Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №6

по дисциплине «Методы оптимизации»

Вариант: 17

Преподаватель:
Селина Елена Георгиевна

Выполнил:
Тимошкин Роман Вячеславович
Группа: P3231

Санкт-Петербург, 2025

Решение вручную:

Исходная популяция

№ строки	Код	Значение целевой функции
1	14352	13
2	12543	15
3	32514	23
4	53214	24

Пусть для скрещивания были выбраны следующие пары: (3, 4) и (1, 3)
В результате были получены потомки:

№ строки	Родители	Потомки	Значение целевой функции для потомков
1	3|251|4	5|321|4	24
2	5|321|4	4|251|3	21
3	14|35|2	24|51|3	32
4	32|51|4	14|35|2	13

Популяция первого поколения после отсечения худших особей в результате работы оператора редукции:

№ строки	Код	Значение целевой функции	Вероятность участия в процессе размножения
1	14352	13	0.33
2	12543	15	0.28
3	42513	21	0.20
4	32514	23	0.18

Пусть для получения второго поколения были выбраны следующие пары строк: (1, 2) и (2, 4).
В результате были получены потомки:

№ строки	Родители	Потомки	Значение целевой функции для потомков
1	143|52|	215|43|	26
2	125|43|	314|52|	24
3	12|54|3	43|51|2	20
4	32|51|4	13|54|2	22

Популяция второго поколения после отсечения худших особей в результате работы оператора редукции:

№ строки	Код	Значение целевой функции
1	14352	13

2	12543	15
3	43512	20
4	42513	21

Пусть для получения третьего поколения были выбраны следующие пары строк: (2, 4) и (1, 4).
В результате были получены потомки:

№ строки	Родители	Потомки	Значение целевой функции для потомков
1	1|254|3	4|251|3	21
2	4|251|3	1|254|3	15
3	1|43|52	3|25|14	23
4	4|25|13	5|43|12	15

Финальное поколение

№ строки	Код	Значение целевой функции
1	14352	13
2	12543	15
3	54312	15
4	43512	20

Таким образом после 3 итераций значение целевой функции для лучшего решения изменилось с 13 на 13, среднее значение изменилось с 18.75 до 15.75, а общее качество с 75 до 63.

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(lambda x: str(x + 1), p1[:splits[0]])) + '|' + ''.join(
        map(lambda x: str(x + 1), p1[splits[0]:splits[1]])) + '|' +
        ''.join(map(lambda x: str(x + 1), p1[splits[1]:]))
    par2 = ''.join(map(lambda x: str(x + 1), p2[:splits[0]])) + '|' + ''.join(
        map(lambda x: str(x + 1), p2[splits[0]:splits[1]])) + '|' +
        ''.join(map(lambda x: str(x + 1), p2[splits[1]:]))
    ch1 = ''.join(map(lambda x: str(x + 1), c1[:splits[0]])) + '|' + ''.join(
        map(lambda x: str(x + 1), c1[splits[0]:splits[1]])) + '|' +
        ''.join(map(lambda x: str(x + 1), c1[splits[1]:]))
    ch2 = ''.join(map(lambda x: str(x + 1), c2[:splits[0]])) + '|' + ''.join(
        map(lambda x: str(x + 1), c2[splits[0]:splits[1]])) + '|' +
        ''.join(map(lambda x: str(x + 1), 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(lambda x: str(x + 1), c1)))
    if mutate_child(c2):
        print("Потомок 2 мутировал: " + ''.join(map(lambda x: str(x + 1), c2)))
    return c1, c2


def generation(c, matrix, p, g):
    global first_length, first_average, first_sum, final_length, final_average, final
    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 размножен
            for i, (code, length, prob) in enumerate(
                zip(population, lengths, probabilities), 1
            ):
                print(f"{''.join(map(lambda x: str(x + 1), code))} | {length}
        else:
            print("Код   | Значение целевой функции")
            for i, (code, length) in enumerate(zip(population, lengths), 1):
                print(f"{''.join(map(lambda x: str(x + 1), code))} | {length}")
        print()

        all_pairs = list(itertools.combinations(range(p), 2))
        pairs = random.sample(all_pairs, p // 2)
        pairs_str = ", ".join([f"({pair[0] + 1}, {pair[1] + 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(lambda x: str(x + 1), code))} | {length}")
    print()
    return population[0], route_length(population[0], matrix)


if __name__ == "__main__":
    c = 5
    matrix = [
        [0, 1, 7, 2, 8],
        [2, 0, 10, 3, 1],
        [7, 10, 0, 2, 6],
        [2, 3, 2, 0, 4],
        [8, 1, 6, 4, 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 с {first_length} на {final_length},
    среднее значение изменилось с {first_average} до {final_average},
    а общее качество с {first_sum} до {final_sum}.""")
```