = 0
        for probability in probabilities:
            cumulative_sum += probability
            cumulative_probabilities.append(cumulative_sum)

        r = random()

        chosen_way_index = bisect(cumulative_probabilities, r)
        parents.append(non_chosen_ways[chosen_way_index])
        non_chosen_ways.remove(non_chosen_ways[chosen_way_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("")

        next_population.append(Way(child_1))
        next_population.append(Way(child_2))

    return next_population


def mutation(way):
    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_way_sequence = way.sequence

        mutated_way_sequence[index_1], mutated_way_sequence[index_2] =
mutated_way_sequence[index_2], \
        mutated_way_sequence[index_1]
```

```
        print("Произошла мутация:", "".join(map(str, way.sequence)), "-->",
"".join(map(str, mutated_way_sequence)))

        return Way(mutated_way_sequence)

    return None


population_count = 3
cities = range(cities_count)

population = []
for sequence in permutations(cities):
    if len(population) < population_size:
        population.append(Way(sequence))
    else:
        break

current_population_number = 1

while True:
    print(f"Популяция №{current_population_number}:")
    print("Путь | Значение целевой функции | Вероятность участия в процессе
размножения")
    sum_of_distances = 0
    for way in population:
        sum_of_distances += way.f()
    for way in population:
        print(f"{"".join(map(str, way.sequence))} | {way.f()} |
{way.f()}/{sum_of_distances}")

    if current_population_number == population_count:
        break

    print("")

    parents = choose_parents(population)
    print("В качестве родителей выбраны:")
    for parent in parents:
        print("".join(map(str, parent.sequence)))
    print("")

    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("Полученная расширенная популяция:")
    print("Путь | Значение целевой функции")
    for way in population:
        print(f"{"".join(map(str, way.sequence))} | {way.f()}")
    population.sort(key=lambda x: x.f())
    population = population[:population_size]
```

```
    current_population_number += 1
    print("")

print("")

optimal_way = population[0]
print("Оптимальный путь:", "".join(map(str, optimal_way.sequence)),
"Расстояние:", optimal_way.f())
```

Результат работы программы

Популяция №1:
Путь | Значение целевой функции | Вероятность участия в процессе размножения
01234 | 19 | 19/52
01243 | 11 | 11/52
01324 | 17 | 17/52
01342 | 5 | 5/52

В качестве родителей выбраны:
01243
01324
01234
01342

Скрешиваем: 012|4|3 013|2|4
Получены потомки:430|2|1 201|4|3

Скрешиваем: 01|2|34 01|3|42
Получены потомки:24|3|01 34|2|01

Полученная расширенная популяция:
Путь | Значение целевой функции
01234 | 19
01243 | 11
01324 | 17
01342 | 5
43021 | 15
20143 | 19
24301 | 11
34201 | 5

Популяция №2:
Путь | Значение целевой функции | Вероятность участия в процессе размножения
01342 | 5 | 5/32
34201 | 5 | 5/32
01243 | 11 | 11/32
24301 | 11 | 11/32

В качестве родителей выбраны:
01342
34201
24301
01243

Скрешиваем: 0|13|42 3|42|01
Получены потомки:1|42|30 4|13|20

Скрешиваем: 24|30|1 01|24|3
Получены потомки:30|24|1 24|30|1

Полученная расширенная популяция:
Путь | Значение целевой функции
01342 | 5
34201 | 5
01243 | 11
24301 | 11
14230 | 23
41320 | 21
30241 | 13
24301 | 11

Популяция №3:
Путь | Значение целевой функции | Вероятность участия в процессе размножения
01342 | 5 | 5/32
34201 | 5 | 5/32
01243 | 11 | 11/32
24301 | 11 | 11/32

В качестве родителей выбраны:
24301
01243
01342
34201

Скрешиваем: 24|3|01 01|2|43
Получены потомки:30|2|14 24|3|01

Скрешиваем: 0|134|2 3|420|1
Получены потомки:1|420|3 2|134|0

Полученная расширенная популяция:
Путь | Значение целевой функции
01342 | 5
34201 | 5
01243 | 11
24301 | 11
30214 | 15
24301 | 11
14203 | 13
21340 | 9

Популяция №4:

Путь | Значение целевой функции | Вероятность участия в процессе размножения
01342 | 5 | 5/30
34201 | 5 | 5/30
21340 | 9 | 9/30
01243 | 11 | 11/30

В качестве родителей выбраны:
01243
34201
01342
21340

Скрешиваем: 012|4|3 342|0|1
Получены потомки:431|0|2 013|4|2

Скрешиваем: 0|1|342 2|1|340
Получены потомки:3|1|420 3|1|402

Полученная расширенная популяция:
Путь | Значение целевой функции
01342 | 5
34201 | 5
21340 | 9
01243 | 11
43102 | 5
01342 | 5
31420 | 13
31402 | 21

Популяция №5:
Путь | Значение целевой функции | Вероятность участия в процессе размножения
01342 | 5 | 5/20
34201 | 5 | 5/20
43102 | 5 | 5/20
01342 | 5 | 5/20

Оптимальный путь: 01342 Расстояние: 5

Вывод

Я ознакомился с идеями генетического алгоритма и применил их для решения задачи о коммивояжере.