МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор		Ю.А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Комбинаторная оптимизация

по дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	Z1431		М.Д.Быстров
	номер группы	подпись, дата	инициалы, фамилия

Студенческий билет № 2021/3572

Санкт-Петербург 2025

Оглавление

Индивидуальное задание

Реализовать с использованием генетических алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта (см. таблицу 3.1. и приложение Б.).

Сравнить найденное решение с представленным в условии задачи оптимальным решением.

Представить графически найденное решение.

Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.

Исходные данные ЗК:

bayg29: 29 городов в Баварии, географические расстояния (Groetschel, Juenger, Reinelt)

Тип данных – транспонированная диагональная матрица

```
97 205 139 86 60 220 65 111 115 227 95 82 225 168 103 266 205 149 120
58 257 152 52 180 136 82 34 145
129 103 71 105 258 154 112 65 204 150 87 176 137 142 204 148 148 49 41
211 226 116 197 89 153 124 74
219 125 175 386 269 134 184 313 201 215 267 248 271 274 236 272 160 151 300
350 239 322 78 276 220 60
167 182 180 162 208 39 102 227 60 86 34 96 129 69 58 60 120 119 192
114 110 192 136 173 173
51 296 150 42 131 268 88 131 245 201 175 275 218 202 119 50 281 238 131
244 51 166 95 69
279 114 56 150 278 46 133 266 214 162 302 242 203 146 67 300 205 111 238
98 139 52 120
178 328 206 147 308 172 203 165 121 251 216 122 231 249 209 111 169 72 338
144 237 331
169 151 227 133 104 242 182 84 290 230 146 165 121 270 91 48 158 200 39
64 210
172 309 68 169 286 242 208 315 259 240 160 90 322 260 160 281 57 192 107
90
140 195 51 117 72 104 153 93 88 25 85 152 200 104 139 154 134 149 135
320 146 64 68 143 106 88 81 159 219 63 216 187 88 293 191 258 272
174 311 258 196 347 288 243 192 113 345 222 144 274 124 165 71 153
144 86 57 189 128 71 71 82 176 150 56 114 168 83 115 160
61 165 51 32 105 127 201 36 254 196 136 260 212 258 234
106 110 56 49 91 153 91 197 136 94 225 151 201 205
215 159 64 126 128 190 98 53 78 218 48 127 214
61 155 157 235 47 305 243 186 282 261 300 252
105 100 176 66 253 183 146 231 203 239 204
113 152 127 150 106 52 235 112 179 221
79 163 220 119 164 135 152 153 114
236 201 90 195 90 127 84 91
273 226 148 296 238 291 269
112 130 286 74 155 291
130 178 38 75 180
281 120 205 270
213 145 36
94 217
```

```
162
```

bayg29: координаты городов:

```
1 1150.0 1760.0
2 630.0 1660.0
3 40.0 2090.0
4 750.0 1100.0
5 750.0 2030.0
6 1030.0 2070.0
7 1650.0 650.0
8 1490.0 1630.0
9 790.0 2260.0
10 710.0 1310.0
11 840.0 550.0
12 1170.0 2300.0
13 970.0 1340.0
14 510.0 700.0
15 750.0 900.0
16 1280.0 1200.0
17 230.0 590.0
18 460.0 860.0
19 1040.0 950.0
20 590.0 1390.0
21 830.0 1770.0
22 490.0 500.0
23 1840.0 1240.0
24 1260.0 1500.0
25 1280.0 790.0
26 490.0 2130.0
27 1460.0 1420.0
28 1260.0 1910.0
29 360.0 1980.0
EOF
```

bayg29:лучшие решения

```
1
28
6
12
9
26
3
29
5
21
2
20
10
4
15
18
14
17
22
11
19
25
7
23
8
27
16
13
```

```
24
-1
      EOF
```

Краткие теоретические сведения

Общие сведения

Задача коммивояжера (ЗК) считается классической задачей генетических алгоритмов. Она заключается в следующем: путешественник (или коммивояжер) должен посетить каждый из базового набора городов и вернуться к исходной точке. Имеется стоимость билетов из одного города в другой. Необходимо составить план путешествия, чтобы сумма затраченных средств была минимальной. Поисковое пространство для ЗК- множество из N городов. Любая комбинация из N городов, где города не повторяются, является решением. Оптимальное решение – такая комбинация, стоимость которой (сумма из стоимостей переезда между каждыми из городов в комбинации) является минимальной.

В настоящее время существует три основных представления пути: соседское, порядковое и путевое. Каждое из этих представлений имеет собственные полностью различные операторы рекомбинации.

Представление соседства

В представлении соседства тур является списком из n городов. Город J находится на позиции I только в том случае, если маршрут проходит из города I в город J. Например, вектор (2 4 8 3 9 7 1 5 6) представляет следующий тур: 1-2-4-3-8-5-9-6-7. Каждый маршрут имеет только одно соседское представление, но некоторые векторы в соседском представлении могут представлять неправильный маршрут. Например, вектор (2 4 8 1 9 3 5 7 6) обозначает маршрут 1-2-4-1..., т.е. часть маршрута – замкнутый цикл. Это представление не поддерживает классическую операцию кроссовера. Три операции кроссовера были определены и исследованы для соседского представления: alternating edges (альтернативные ребра), subtour chunks (куски подтуров), heuristic crossovers (евристический кроссовер).

Кроссовер обмен ребрами (alternating edges) строит потомков, выбрав (случайно) ребро от первого родителя, потом следующее ребро от второго, потом опять следующее от первого и т.д. Если новое ребро представляет

замкнутый цикл, из того же родителя берут случайное ребро, которое еще не выбирался и не образуют замкнутого цикла. Для примера, один из потомков родителей

P1 = (2 3 8 7 9 1 4 5 6) и

P2 = (7 5 1 6 9 2 8 4 3)

Может быть

П1 = (2 5 8 7 9 1 6 4 3),

где процесс начинался от угла (1,2) родителя P1, продолжая до угла (7,8), вместо которого выбран угол (7,8), поскольку тот образуют замкнутый цикл.

Кроссовер обмен подтуров (subtour chunks) создает потомков, выбирая (случайно) подтур от одного из родителей, затем случайной длины кусок от другого из родителей, и т.д. Как и в alternating edges, в случае образования замкнутого цикла, он ремонтируется аналогичным образом.

Эвристический кроссинговер (heuristic crossover) строит потомков, выбирая случайный город как стартовую точку для маршрута – потомка. Потом он сравнивает два соответствующих ребра от каждого из родителей и выбирает более короткое. Затем конечный город выбирается как начальный для выбора следующего более короткого ребра из этого города. Если на каком–то шаге получается замкнутый тур, тур продолжается любым случайным городом, который еще не посещался.

Преимущества этого представления – в том, что она позволяет схематически анализировать подобные маршруты. Это представление имеет в основании натуральные «строительные блоки» - ребра, маршруты между городами. Например, схема (* * * 3 * 7 * * *) описывает множество всех маршрутов с ребрами (4 3) и (6 7). Основной недостаток данного представления: множество операций бедно. Кроссовер alternating edges часто разрушает хорошие туры. Кроссовер subtour chunks имеет лучшие характеристики благодаря меньшим разрушительным свойствам. Но все равно его эксплуатационные качества все же достаточно низки. Кроссовер heuristic crossover, является наилучшим оператором для данного представления

благодаря тому, что остальные операции «слепы». Но производительность этой операции нестабильна. В трех экспериментах на 50, 100 и 200 городах система нашла туры с 25%, 16% и 27% оптимального, приблизительно за 15000, 20000 и 25000 итераций соответственно.

Листинг программы

Программа выполнена в системе MATLAB. Графики сформированы с помощью встроенных возможностей системы.

```
nodes_count = 29;
ways = read_matrix2(nodes_count);

f = @(x) fitness(x, ways);
heuristic_c = @(p1,p2) heuristic_cross(p1,p2,ways);


% res = commi_genetic(f, 100, nodes_count, 50000, 0.9, 0.1,
@alternating_cross);
% res = commi_genetic(f, 100, nodes_count, 50000, 0.9, 0.1,
@subtour_cross);
res = commi_genetic(f, 100, nodes_count, 5000, 0.9, 0.1, heuristic_c);

coords = read_coords(nodes_count);

draw_route(parse_path(res{1}), coords);


function res = commi_genetic( ...
    f, ...
    population_size, ...
    nodes_count, ...
    max_steps, ...
    cross_prob, ...
    mutation_prob, ...
    cross)

    population = generate_population( ...
        population_size, ...
        nodes_count);

    for i = 1:max_steps

        if (length(population) < population_size)
            a=1;
        end

        parents = reproduction(population, f);

        % Сохраняем лучшую особь
        best = best_individ(f, population);
        best_ind = population{best};

        if (best > 1)
            start = population(1:best-1);
        else
            start = cell(0,1);
        end

        if (best < population_size)
            ending = population(best+1:population_size);
        else
            ending = cell(0,1);
        end
```

```
        without_best_pop = [start;ending];

        % рождение и мутация детей
        children = crossover(parents, cross_prob, cross);
        children = mutation(children, mutation_prob);

        % редукция
        new_population = [without_best_pop; children];

        population = reduction(f, new_population, population_size -
1);
        population = [{best_ind}; population];

        best = best_individ(f, population);
        best_ind = population{best};

        fprintf("Шаг %d: лучший маршрут:\n%s\n%s\n", i,
mat2str(best_ind), mat2str(parse_path(best_ind)));
        fprintf("Длина маршрута: %d\n", f(best_ind));
    end

    best = best_individ(f, population);
    res = {population{best}; f(population{best})};
end

function draw_route(path, coords)
    x = zeros(1, length(path));
    y = zeros(1, length(path));

    for i = 1:length(path)
        node = path(i);
        x(i) = coords(node, 1);
        y(i) = coords(node, 2);
    end

    figure;
    plot(x, y, '-x');

end

function num = best_individ(f, pop)

    min_val = intmax();
    num = 0;

    for i = 1:length(pop)
        val = f(pop{i});
        if (val < min_val)
            min_val = val;
            num = i;
        end
    end
end

% Мутация
function mutated_pop = mutation(pop, prob)

    for i = 1:length(pop)
        if (rand() < prob)
            val = pop{i};

            while true

                pos1 = 0;
                pos2 = 0;
```

```
                while (pos1 == pos2)
                    pos1 = round(rand_range(1, length(val)));
                    pos2 = round(rand_range(1, length(val)));
                end

                tmp = val(pos1);
                val(pos1) = val(pos2);
                val(pos2) = tmp;

                if (~has_loop(val))
                    break;
                end
            end

            pop{i} = val;
        end
    end

    mutated_pop = pop;
end

function reducted_pop = reduction2(pop, remain)
    reducted_pop = cell(remain,1);
    indeces = randperm(length(pop), remain);
    for i = 1:remain
        reducted_pop{i} = pop{indeces(i)};
    end
end

% Редукция
function reducted_pop = reduction(f, pop, len)
    valind = configureDictionary("double", "cell");
    cnt = 0;

    arr = {{}};

    for i = 1:length(pop)
        val = f(pop{i});
        if cnt > 0
            if (valind.isKey(val))
                arr = valind.lookup(val);
            else
                arr = {{}};
            end
        else
            arr = {{}};
        end

        subarr = arr{1};
        subarr{length(subarr) + 1} = pop{i};
        arr{1} = subarr;

        % if (cnt == 0)
        %     val2ind = dictionary(val, arr);
        % end
        valind = valind.insert(val, arr);
        cnt = cnt + 1;
    end

    reducted_pop = {};

    while (length(reducted_pop) < len & ~isempty(valind.keys()))
        minval = min(valind.keys);
        arr = valind.lookup(minval);
```

```
        unique = configureDictionary("double", "double");

        for j = 1:length(arr{1})

            ind = arr{1}{j};

            % is_member = ismember([ind], cell2mat(reducted_pop));

            if (~unique.isKey(ind))
                reducted_pop = [reducted_pop; ind];
                unique = unique.insert(ind, 1);
            end
        end
        valind = valind.remove(minval);
    end
end


% Репродукция
function intermediate_population = reproduction(population, f)

    pop_size = length(population);

    % Оператор репродукции
    intermediate_population = cell(pop_size, 1);

    individ_values = zeros(pop_size, 1);

    % сумма всех значений и значение для каждой особи
    for i = 1:pop_size
        val = f(population{i});
        individ_values(i) = val;
    end

    values_sum = sum(individ_values);

    % все особи одинаковы
    if values_sum == 0
        intermediate_population = population;
        return
    end

    potentials = zeros(pop_size, 1);

    % подсчет потенциала для каждой особи
    % потенциалы отрицательны, т.к. ищем минимум
    for i = 1:length(individ_values)
        val = individ_values(i);
        prob = -(val / values_sum);
        potentials(i) = prob;
    end

    min_potential = min(potentials);

    % сдвигаем потенциалы в положительную часть
    for i = 1:length(individ_values)
        potentials(i) = potentials(i) - min_potential;
    end

    sum_potentials = sum(potentials);

     % выбор такого же кол-ва особей
    for i = 1:length(population)
```

```
        % крутите барабан
        shot = rand_range(0, sum_potentials);
        individ_num = 0;
        tmp_sum = 0;

        % определяем куда попали барабаном
        for j = 1:length(population)
            individ_num = j;
            tmp_sum = tmp_sum + potentials(individ_num);
            if (tmp_sum >= shot)
                break
            end
        end

        % добавляем выбранную особь в промежуточную популяцию
        individ = population{individ_num};
        intermediate_population{i} = individ;
    end
end


% Скрещивание
function pop = crossover(parents, prob, cros)
    len = length(parents);
    pop = cell(0, 1);

    for i = 1:len
        p1 = parents{round(rand_range(1, len))};
        p2 = parents{round(rand_range(1, len))};

        if (rand() < prob)
            child = cros(p1, p2);
            pop{length(pop) + 1, 1} = child;
        end
    end
end

function child = alternating_cross(p1, p2)
    while (true)
        child = zeros(1, length(p1));

        is_first = true;

        for i = 1:length(p1)
            if (rand() < 0.5)
                parent = p1;
            else
                parent = p2;
            end

            % Выбираем из родителя ребро
            node = parent(i);
            % node = 0;

            % Если уже есть такое, выбираем другое по возможности
            while (node == 0 || node == i || any(ismember(child,
node)))
                node = parent(round(rand_range(1, length(parent))));
                child(i) = node;
                if (all(ismember(child, parent)))
                    break;
                end
                child(i) = 0;
            end
```

```
            child(i) = node;
            is_first = ~is_first;
        end

        if (has_loop(child) == false)
            break;
        end
    end
end

% кроссовер обмен подтурами
function child = subtour_cross(p1, p2)
    len = length(p1);

    while true

        child = zeros(1, len);
        child_len = 0;

        while true

            if (len - child_len < 1)
                break;
            end

            subtour_from = child_len + 1;
            subtour_len = round(rand_range(1, len - child_len));

            if (rand() < 0.5)
                parent = p1;
            else
                parent = p2;
            end

            subtour = parent(subtour_from:subtour_from + subtour_len -
1);

            is_loop = false;

            for j = 1:length(subtour)
                node = subtour(j);

                % if (has_loop(child))
                if (any(ismember(child, node)))
                    is_loop = true;
                    break;
                end

                child(j + subtour_from - 1) = node;
                child_len = child_len + 1;
            end

            if (is_loop || child_len >= len)
                break;
            end
        end

        if (is_loop)
            vars_dict = configureDictionary("double", "double");

            % Все возможные вершины добавляем в множество
            for i = 1:len
                vars_dict = vars_dict.insert(i, 1);
            end
```

```
            % Удаляем те что уже есть
            for i = 1:child_len
                if (vars_dict.isKey(child(i)))
                    vars_dict = vars_dict.remove(child(i));
                end
            end

            % Рандомно добавляем те что есть
            for j = child_len + 1:len
                keys = vars_dict.keys();
                node = keys(randperm(len - child_len, 1));
                child(j) = node;
                vars_dict = vars_dict.remove(node);
                child_len = child_len + 1;
            end
        end

        if (~has_loop(child))
            break
        end
    end
end

% кроссовер эвристический
function child = heuristic_cross(p1, p2, ways)
    len = length(p1);

    while true
        child = zeros(1, len);

        source = randperm(len, 1);

        is_loop = false;

        for i = 1:len
            dest1 = p1(source);
            dest2 = p2(source);

            way1 = ways(source, dest1);
            way2 = ways(source, dest2);

            if (way1 < way2)
                dest = dest1;
            else
                dest = dest2;
            end

            res_dest = dest;

            while (any(ismember(child, res_dest)))
                % is_loop = true;
                res_dest = randperm(len, 1);
                % break;
            end

            child(source) = res_dest;
            source = res_dest;
        end

        % Рандомно выставляем нули
        for i = 1:len
            if (child(i) == 0)
                dest = randperm(len, 1);
                while (any(ismember(child, dest)))
                    dest = randperm(len, 1);
```

```
                end
                child(i) = dest;
            end
        end

        is_loop = has_loop(child);

        if (~is_loop)
            break;
        end

    end
end

% случайное число в диапазоне
function ret = rand_range(from, to)
    ret = (to-from) .* rand() + from;
end


% parsed = parse_path(path);
% parsed

% Чтение файла координат
function coords = read_coords(s)
    % читаем файл в вектор
    coords = zeros(s, 2);
    fileID = fopen('bayg_coords.txt', 'r');
    formatSpec = '%f';
    a = fscanf(fileID, formatSpec);

    for i = 1:s
        coords(i, 1) = a((i - 1) * 3 + 2);
        coords(i, 2) = a((i - 1) * 3 + 3);
    end
end

% Разбор транспонированной матрицы из файла
function matrix = read_matrix(s)

    % читаем файл в вектор
    transposed_len = s - 1;
    m = zeros(transposed_len);
    fileID = fopen('bayg_t_matrix.txt', 'r');
    formatSpec = '%f';
    a = fscanf(fileID, formatSpec);

    % распределеляем вектор в матрицу
    offset = 0;

    for i = 1:transposed_len
        line_offset = i - 1;
        line_len = transposed_len - line_offset;
        line = a(offset + 1 : offset + line_len);
        m(i, 1:line_len) = line;
        offset = offset + line_len;
    end

    %  транспонируем и перемещаем строки местами
    % t = transpose(m);

    t = m;

    for i = 1:transposed_len
        m(:, i) = t(:, transposed_len - i + 1);
```

```
    end

    matrix = zeros(s);

    %  итоговая матрица: заполняем верхнюю половину
    %  и выставляем нижнюю
    matrix(1:s-1,2:s) = m;

    for i = 1:s
        for j = 1:s
            matrix(j, i) = matrix(i, j);
        end
    end
end

% Разбор транспонированной матрицы из файла
function matrix = read_matrix2(s)

    % читаем файл в вектор
    transposed_len = s - 1;
    m = zeros(s);
    fileID = fopen('bayg_t_matrix.txt', 'r');
    formatSpec = '%f';
    a = fscanf(fileID, formatSpec);

    % распределеляем вектор в матрицу
    offset = 0;

    for i = 1:transposed_len
        line_offset = i - 1;
        line_len = transposed_len - line_offset;
        line = a(offset + 1 : offset + line_len);
        m(i, i+1:s) = line;
        offset = offset + line_len;
    end

    %  транспонируем и перемещаем строки местами
    % t = transpose(m);

    % t = m;
    %
    % for i = 1:transposed_len
    %     m(:, i) = t(:, transposed_len - i + 1);
    % end

    matrix = m;

    %  итоговая матрица: заполняем верхнюю половину
    %  и выставляем нижнюю
    % matrix(1:s-1,2:s) = m;

    for i = 1:s
        for j = 1:s
            matrix(j, i) = matrix(i, j);
        end
    end
end

% Фитнесс-функция: длина пути
function way_length = fitness(comm_path, way_matrix)
    way_length = path_length(parse_path(comm_path), way_matrix);
end


% Найти длину пути
```

```
function way_length = path_length(path, way_matrix)
    way_length = 0;
    prev_node = 0;
    for i = 1:length(path)
        node = path(i);
        if (prev_node > 0)
            way_length = way_length + way_matrix(prev_node, node);
        end
        prev_node = node;
    end
end

% Генерация начальной популяции
function individs = generate_population(len, ind_size)
    individs = cell(len, 1);
    for i = 1:len
        [comm_path] = gen_path(ind_size);
        individs{i} = comm_path;
    end
end

% Генерация вектора соседей и пути
function [comm_path, way] = gen_path(size)
    % Генерируем путь, всегда оканчивающийся на 1
    way = randperm(size - 1) + 1;
    way = [way, 1];

    % По созданному пути генерируем соседей
    comm_path = zeros(1, size);
    source = 1;
    for i = 1:size
        dest = way(i);
        comm_path(source) = dest;
        source = dest;
    end
end

% Преобразование вектора соседей в пути
function way = parse_path(comm_path)
    len = size(comm_path, 2);
    way = zeros(1, len);

    node = 1;

    for i = 1:len
        if (ismember(node, way))
            way = false;
            return
        end

        way(i) = node;

        if (mod(node, 1) ~= 0)
            a=1;
        end

        node = comm_path(node);
    end

    way = [way, node];
end

% Проверка есть ли  замкнутый цикл
function is_loop = has_loop(comm_path)
    len = size(comm_path, 2);
```

```
    way = zeros(1, len);

    node = 1;

    for i = 1:len
        node = comm_path(node);

        if (node == 0)
            is_loop = false;
            return
        end

        if (node == 1 && i < len)
            is_loop = true;
            return
        end

        if (any(ismember(node, way)))
            is_loop = true;
            return
        end

        way(i) = node;
    end

    way = [way, node];

    is_loop = false;
     end
```

Результаты выполнения программы

Дано известное оптимальное решение:

Маршрут:

[1,28,6,12,9,26,3,29,5,21,2,20,10,4,15,18,14,17,22,11,19,25,7,23,8,27,16,13,24,1].

Длина маршрута составляет 1610.

Путь маршрута показан на рисунке 1.

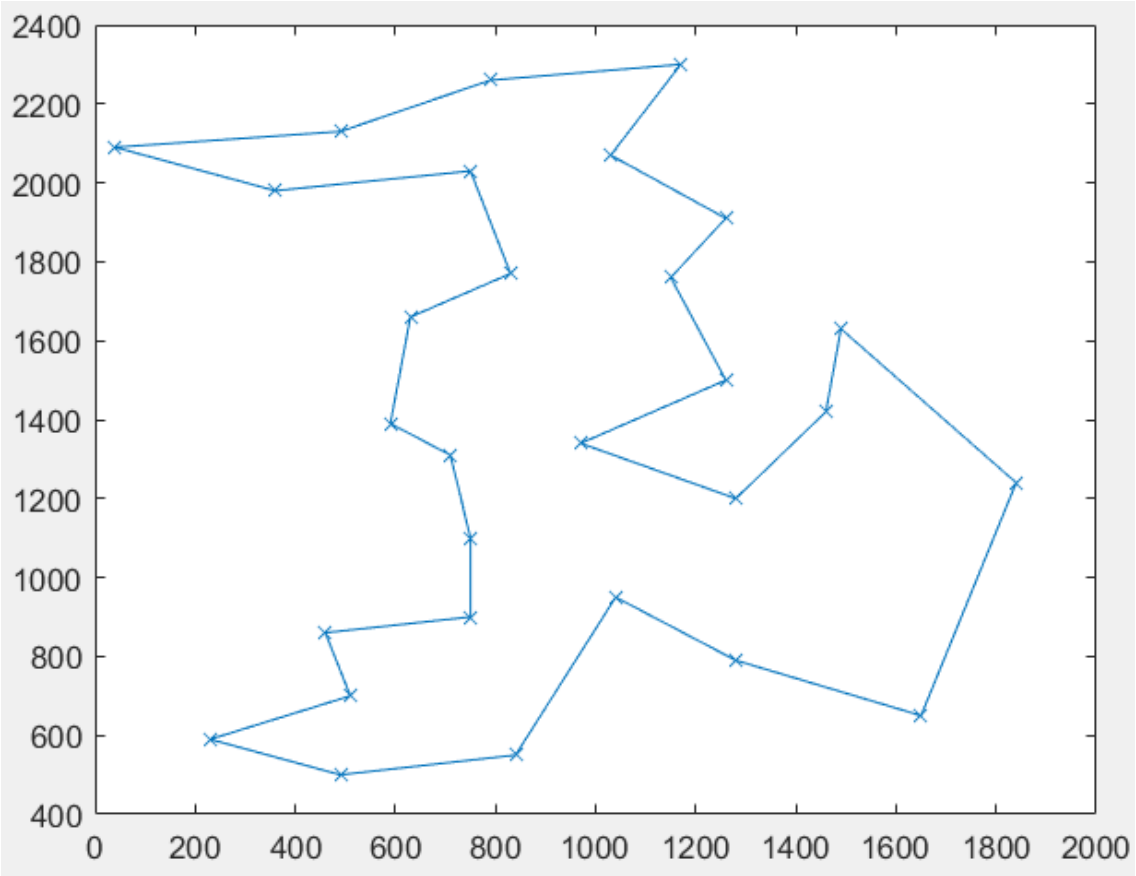

Рисунок 1 Известное оптимальное решение

Далее будут приведены результаты запуска написанной программы.

Используется три оператора кроссинговера: обмен ребрами, обмен подтурами, эвристический кроссовер.

Оператор мутации представляет собой обмен в соседском представлении двух пунктов назначений случайно выбранных ребер таким образом, чтобы не образовалось преждевременного цикла.

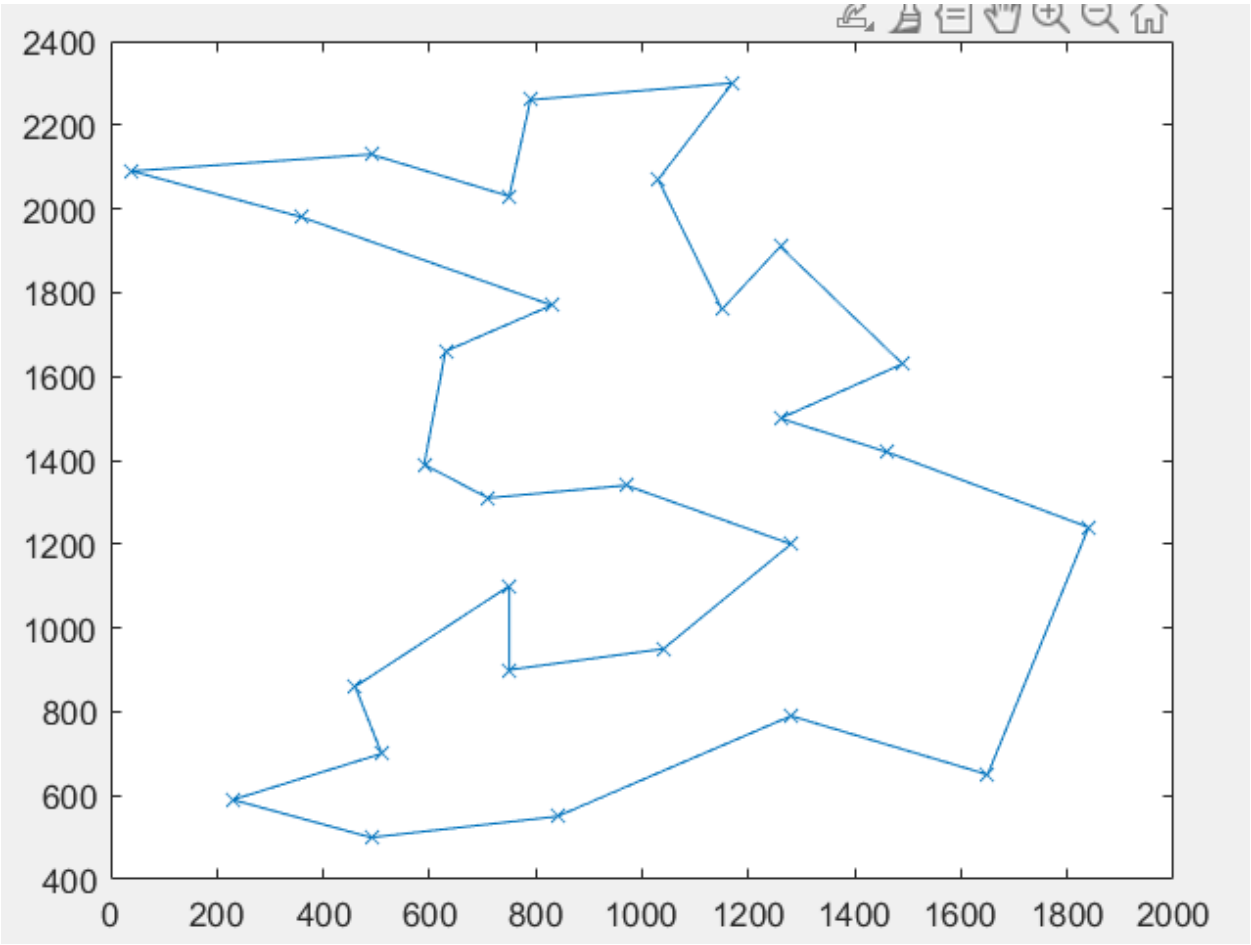

Рисунок 2 Решение обменом ребер

Шаг 10000: лучший маршрут:

[28 21 26 15 9 1 25 24 12 20 22 6 10 18 19 13 14 4 16 2 29 17 7 27 11 5 23 8 3]

[1 28 8 24 27 23 7 25 11 22 17 14 18 4 15 19 16 13 10 20 2 21 29 3 26 5 9 12 6 1]

Длина маршрута: 1657

Вероятность кроссинговера: 0.9

Вероятность мутации: 0.1

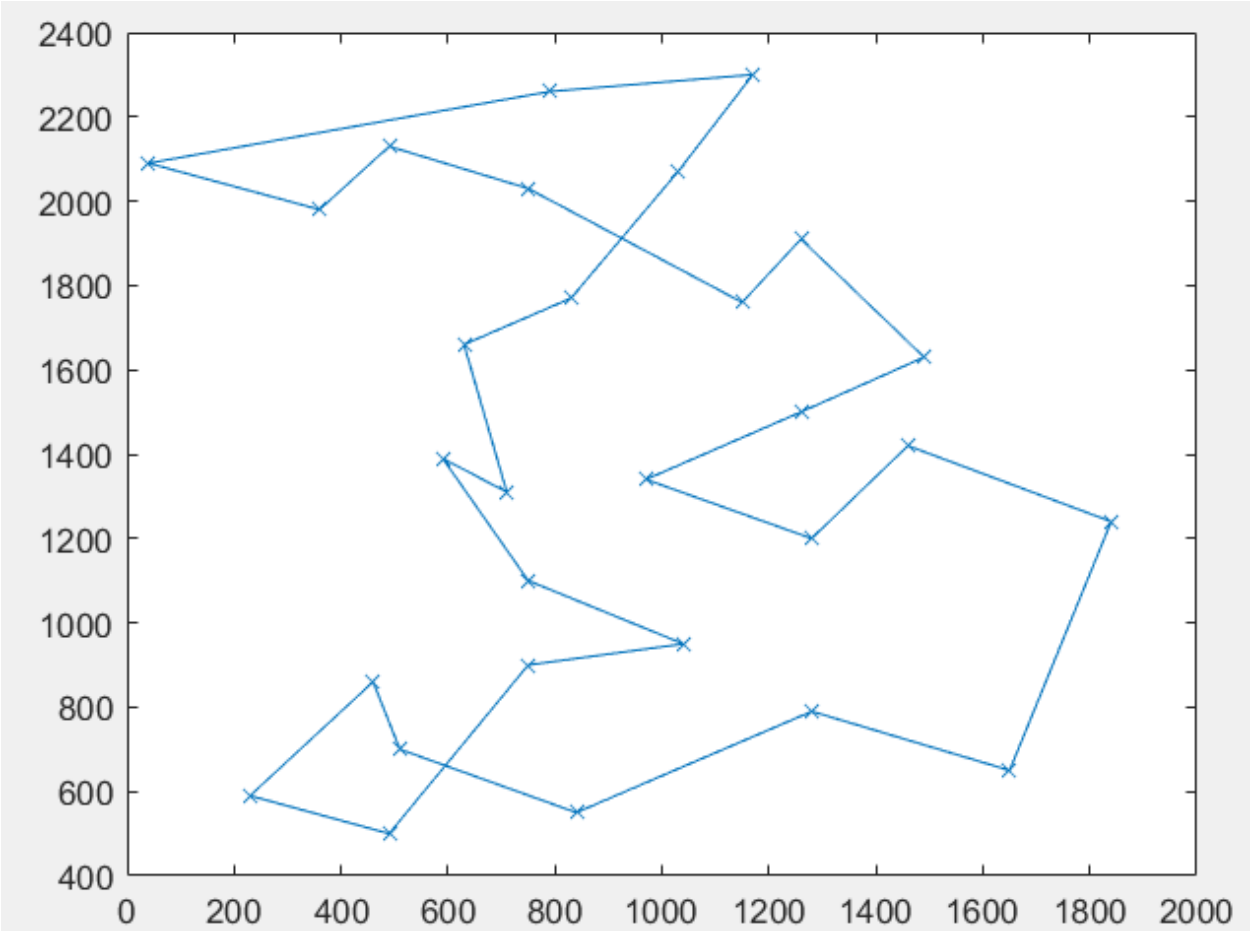

Рисунок 3 Решение обменом подтуров

Шаг 10000: лучший маршрут:

[28 21 29 20 1 12 25 24 3 2 14 9 16 18 19 27 22 17 4 10 6 15 7 13 11 5 23 8 26]

[1 28 8 24 13 16 27 23 7 25 11 14 18 17 22 15 19 4 20 10 2 21 6 12 9 3 29 26 5 1]

Длина маршрута: 1793

Вероятность кроссинговера: 0.9

Вероятность мутации: 0.1

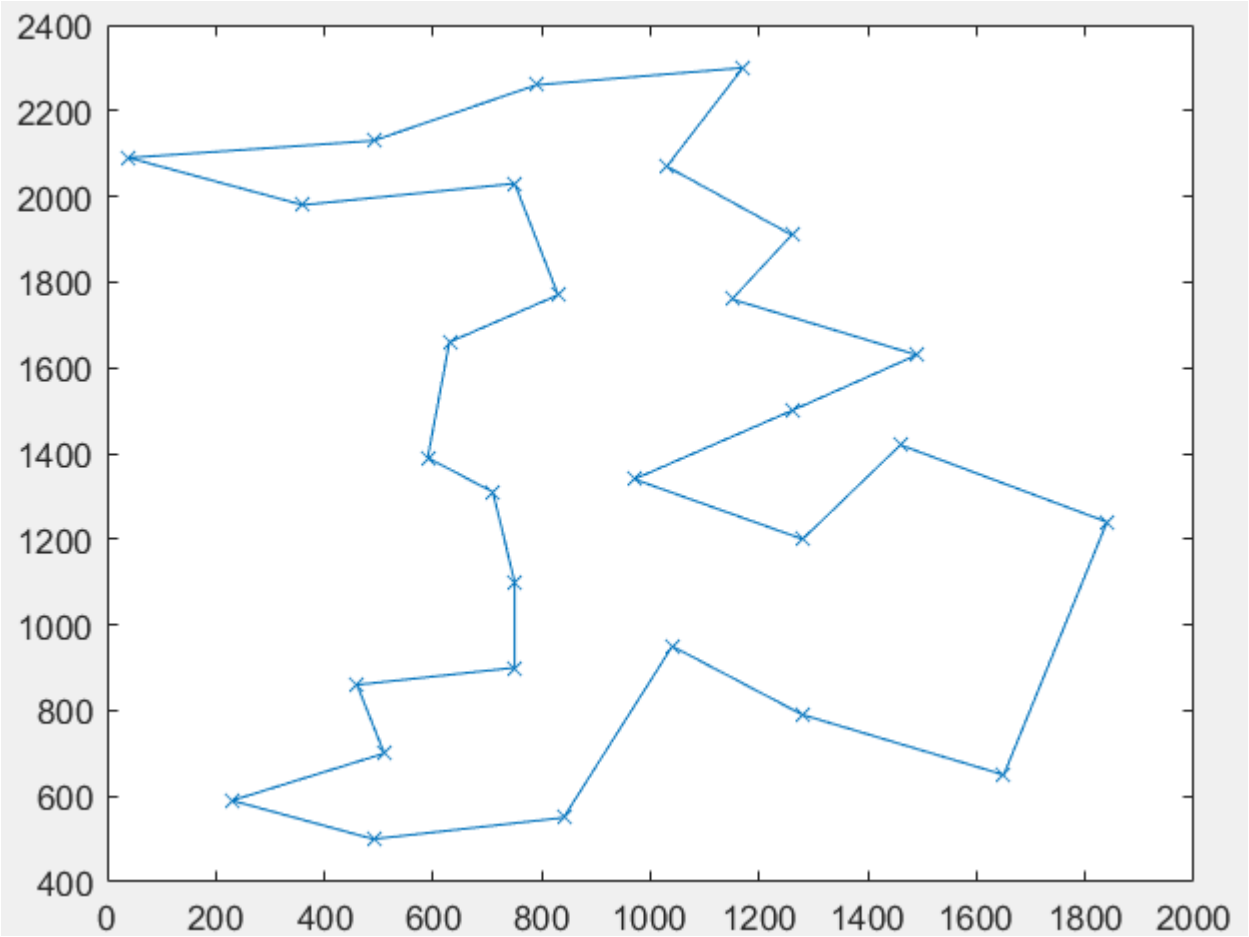

Рисунок 4 Решение эвристическим кроссовером

Шаг 10000: лучший маршрут:

[8 21 26 10 29 28 25 24 12 20 22 6 16 18 4 27 14 15 11 2 5 17 7 13 19 9 23 1 3]

[1 8 24 13 16 27 23 7 25 19 11 22 17 14 18 15 4 10 20 2 21 5 29 3 26 9 12 6 28 1]

Длина маршрута: 1615

Вероятность кроссинговера: 0.9

Вероятность мутации: 0.1

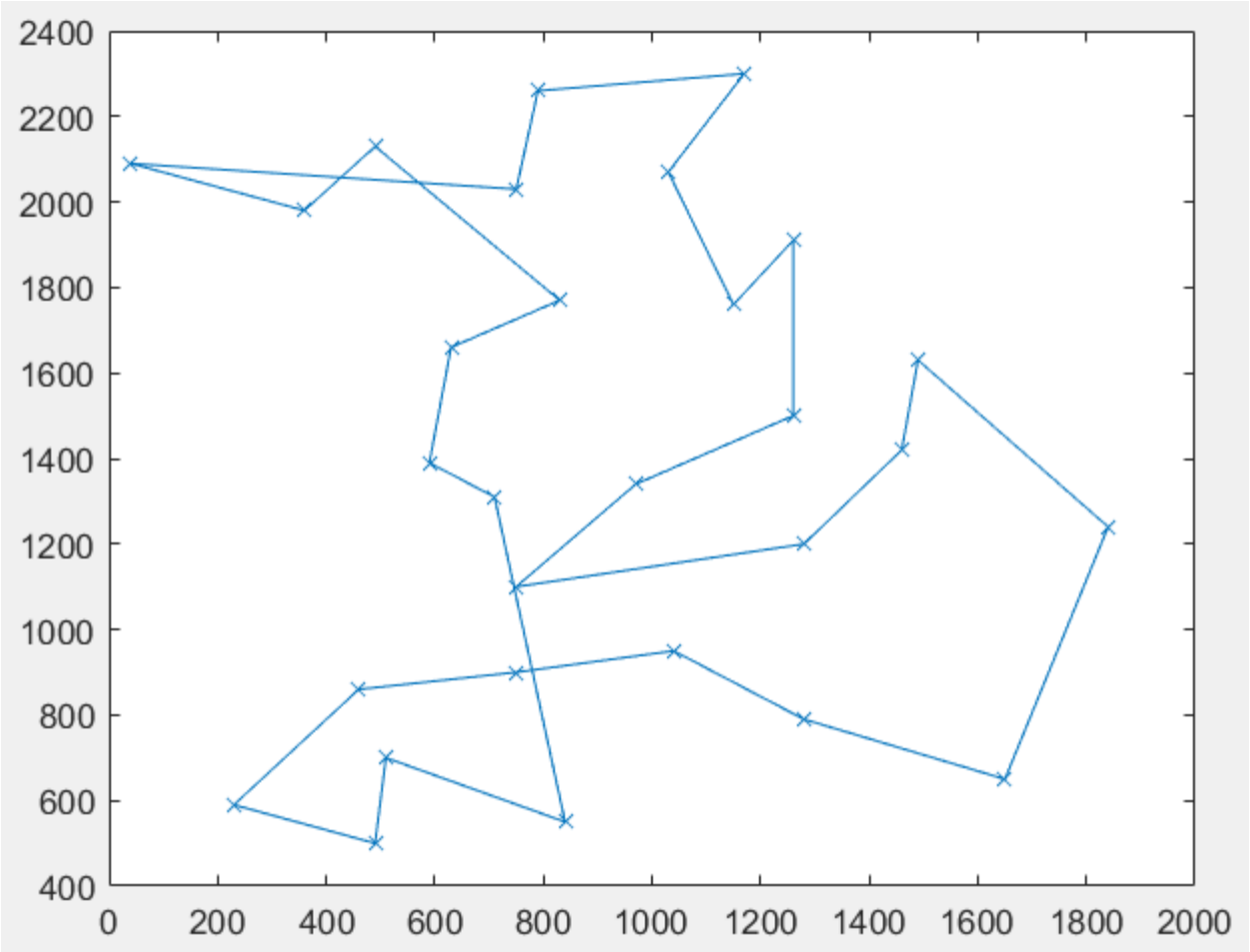

Рисунок 5 Решение обменом ребер

Решение обменом ребер с вероятностью мутации 0.001:

Шаг 10000: лучший маршрут:

[28 21 5 16 9 1 25 23 12 20 10 6 4 11 18 27 22 17 15 2 26 14 7 13 19 29 8 24 3]

[1 28 24 13 4 16 27 8 23 7 25 19 15 18 17 22 14 11 10 20 2 21 26 29 3 5 9 12 6 1]

Длина маршрута: 1829

Вероятность кроссинговера: 0.9

Вероятность мутации: 0.001

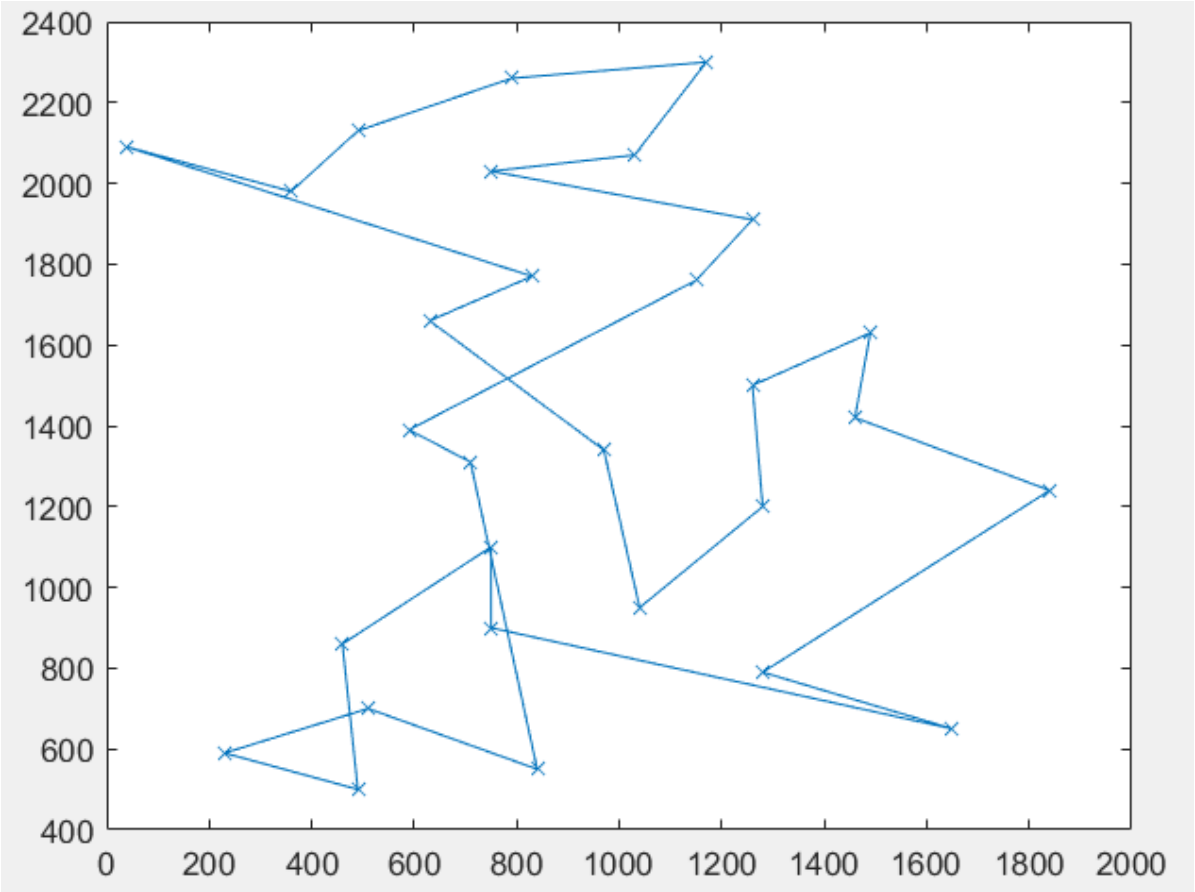

Рисунок 6 Решение обменом подтуров

Решение обменом подтуров с вероятностью мутации 0.001:

Шаг 10000: лучший маршрут:

[20 21 29 15 28 5 25 24 12 11 14 6 2 17 7 19 22 4 13 10 3 18 27 16 23 9 8 1 26]

[1 20 10 11 14 17 22 18 4 15 7 25 23 27 8 24 16 19 13 2 21 3 29 26 9 12 6 5 28 1]

Длина маршрута: 2058

Вероятность кроссинговера: 0.9

Вероятность мутации: 0.001

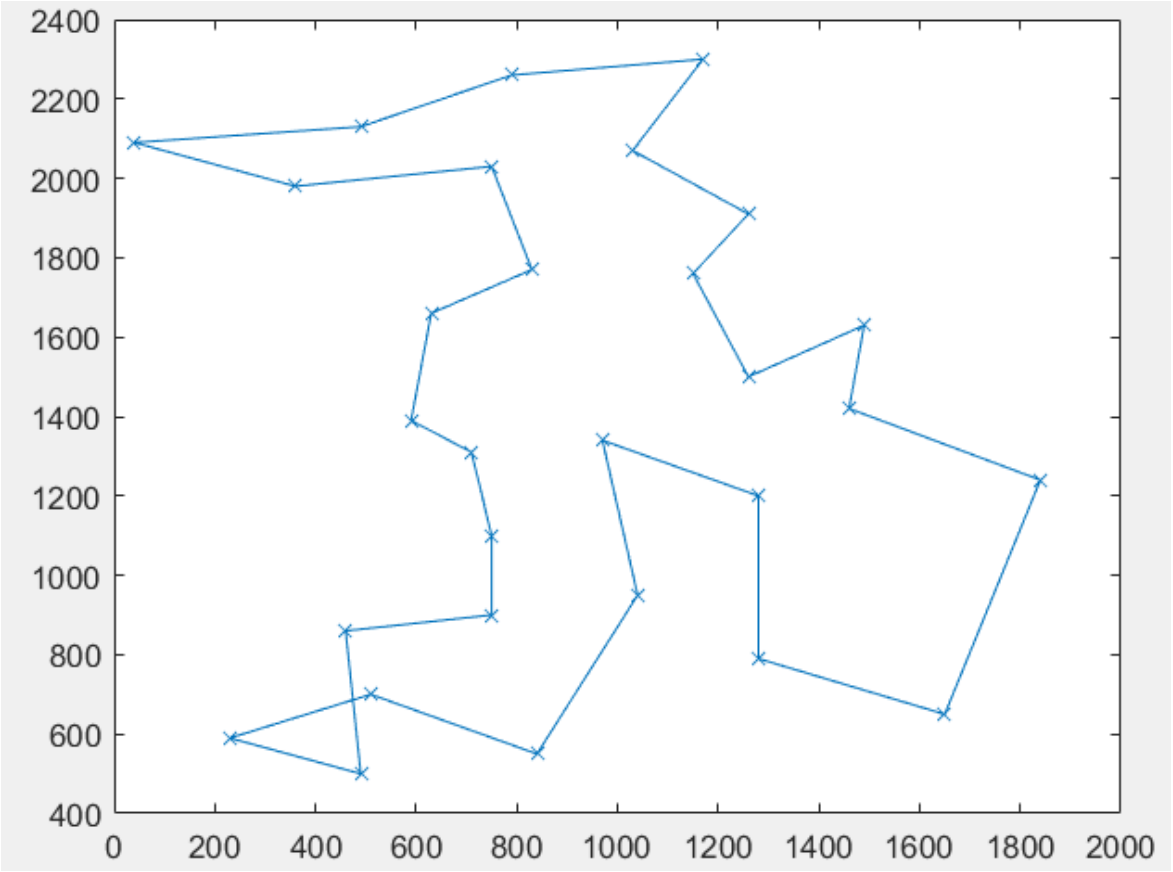

Рисунок 7 Решение эвристическим кроссовером

Решение эвристическим кроссовером с вероятностью мутации 0.001:

Шаг 10000: лучший маршрут:

[24 21 26 10 29 28 25 27 12 20 14 6 19 17 4 13 22 15 11 2 5 18 7 8 16 9 23 1 3]

[1 24 8 27 23 7 25 16 13 19 11 14 17 22 18 15 4 10 20 2 21 5 29 3 26 9 12 6 28 1]

Длина маршрута: 1669

Вероятность кроссинговера: 0.9

Вероятность мутации: 0.001

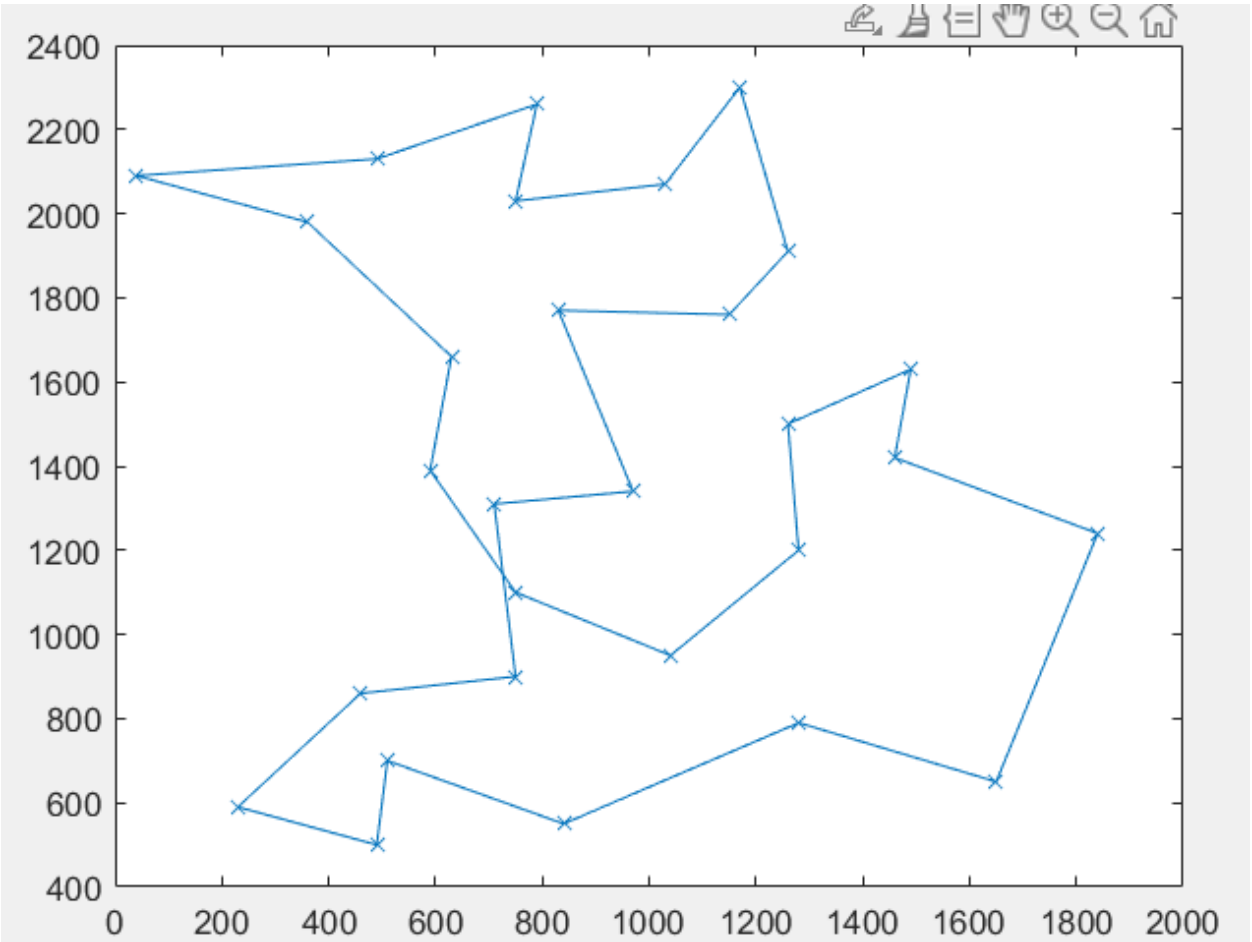

Рисунок 8 Решение обменом ребрами

Решение обменом ребрами с вероятностью кроссинговера 0.5:

Шаг 10000: лучший маршрут:

[21 29 26 20 6 12 23 24 5 15 25 28 10 11 18 19 22 17 4 2 13 14 27 16 7 9 8 1 3]

[1 21 13 10 15 18 17 22 14 11 25 7 23 27 8 24 16 19 4 20 2 29 3 26 9 5 6 12 28 1]

Длина маршрута: 1756

Вероятность кроссинговера: 0.5

Вероятность мутации: 0.1

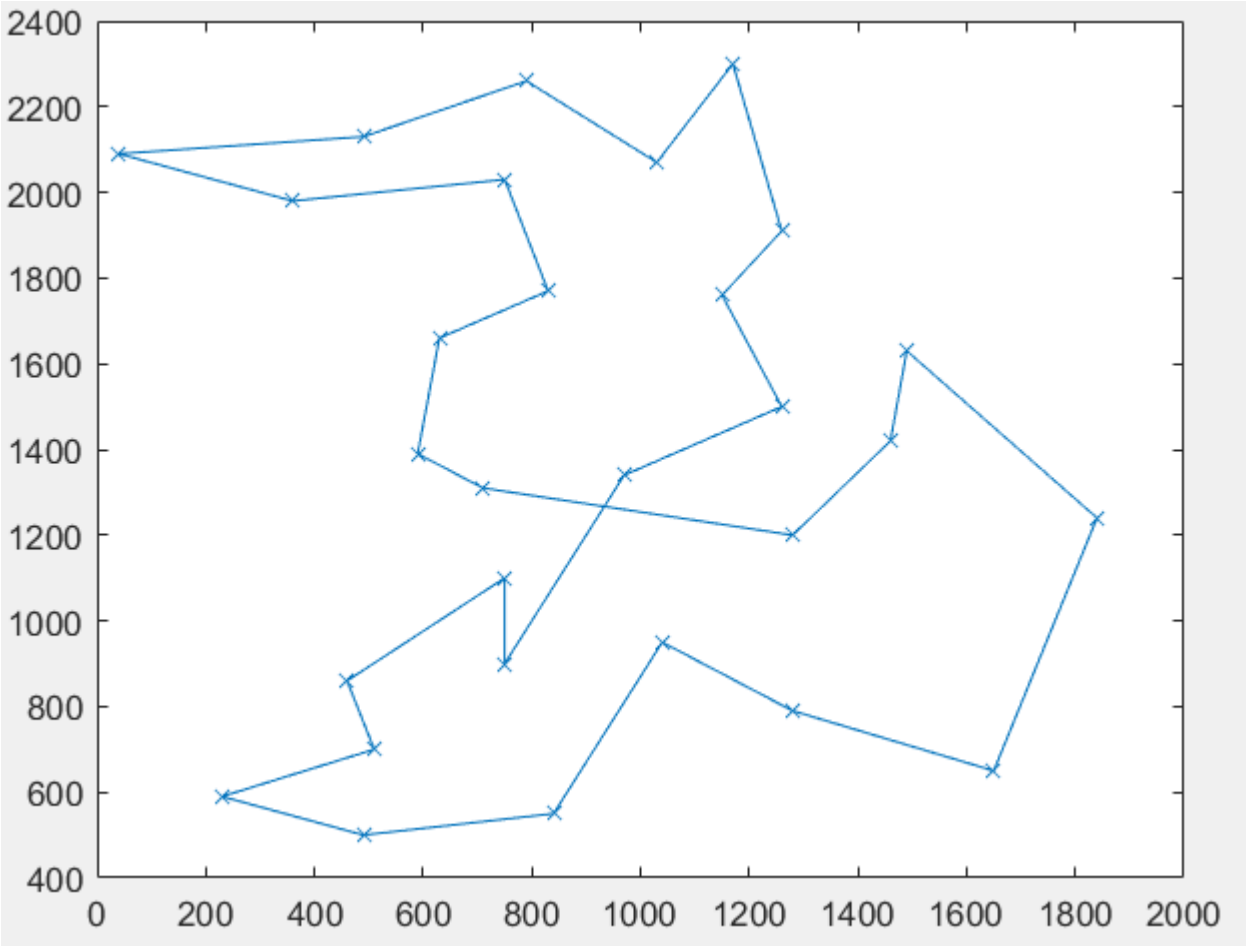

Рисунок 9 Решение обменом подтуров

Решение обменом ребрами с вероятностью кроссинговера 0.5:

Шаг 10000: лучший маршрут:

[28 20 29 15 21 9 25 23 26 16 22 6 24 18 13 27 14 4 11 10 2 17 7 1 19 3 8 12 5]

[1 28 12 6 9 26 3 29 5 21 2 20 10 16 27 8 23 7 25 19 11 22 17 14 18 4 15 13 24 1]

Длина маршрута: 1724

Вероятность кроссинговера: 0.5

Вероятность мутации: 0.1

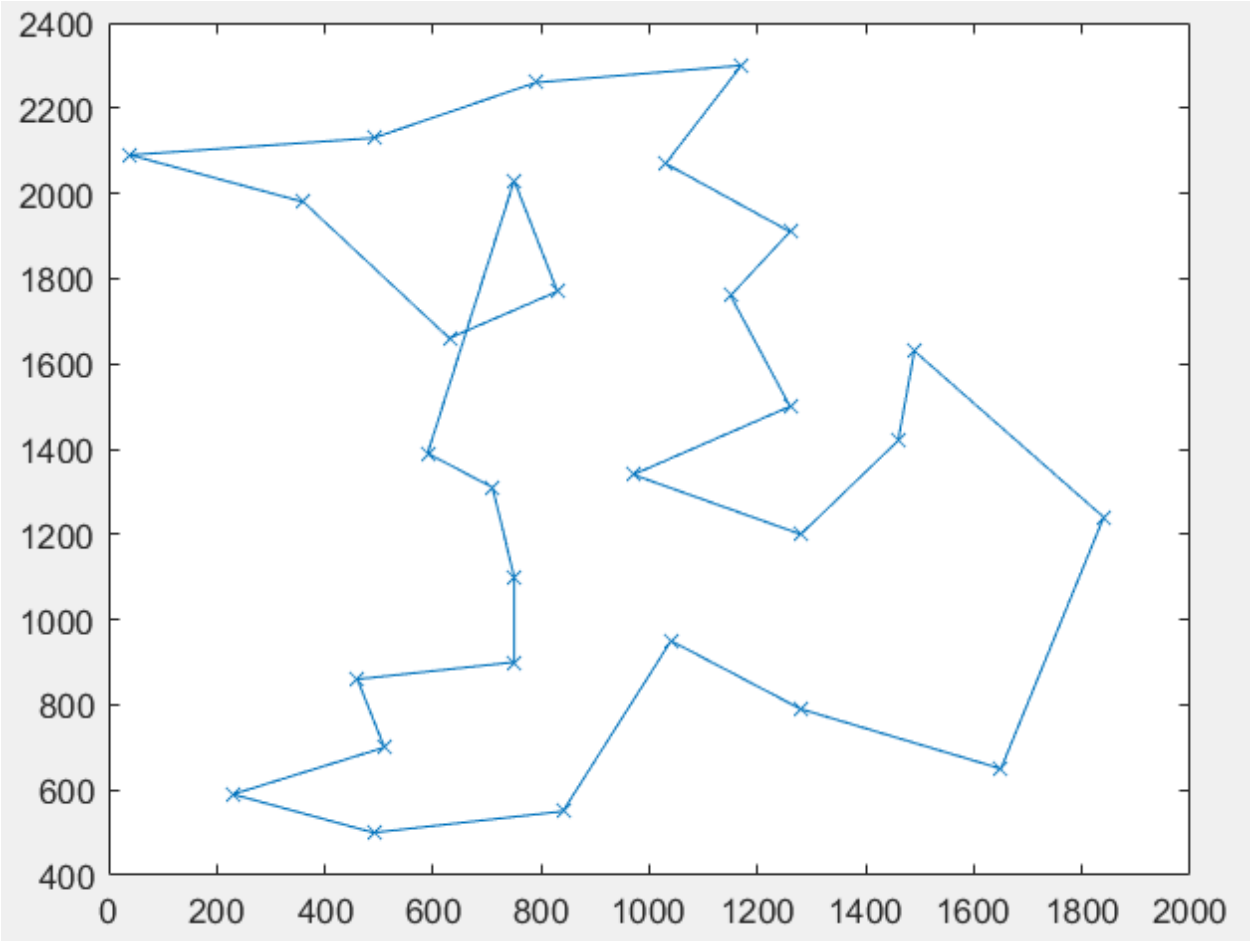

Рисунок 10 Решение эвристическим кроссовером

Решение эвристическим кроссовером с вероятностью кроссинговера 0.5:

Шаг 10000: лучший маршрут:

[28 21 29 15 20 12 23 27 26 4 19 9 24 17 18 13 22 14 25 10 5 11 8 1 7 3 16 6 2]

[1 28 6 12 9 26 3 29 2 21 5 20 10 4 15 18 14 17 22 11 19 25 7 23 8 27 16 13 24 1]

Длина маршрута: 1685

Вероятность кроссинговера: 0.5

Вероятность мутации: 0.1

Наилучшие показатели показал эвристический алгоритм кроссинговера, который во всех трех тестах показал результат 1615, который больше оптимального длиной 1610 всего на 5 единиц. Результат показан при вероятности мутации 0.1 и вероятности скрещивания 0.9.

Кроссовер обмена ребрами по результатам запусков оказался вторым по эффективности, а кроссовер обмена подтурами показал наименее эффективную работу.

По результатам тестов можно заключить, что уменьшение вероятности мутации и уменьшение вероятности скрещивания приводят к ухудшению приближения решения, найденного за отведенное кол-во итераций, к известному оптимальному решению.

Стоит отметить высокую важность мутации – при резком уменьшении её вероятности результаты заметно снизились.

Запуски производились с размером популяции 100 особей.

Ответ на контрольный вопрос

4. Опишите структуру ГА для решения комбинаторных задач.

В комбинаторной задаче объектом операций скрещивания является какая-либо последовательность элементов определенной длины, имеющая известное множество возможных комбинаций. Эта последовательность может быть оценена фитнесс-функцией, представляя собой критерий оптимальности.

ГА для решения комбинаторной задачи принимает такую последовательность в виде хромосомы и оперирует её элементами в ходе операторов репродукции, кроссинговера, мутации. При этом реализация этих операторов может быть различной, в зависимости от типа представления данных в последовательности относительно решаемой задачи.

Цель работы ГА для решения комбинаторных задач – получить комбинацию, максимально приближенную к оптимальному решению.

Вывод

В ходе выполнения третьей лабораторной работы была написана программа в среде MATLAB для поиска оптимального решения задачи коммивояжера в виде решения комбинаторной задачи генетическим алгоритмом.

Используется три оператора кроссинговера: обмен ребрами, обмен подтурами, эвристический кроссовер.

Используется оператор мутации, который обменивает случайно выбранные элементы последовательности.

Приведены графики решения в 3-размерном пространстве для размерности X=2, в том числе промежуточные.

Проведены тестовые запуски с применением различных кроссоверов и изменением параметров алгоритма, сделаны выводы.

Дан ответ на контрольный вопрос согласно варианту.