МГТУ им. Н.Э.Баумана

Лабораторная работа №6

По курсу: "Анализ алгоритмов"

Муравьиный алгоритм

Работу выполнил: Луговой Дмитрий, ИУ7-51Б

Преподаватель: Волкова Л.Л.

Москва, 2019

Оглавление

Введение

Задача коммивояжера формулируется как задача поиска минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе с n вершинами. Содержательно вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояния (длины) или стоимости проезда.

Цель работы: изучить муравьиный алгоритм по материалам решения задачи коммивояжера.

Задачи работы

Задачами данной лабораторной являются:

1) Описать методы решения.
2) Реализовать алгоритм и описать полученные результаты.
3) Выбрать класс данных и составить набор данных.
4) Провести параметризацию метода на основе муравьиного алгоритма для выбранного класса данных.
5) Провести сравнительный анализ двух методов.
6) Дать рекомендации о применимости метода решения задачи коммивояжера на основе муравьиного алгоритма.

1 Аналитическая часть

В данном разделе содержится описание задачи коммивояжера и методы её решения.

1.1 Задача коммивояжера

В общем случае задача коммивояжера (странствующего торговца) может быть сформулирована следующим образом: найти самый выгодный (самый короткий, самый дешевый, и т.п.) маршрут, начинающийся в исходном городе и проходящий ровно один раз через каждый из указанных городов, с последующим возвратом в исходный город.

Проблему коммивояжера можно представить в виде модели на графе, то есть, используя вершины и ребра между ними. Таким образом, M вершин графа соответствуют M городам, а ребра (i, j) между вершинами i и j — пути сообщения между этими городами. Каждому ребру (i, j) можно сопоставить критерий выгодности маршрута $c_{ij} \geq 0$,который можно понимать как, например, расстояние между городами, время или стоимость поездки. В целях упрощения задачи и гарантии существования маршрута обычно считается, что модельный граф задачи является полностью связным, то есть, что между произвольной парой вершин существует ребро.

Гамильтоновым циклом называется маршрут, включающий ровно по одному разу каждую вершину графа. Таким образом, решение задачи коммивояжёра — это нахождение гамильтонова цикла минимального веса в полном взвешенном графе.

1.2 Стандартный алгоритм

Пусть дано M - число городов, D - матрица смежности, каждый элемент которой - вес пути из одного города в другой. Существует метод грубой силы решения поставленной задачи, а именно полный перебор всех возможных гамильтоновых циклов в заданном графе с нахождением минимального по весу. Этот метод гарантированно даст идеальное решение (глобальный минимум по весу). Однако стоит учитывать, что сложность такого алгоритма составляет M! и время выполнения программы, реализующий такой подход, будет расти экспоненциально в зависимости от размеров входной матрицы.

1.3 Муравьиный алгоритм

Поскольку на практике чаще всего необходимо получить решение как можно быстрее, при этом требуемое решение не обязательно должно быть наилучшем, был разработан ряд методов, называемых эвристическими, которые решают поставленную задачу за гораздо меньшее время, чем метод полного перебора. В основе таких методов лежат принципы из окружающего мира, которые в дальнейшем могут быть формализованы.

Одним из таких методов является муравьиный метод. Он применим к решению задачи коммивояжера и основан на идее муравейника. Модель данного метода: у муравья есть 3 чувства:

- зрение (муравей может оценить длину ребра);
- обоняние (муравей может унюхать феромон - вещество, выделяемое муравьем, для коммуникации с другими муравьями);
- память (муравей запоминает свой маршрут).

Благодаря введению обоняния между муравьями возможен непрямой обмен информацией.

Введем вероятность $P_{k,ij}(t)$ выбора следующего города j на маршруте муравьем k, который в текущий момент времени t находится в городе i.

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

где

$\tau_{i,j}$ – феромон на ребре ij;

$\eta_{i,j}$ – привлекательность города j;

α – параметр влияния длины пути;

β – параметр влияния феромона.

Очевидно, что при $\beta = 0$ алгоритм превращается в классический жадный алгоритм, а при $\alpha = 0$ он быстро сойдется к некоторому субоптимальному решению. Выбор правильного соотношения параметров является предметом исследований, и в общем случае производится на основании опыта.

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах феромон, обратно пропорциональный длине пройденного пути:

$$\Delta\tau_{k,ij} = \frac{Q}{L_k} \tag{1.2}$$

где

Q – количество феромона, переносимого муравьем;

L_k – стоимость k-го пути муравья (обычно длина).

После окончания условного дня наступает условная ночь, в течение которой феромон испаряется с ребер с коэффициентом ρ. Количество феромона на следующий день вычисляется по следующей формуле:

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \Delta\tau_{i,j}(t), \tag{1.3}$$

где

$\rho_{i,j}$ – доля феромона, который испарится;

$\tau_{i,j}(t)$ – количество феромона на дуге ij;

$\Delta\tau_{i,j}(t)$ – количество отложенного феромона.

Таким образом, псевдокод муравьиного алгоритма можно представить так:

1. Ввод матрицы расстояний D, количества городов M;
2. Инициализация параметров алгоритма — α β, Q, $tmax$, ρ;
3. Инициализация ребер — присвоение "привлекательности" η_{ij} и начальной концентрации феромона τ_{start};
4. Размещение муравьев в случайно выбранные города без совпадений;
5. Инициализация начального кратчайшего маршрута L_p = null и определение длины кратчайшего маршрута L_{min} = inf;
6. Цикл по времени жизни колонии t = 1, tmax;
    (a) Цикл по всем муравьям k = 1, M;
        i. Построить маршрут $T_k(t)$ по правилу 1.1 и рассчитать длину получившегося маршрута $L_k(t)$;
        ii. Обновить феромон на маршруте по правилу 1.2;
        iii. Если $L_k(t) < L_{min}$, то $L_{min} = L_k(t)$ и $L_p = T_k(t)$;
    (b) Конец цикла по муравьям;
    (c) Цикл по всем ребрам графа;
        i. Обновить следы феромона на ребре по правилу 1.3;
    (d) Конец цикла по ребрам;
7. Конец цикла по времени;
8. Вывести кратчайший маршрут L_p и его длину L_{min}.

1.4 Вывод

Таким образом, существуют две группы методов для решения задачи коммивояжера - точные и эвристические. К точным относится метод полного перебора, к эвристическим - муравьиный метод. Применение муравьиного алгоритма обосновано в тех случаях, когда необходимо быстро найти решение или когда для решения задачи достаточно получения первого приближения. В случае необходимости максимально точного решения используется алгоритм полного перебора.

2 Конструкторская часть

В этом разделе содержатся схемы алгоритмов решения задачи коммивояжера.

2.1 Схемы алгоритмов

На рисунках 2.1 и 2.2 представлена схема муравьиного алгоритма.

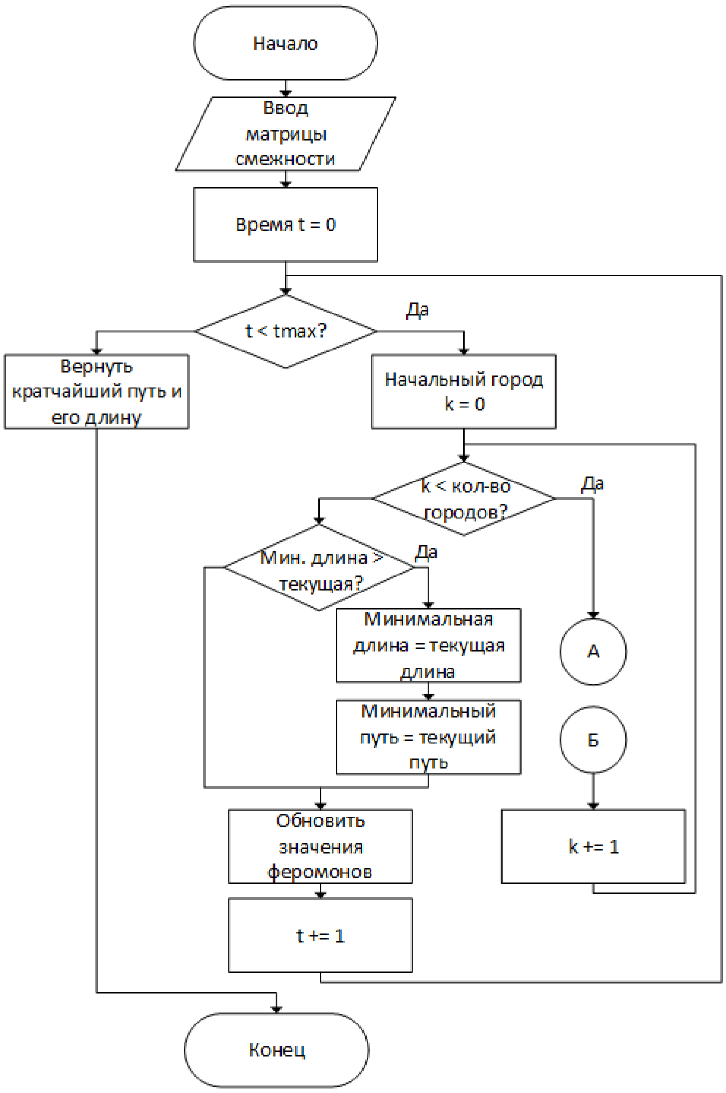

Рис. 2.1: Схема муравьиного алгоритма

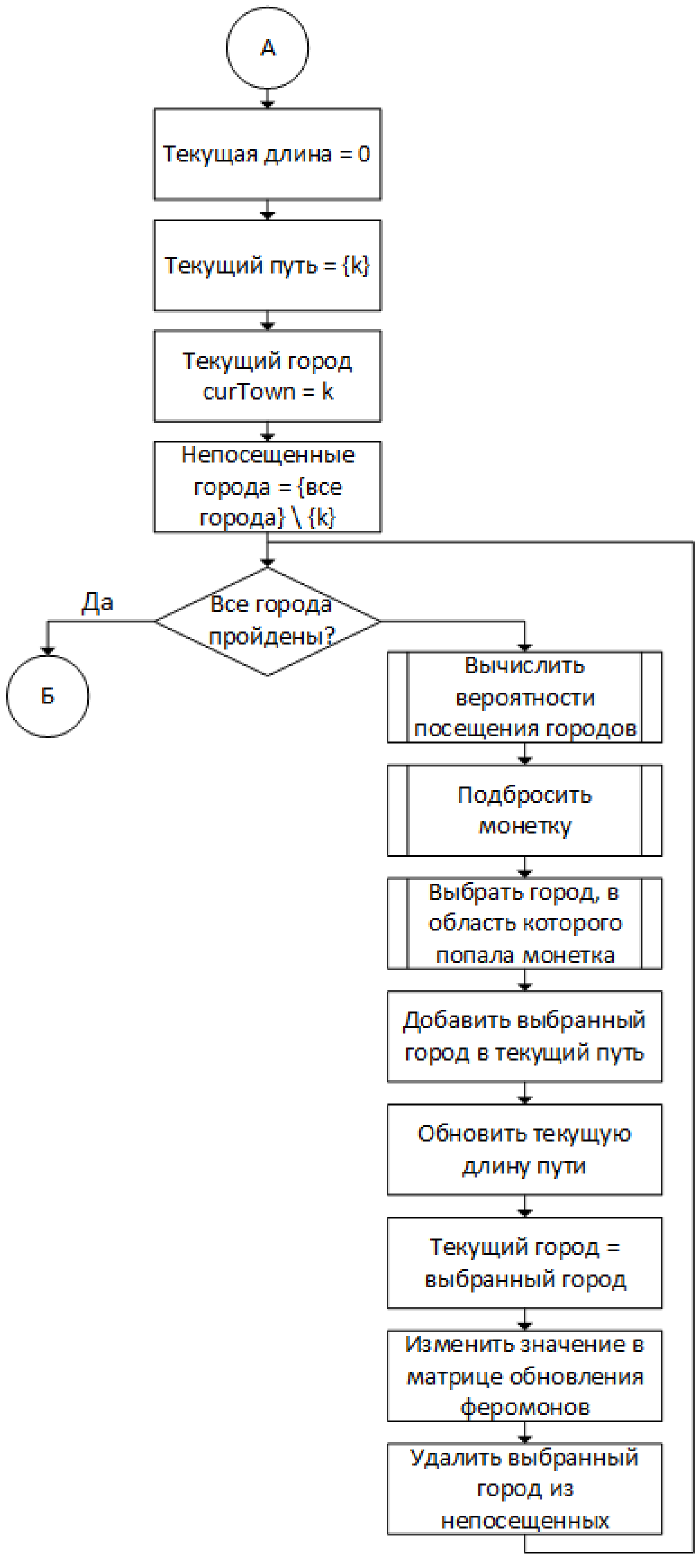

Рис. 2.2: Схема муравьиного алгоритма(продолжение)

На рисунке 2.3 представлена схема основной части алгоритма полного перебора.

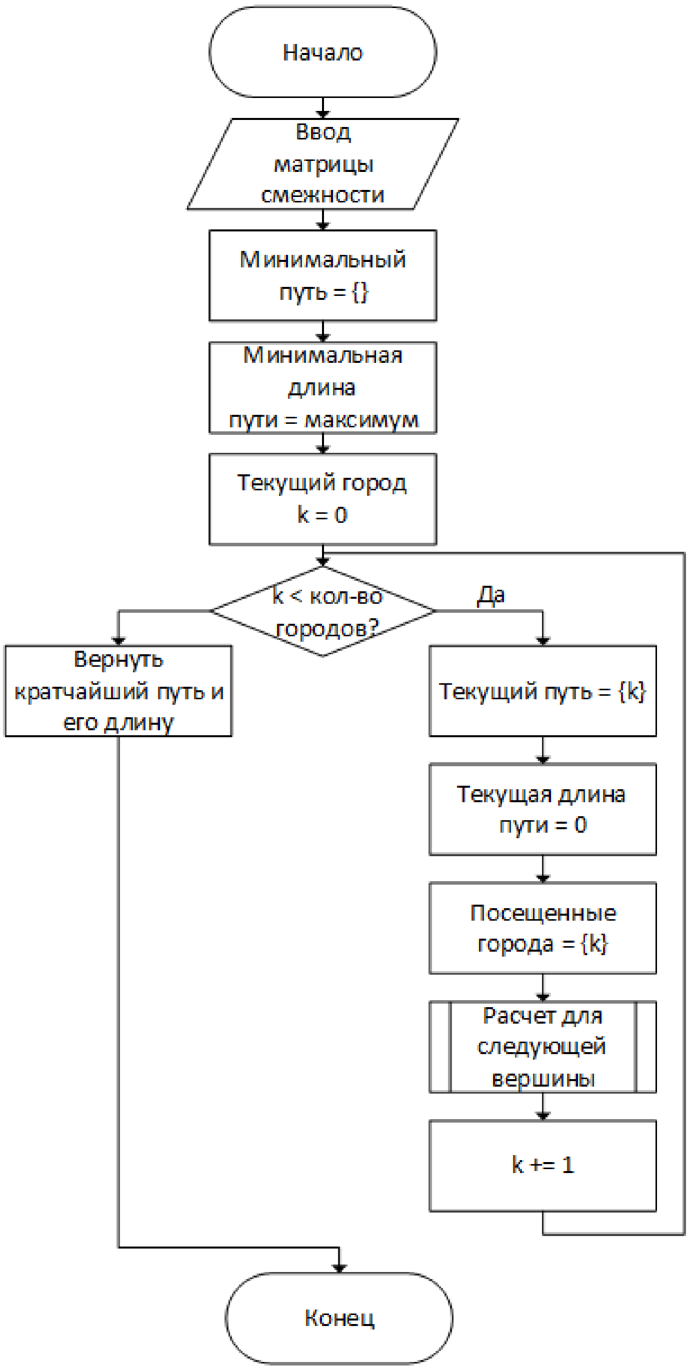

Рис. 2.3: Схема основной части алгоритма полного перебора

На рисунке 2.4 представлена схема рекурсивной части алгоритма полного перебора.

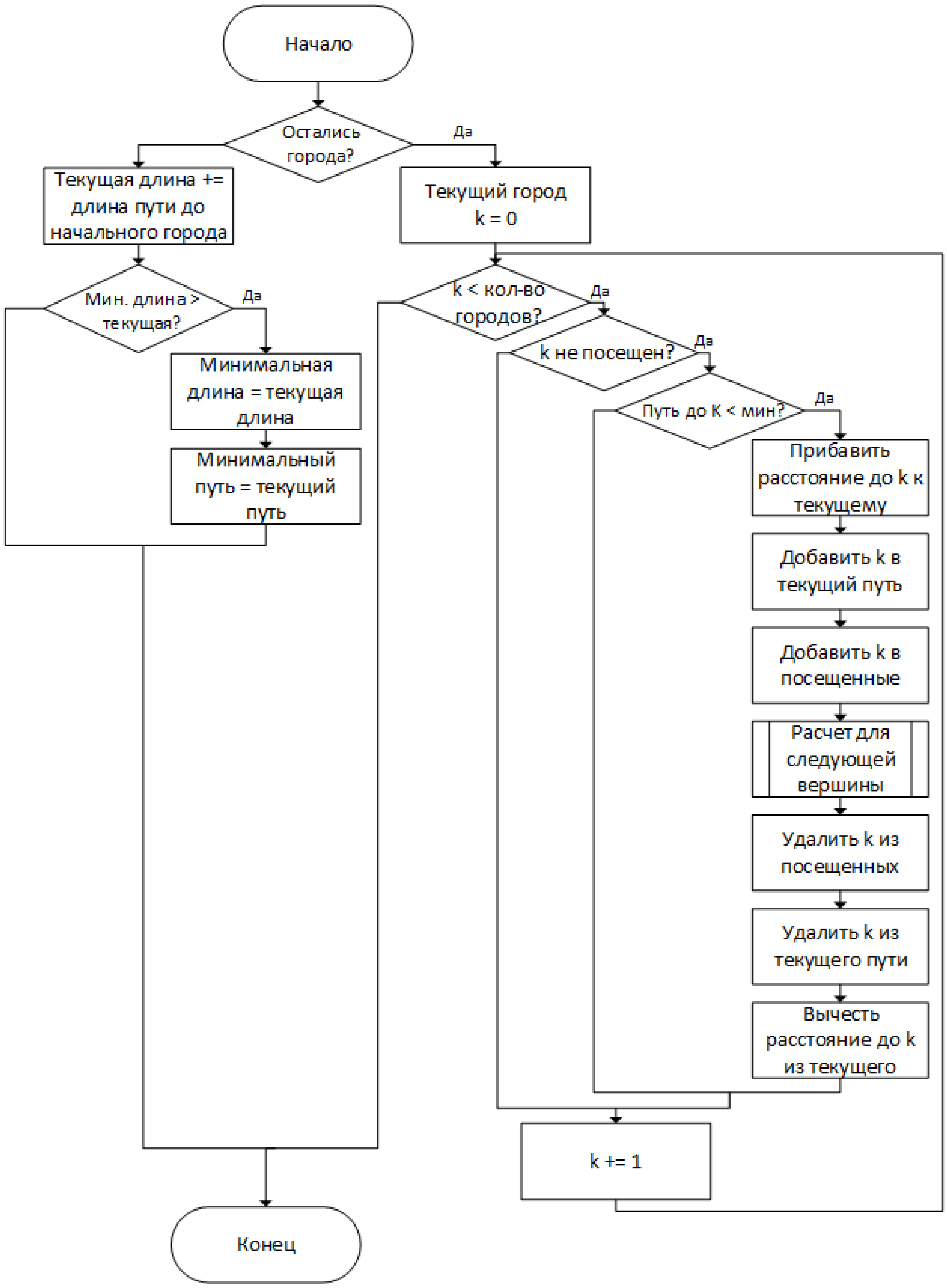

Рис. 2.4: Схема рекурсивной части алгоритма полного перебора

2.2 Вывод

Как видно из схем алгоритмов, количество блоков операций над данным в алгоритме полного перебора меньше, чем в муравьином. Это объясняется более сложной моделью данных в случае муравьиного метода. При этом стоит отметить, что в стандартный алгоритм полного перебора внесено одно изменение, а именно проверка текущей длины пути: если она уже больше длины минимального на текущий момент маршрута, то дальше маршрут по этому пути не строится. Такая модификация позволяет увеличить быстродействие программы.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода

3.1 Требования к ПО

Программа на вход получает матрицу смежности графа. Выход программы: последовательность целых чисел - минимальный по стоимости маршрут и действительное число - суммарная стоимость этого маршрута.

3.2 Средства реализации

Для реализации представленных алгоритмов был выбран язык C++. Время работы алгоритмов было замерено с помощью функции system_clock() из библиотеки chrono. Для тестирования использовался компьютер на базе процессора Intel Core i5 (4 физических ядра, 8 логических).

3.3 Листинги кода

В процессе разработки программы был спроектирован шаблон класса Matrix, служащий для работы с матрицей смежности, а также с матрицами для хранения количества феромона на ребрах.

В Листинге 3.1 показана реализация муравьиного алгоритма.

Листинг 3.1: Функция муравьиного поиска кратчайшего пути

```cpp
Route ant(const Matrix<double> &distances, const int &tMax, const double &
    alpha, const double &p)
{
    const size_t nTowns = distances.size();
    // Pheromone coefficient
    const double q = distances.average() * nTowns;
    const double betta = 1 - alpha;
    // Attractions of the towns
    Matrix <double> attractions(distances);
    attractions.inverse();
    // Pheromones between towns
    Matrix<double> teta(nTowns, START_TETA);
```

```cpp
    // Pheromone daily change
    Matrix<double> deltaTeta(nTowns);

    Route minRoute;
    minRoute.length = -1;
    std::vector<double> probabilities(nTowns, 0.0);

    for (int t = 0; t < tMax; t++)
    {
        // Day
        deltaTeta.zero();
        // Ants loop
        for (int k = 0; k < nTowns; k++)
        {
            std::vector<int> curPath = {k};
            double curLength = 0;
            int curTown = k;

            // Ants path
            std::vector<int> unvisited = getUnvisited(curPath, nTowns);
            while (curPath.size() != nTowns)
            {
                fill(probabilities.begin(), probabilities.end(), 0.0);

                // Probabilities of visiting towns
                for (const auto &town : unvisited)
                {
                    int index = getIndex(unvisited, town);
                    if (fabs(distances[curTown][town]) > EPS)
                    {
                        double sum = 0;
                        for (auto n : unvisited)
                            sum += pow(teta[curTown][n], alpha) * pow(attractions[curTown][n], betta);
                        probabilities[index] = pow(teta[curTown][town], alpha) * pow(attractions[curTown][town], betta) / sum;
                    }
                    else
                        probabilities[index] = 0;
                }

                // Choosing town
                double coin = tossCoin();
                size_t townIndex = 0;
                double curProbability = 0;
                for (size_t j = 0; j < nTowns; j++)
                {
                    curProbability += probabilities[j];
                    if (coin < curProbability)
                    {
```

```
                        townIndex = j;
                        break;
                    }
                }
                int chosenTown = unvisited[townIndex];
                curPath.push_back(chosenTown);
                curLength += distances[curTown][chosenTown];
                deltaTeta[curTown][chosenTown] += q / curLength;
                curTown = chosenTown;
                unvisited.erase(unvisited.begin() + townIndex);
            }
            // Update path
            curLength += distances[curPath[curPath.size() - 1]][curPath[0]];
            if (minRoute.length < -EPS || (curLength < minRoute.length))
            {
                minRoute.length = curLength;
                minRoute.path = curPath;
            }
        }
        // Night
        teta *= (1.0 - p);
        teta += deltaTeta;
        teta.replaceZero(MIN_TETA);
    }
    return minRoute;
}
```

В Листингах 3.1 и 3.2 представлена реализация алгоритма полного перебора, представленная главной функцией bruteForce и рекурсивной hamilton.

Листинг 3.2: Главная функция алгоритма полного перебора

```
Route bruteForce(const Matrix<double> &distances)
{
    int n = distances.size();
    std::vector<bool> visited(n, false);
    std::vector<int> curPath;
    Route minRoute;
    minRoute.length = DBL_MAX;
    double curLen = 0;
    for (int i = 0; i < n; i++)
    {
        curPath.clear();
        curPath.push_back(i);
        fill(visited.begin(), visited.end(), 0);
        visited[i] = true;
        curLen = 0;
        hamilton(distances, minRoute, curPath, visited, curLen);
    }
    return minRoute;
}
```

Основная функция вызывает в цикле для каждой вершины рекурсивную функцию, в которой проверяются все непосещенные вершины и если такие есть, то эта функция вызывается для следующих вершин пока не будет пройден полноценный маршрут.

Листинг 3.3: Рекурсивная функция алгоритма полного перебора

```
void hamilton(const Matrix<double> &distances,Route &minRote, std::vector<int
    > &curPath, std::vector<bool> &visited, double &curLen)
{
    if (curPath.size() == distances.size())
    {
        double tmp = distances[curPath.back()][curPath[0]];
        if (curLen + tmp < minRote.length)
        {
            minRote.path = curPath;
            minRote.length = curLen + tmp;
        }
        return;
    }
    for (int i = 0; i < distances.size(); i++)
    {
        if (!visited[i])
        {
            double tmp = distances[curPath.back()][i];
            if (curLen + tmp > minRote.length)
                continue;
            curLen += tmp;
            curPath.push_back(i);
            visited[i] = true;
            hamilton(distances, minRote, curPath, visited, curLen);
            visited[i] = false;
            curPath.pop_back();
            curLen -= tmp;
        }
    }
}
```

3.4 Вывод

Были реализованы муравьиный алгоритм и алгоритм полного перебора, решающие задачу коммивояжера, для дальнейшего сравнительного анализа точности и скорости вычислений.

4 Экспериментальная часть

В данном разделе приведены примеры работы программы, проведена параметризация метода решения задачи коммивояжера на основе муравьиного алгоритма, а также проведен сравнительный анализ двух алгоритмов.

4.1 Примеры работы

На рисунках 4.1 - 4.3 приведены примеры работы программы. В консоль выводится входная матрица смежности, длина кратчайшего маршрута и сам маршрут для двух алгоритмов.

```
   | 0  1  2
-------------
0  | 0  5  9
1  | 5  0  5
2  | 9  5  0

Ant result: 19 length
0 1 2
Brute force result: 19 length
0 1 2
```

Рис. 4.1: Пример работы для матрицы 3x3

```
   | 0  1  2  3  4
-------------------
0  | 0  6  2  4  6
1  | 6  0  4  3  4
2  | 2  4  0  8  3
3  | 4  3  8  0  2
4  | 6  4  3  2  0

Ant result: 16 length
3 4 2 0 1
Brute force result: 16 length
0 1 3 4 2
```

Рис. 4.2: Пример работы для матрицы 5x5

```
   | 0  1  2  3  4  5  6  7  8  9
---------------------------------
0  | 0  9  9  2  1  8  8  2  3  3
1  | 9  0  5  9  6  4  6  3  2  6
2  | 9  5  0  5  7  8  1  1  4  4
3  | 2  9  5  0  6  4  3  10 2  1
4  | 1  6  7  6  0  2  5  8  1  3
5  | 8  4  8  4  2  0  1  10 6  5
6  | 8  6  1  3  5  1  0  2  10 10
7  | 2  3  1  10 8  10 2  0  1  5
8  | 3  2  4  2  1  6  10 1  0  4
9  | 3  6  4  1  3  5  10 5  4  0

Ant result: 18 length
4 9 3 0 7 2 6 5 1 8
Brute force result: 17 length
0 4 5 6 2 7 1 8 3 9
```

Рис. 4.3: Пример работы для матрицы 10x10

Как видно из приведенных примеров, с увеличением размерности матрицы, муравьиный алгоритм начинает показывать меньшую точность, в то время как полный перебор находит наилучший вариант в любом случае.

4.2 Параметризация метода

Для проведения экспериментов была использована матрица смежности 10x10, изображенная на рисунке 4.4.

```
   | 0  1  2  3  4  5  6  7  8  9
---------------------------------
0  | 0  4  8  6  8  10 9  5  8  6
1  | 4  0  3  8  7  8  10 4  9  6
2  | 8  3  0  3  6  3  6  10 7  9
3  | 6  8  3  0  2  8  5  3  9  9
4  | 8  7  6  2  0  2  7  6  6  7
5  | 10 8  3  8  2  0  6  3  10 9
6  | 9  10 6  5  7  6  0  1  7  6
7  | 5  4  10 3  6  3  1  0  4  1
8  | 8  9  7  9  6  10 7  4  0  9
9  | 6  6  9  9  7  9  6  1  9  0
```

Рис. 4.4: Матрица для параметризации метода

В каждом эксперименте фиксировались значения α, β, ρ и $tmax$. В течение экспериментов значения α, β, ρ менялись от 0 до 1 с шагом 0.25, $tmax$ от 10 до 300 с шагом 10. Количество повторов каждого эксперимента равнялось 100, результатом проведения эксперимента считалась усредненная разница между длинной маршрута, рассчитанного алгоритмом полного перебора и муравьиным алгоритмом с текущими параметрами.

В таблице 4.1 представлены 10 лучших результатов по наименьшему отклонению от минимального расстояния.

Таблица 4.1: Результаты проведенных экспериментов

Параметр ρ	Параметр α	Параметр tmax	Отклонение от минимального пути
0.25	0.75	290	0.05
0.25	0.75	250	0.09
0.25	0.75	280	0.09
0.25	0.75	300	0.1
0.5	0.75	270	0.1
0.5	0.75	300	0.1
0.25	0.75	240	0.11
0.25	0.75	270	0.11
0.5	0.75	290	0.11
0.25	0.75	220	0.12
0.25	0.75	260	0.12
0.25	0.75	300	0.12
0.5	0.75	250	0.12
0.25	0.75	210	0.13
0.25	0.75	230	0.13
0.5	0.75	280	0.13
0.25	0.75	180	0.15
0.5	0.75	210	0.15
0.5	0.75	240	0.15
0.5	0.75	260	0.16
0.75	0.5	300	0.16
0.5	0.75	230	0.18
0.5	0.75	300	0.18
0.25	0.75	200	0.19
0.5	0.75	220	0.19

Первый столбец представляет собой значения параметра ρ, второй - α, третий - $tmax$ и в четвертом столбце представлено среднее отклонение от длины идеального маршрута, вычисленной алгоритмом полного перебора.

Как видно из представленной таблицы, наилучшим значением настроечного параметра α является значение, равное 0.75, а коэффициента испарения ρ - 0.25. При tmax = 290 достигается наименьшее различие с минимальным путем.

4.3 Сравнение методов

Для сравнения муравьиного алгоритма с алгоримтом полного перебора используются оптимальные параметры муравьиного алгоритма - $\alpha = 0.75$,

$\rho = 0.25$, $tmax = 290$. Размерности матриц меняются от 3 до 15. На рисунке 4.5 представлен график зависимости времени работы каждого из реализованных алгоритмов от размера матрицы смежности.

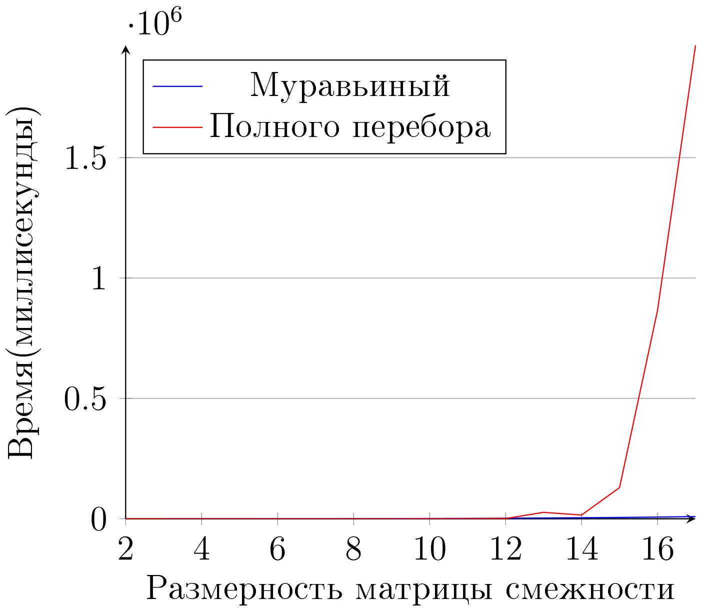

Рис. 4.5: Алгоритмы решения задачи коммивояжера

Как можно наблюдать на графике, время работы алгоритма полного перебора растет экспоненциально в зависимости от размера матрицы смежности,в отличие от муравьиного алгоритма, время которого изменяется практически линейно.

4.4 Вывод

В результате проведенных экспериментов были выявлены оптимальные параметры для муравьиного алгоритма: $\alpha = 0.75$, $\rho = 0.25$, $tmax = 290$. Однако стоит учитывать, что чем больше значение tmax, тем больше вероятность того, что будет найден идеальный маршрут, но при этом будет возрастать время выполнения программы. Также был проведен сравнительный анализ муравьиного алгоритма и алгоритма полного перебора. Алгоритм полного перебора рационально использовать для матриц небольшого размера и в случае, если необходимо получить наименьшее расстояние. В остальных случаях муравьиный алгоритм является более эффективным по времени.

Заключение

В ходе работы были изучены и реализованы основные алгоритмы для решения задачи коммивояжера: муравьиный алгоритм и алгоритм полного перебора. Был проведен их сравнительный анализ, в ходе которого были получены зависимости времени выполнения алгоритмов от размеров входной матрицы расстояний, кроме того, была проведена параметризация муравьиного метода на фиксированном наборе данных. В результате экспериментов было получено, что муравьиный алгоритм работает быстрее алгоритма полного перебора и линейно зависит от размеров входной матрицы, также были выявлены оптимальные параметры для муравьиного алгоритма, обеспечивающие минимальное отклонение длины вычисленного маршрута от длины минимального.