Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 «Программная инженерия» –

Системное и прикладное программное обеспечение

Отчёт

По лабораторной работе №6

По методам оптимизации

Вариант: 4

Выполнил:

студент 2 курса

Батманов Даниил Евгеньевич

Группа: P3207

Приняла:

Селина Елена Георгиевна

Отчёт принят «__»_____2024 г.

Оценка: _____

г. 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.

Ручное решение

Исходная популяция (поколение I):

№	Путь	Значение целевой функции
1	2, 4, 3, 0, 1	32
2	4, 3, 2, 0, 1	33
3	2, 1, 4, 0, 3	33
4	1, 0, 4, 3, 2	35

Рассмотрим пары (2; 0) и (1; 0):

Пара 1: [2, 0]

Родитель 1: 2 1 | 4 0 | 3

Родитель 2: 2 4 | 3 0 | 1

Потомок 1: 2 1 3 0 4

Потомок 2: 1 2 4 0 3

Пара 2: [1, 0]

Родитель 1: 4 3 2 | 0 1 |

Родитель 2: 2 4 3 | 0 1 |

Потомок 1: 4 3 2 0 1

Потомок 2: 2 4 3 0 1

Расширенная популяция:

№	Путь	Значение целевой функции
1	2, 4, 3, 0, 1	32
2	2, 4, 3, 0, 1	32
3	4, 3, 2, 0, 1	33
4	2, 1, 4, 0, 3	33
5	2, 1, 3, 0, 4	33
6	1, 2, 4, 0, 3	33
7	4, 3, 2, 0, 1	33
8	1, 0, 4, 3, 2	35

Поколение II после работы оператора редукции:

№	Путь	Значение целевой функции
1	2, 4, 3, 0, 1	32
2	2, 4, 3, 0, 1	32
3	4, 3, 2, 0, 1	33
4	2, 1, 4, 0, 3	33

Рассмотрим пары (3; 1) и (2; 3):

Пара 1: [3, 1]

Родитель 1: 2 | 1 4 0 3 |

Родитель 2: 2 | 4 3 0 1 |

Потомок 1: 2 4 3 0 1

Потомок 2: 2 1 4 0 3

Пара 2: [2, 3]

Родитель 1: 4 3 | 2 0 1 |

Родитель 2: 2 1 | 4 0 3 |

Потомок 1: 1 2 4 0 3

Потомок 2: 3 4 2 0 1

Расширенная популяция:

№	Путь	Значение целевой функции
1	2, 4, 3, 0, 1	32
2	2, 4, 3, 0, 1	32
3	2, 4, 3, 0, 1	32
4	4, 3, 2, 0, 1	33
5	2, 1, 4, 0, 3	33
6	2, 1, 4, 0, 3	33
7	1, 2, 4, 0, 3	33

8	3, 4, 2, 0, 1	33

Поколение III после работы оператора редукции:

№	Путь	Значение целевой функции
1	2, 4, 3, 0, 1	32
2	2, 4, 3, 0, 1	32
3	2, 4, 3, 0, 1	32
4	4, 3, 2, 0, 1	33

Рассмотрим пары (1; 3) и (0; 2):

Пара 1: [1, 3]

Родитель 1: 2 4 | 3 0 | 1

Родитель 2: 4 3 | 2 0 | 1

Потомок 1: 1 4 2 0 3

Потомок 2: 1 4 3 0 2

Пара 2: [0, 2]

Родитель 1: 2 4 | 3 0 | 1

Родитель 2: 2 4 | 3 0 | 1

Потомок 1: 1 2 3 0 4

Потомок 2: 1 2 3 0 4

Расширенная популяция:

№	Путь	Значение целевой функции
1	1, 4, 2, 0, 3	31
2	2, 4, 3, 0, 1	32
3	2, 4, 3, 0, 1	32
4	2, 4, 3, 0, 1	32
5	1, 4, 3, 0, 2	32
6	4, 3, 2, 0, 1	33
7	1, 2, 3, 0, 4	33
8	1, 2, 3, 0, 4	33

Вывод: в результате работы алгоритма за 3 поколения получили оптимальный путь: 1, 4, 2, 0, 3 со значением 31.

Исходный код программы

```
import random
from numpy.random import choice
from pprint import pprint

def calculate_route_length(path, distance_matrix):
    total_length = 0
```

```
    for index in range(len(path) - 1):
        total_length += distance_matrix[path[index]][path[index + 1]]
    total_length += distance_matrix[path[-1]][path[0]]
    return total_length

def create_offspring(parent1, parent2, crossover_points):
    offspring = [None] * crossover_points[0] +
parent2[crossover_points[0]:crossover_points[1]] + [None] * (num_cities -
crossover_points[1])
    i = 0
    j = crossover_points[0] + 1
    is_complete = False
    while not is_complete:
        if offspring[i] is not None:
            i += 1
            if i >= num_cities:
                is_complete = True
            continue
        while not is_complete:
            if parent1[j] in offspring:
                j += 1
                if j >= num_cities:
                    j = 0
                if j == crossover_points[0] + 1:
                    is_complete = True
                continue
            offspring[i] = parent1[j]
            break
    return offspring

def attempt_mutation(child, mutation_probability=0.01):
    if random.random() < mutation_probability:
        mutation_points = list(choice(range(num_cities), size=2, replace=False))
        child[mutation_points[0]], child[mutation_points[1]] =
child[mutation_points[1]], child[mutation_points[0]]
        return True
    return False

def generate_offspring(parent1, parent2):
    while True:
        crossover_points = sorted(list(choice(range(num_cities + 1), size=2,
replace=False)))
        if 2 <= crossover_points[1] - crossover_points[0] < num_cities:
            break
    print("Родитель 1: " + ' '.join(map(str, parent1[:crossover_points[0]])) + ' | ' +
' '.join(map(str, parent1[crossover_points[0]:crossover_points[1]])) + ' | ' + '
'.join(map(str, parent1[crossover_points[1]:])))
    print("Родитель 2: " + ' '.join(map(str, parent2[:crossover_points[0]])) + ' | ' +
' '.join(map(str, parent2[crossover_points[0]:crossover_points[1]])) + ' | ' + '
'.join(map(str, parent2[crossover_points[1]:])))
    offspring1 = create_offspring(parent1, parent2, crossover_points)
    offspring2 = create_offspring(parent2, parent1, crossover_points)
    print("Потомок 1: " + ' '.join(map(str, offspring1)))
    print("Потомок 2: " + ' '.join(map(str, offspring2)))
    if attempt_mutation(offspring1):
        print("Потомок 1 мутированный: " + ' '.join(map(str, offspring1)))
    if attempt_mutation(offspring2):
        print("Потомок 2 мутированный: " + ' '.join(map(str, offspring2)))
    return offspring1, offspring2


def evolve_population(num_cities, distance_matrix, pop_size, num_generations):
    initial_population = list(range(num_cities))
    population = sorted([random.sample(initial_population, len(initial_population)) for
_ in range(pop_size)], key=lambda path: calculate_route_length(path, distance_matrix))
```

```
    for generation in range(num_generations):
        print(f"Поколение {generation + 1}")
        print(f"Популяция:")
        pprint(population)
        lengths = [calculate_route_length(path, distance_matrix) for path in
population]
        print(f"Значенния целевой функции:")
        pprint(lengths)
        selection_probabilities = [1 / length for length in lengths]
        selection_probabilities = [prob / sum(selection_probabilities) for prob in
selection_probabilities]
        mating_pairs = [list(choice(range(pop_size), size=2, p=selection_probabilities,
replace=False)) for _ in range(pop_size // 2)]
        for pair_index, pair in enumerate(mating_pairs):
            print(f"\nПара {pair_index + 1}: {pair}")
            parent1 = population[pair[0]]
            parent2 = population[pair[1]]
            offspring = generate_offspring(parent1, parent2)
            population += offspring
        print()
        population.sort(key=lambda path: calculate_route_length(path, distance_matrix))
        print(f"Расширенная популяция:")
        pprint(population)
        lengths = [calculate_route_length(path, distance_matrix) for path in
population]
        print(f"Значенния целевой функции:")
        pprint(lengths)
        population = population[:pop_size]
        print()
    return population[0], calculate_route_length(population[0], distance_matrix)


num_cities = int(input("Введите количество: "))
print("Введите матрицу: ")
distance_matrix = [list(map(int, input().split())) for _ in range(num_cities)]
pop_size = int(input("Введите размер популяции: "))
num_generations = int(input("Введите количество поколений: "))
print()
best_path, best_length = evolve_population(num_cities, distance_matrix, pop_size,
num_generations)
print(f"Результат после {num_generations} поколений {best_path} со значением
{best_length}")
```

Результат работы программы

```
Введите количество городов: 5
Введите матрицу:
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
Введите размер популяции: 4
Введите количество поколений: 20

Поколение 1
Популяция:
[[0, 2, 4, 3, 1], [1, 4, 0, 2, 3], [0, 1, 2, 3, 4], [2, 1, 0, 4, 3]]
Значенния целевой функции:
[33, 34, 35, 35]

Пара 1: [3, 0]
Родитель 1:  | 2 1 0 4 | 3
Родитель 2:  | 0 2 4 3 | 1
Потомок 1: 0 2 4 3 1
Потомок 2: 2 1 0 4 3

Пара 2: [1, 0]
Родитель 1: 1 4 0 | 2 3 |
Родитель 2: 0 2 4 | 3 1 |
Потомок 1: 4 0 2 3 1
Потомок 2: 1 0 4 2 3

Расширенная популяция:
[[0, 2, 4, 3, 1],
 [0, 2, 4, 3, 1],
 [1, 4, 0, 2, 3],
 [4, 0, 2, 3, 1],
 [1, 0, 4, 2, 3],
 [0, 1, 2, 3, 4],
 [2, 1, 0, 4, 3],
 [2, 1, 0, 4, 3]]
Значенния целевой функции:
[33, 33, 34, 34, 34, 35, 35, 35]

Поколение 2
… (*продолжение*)
```

Вывод

В ходе выполнения данной лабораторной работы я научился решать задачу о коммивояжере при помощи генетического алгоритма, написал код, реализующий этот алгоритм, выполнил первые 3 итерации алгоритма руками. Большее количество итераций (поколений) позволяет получить более точный ответ, быстрый подсчёт результата работы алгоритма можно получить при помощи кода, который я предоставил.