Государственное образовательное учреждение высшего
профессионального образования
«Московский государственный технический
университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №7

на тему:

«Алгоритм муровья»

Студент ИУ7-54: Морозов И. А
Преподаватель: Погорелов. Д. А.

Москва 2018

Введение

В последние годы интенсивно разрабатывается научное направление с названием «Природные вычисления» (Natural Computing), объединяющее математические методы, в которых заложены принципы природных механизмов принятия решений. Эти механизмы обеспечивают эффективную адаптацию флоры и фауны к окружающей среде на протяжении нескольких миллионов лет.
Целью данной лабораторной работы является изучение и реализация алгоритма муровья.

1. Аналитическая часть

1.1. Описание алгоритмов

1.1.1. Задача Коммивояжера

Описание задачи:
Одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. Задачу можно свести к следующей формулировке : «Поиск минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе с n вершинами». Эта задача является NP-трудной, и точный переборный алгоритм её решения имеет факториальную сложность.

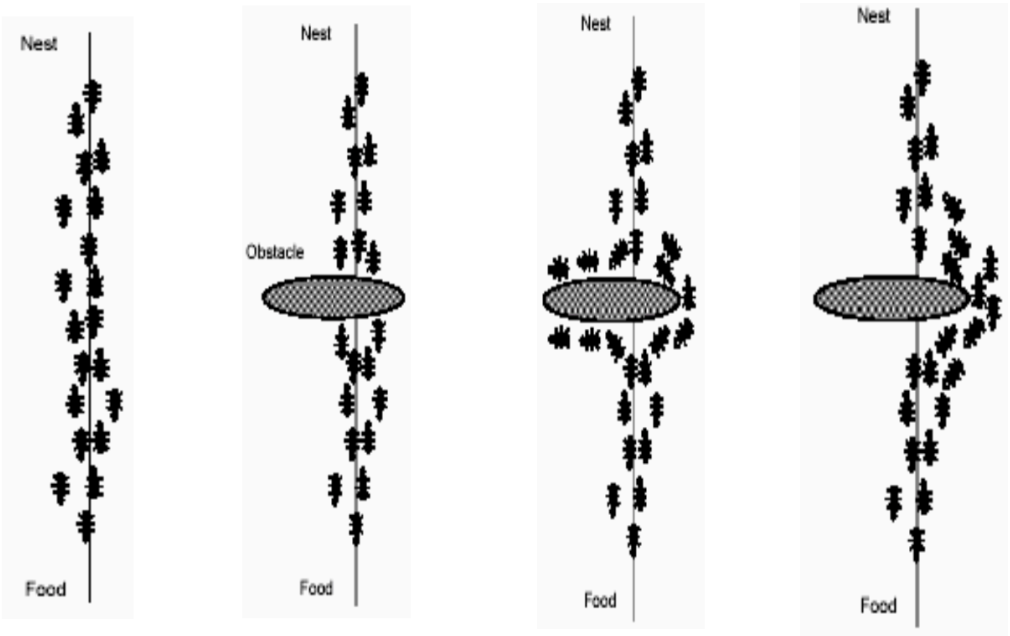
1.1.2. Алгоритм муровья

Описание алгоритма:

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о

качестве решения, полученной их предыдущих решений. Они могут использоваться как для статических, так и для динамических комбинаторных оптимизационных задач. Сходимость гарантирована, то есть в любом случае мы получим оптимальное решение, однако скорость сходимости неизвестна.

Идея муравьиного алгоритма - моделирование поведения муравьев, связанного с их способностью быстро находить кратчайший путь. При своем движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьев находить новый путь, если старый оказывается недоступным. Рассмотрим случай, показанный на рисунке, когда на оптимальном доселе пути возникает преграда. В этом случае необходимо определение нового оптимального пути. Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако, те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащен феромоном. Поскольку движение муравьев определяется концентрацией феромона, то следующие будут прдпочитать именно этот путь, продолжая обогощать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.



Очевидная положительная обратная связь приведет к ому, что кратчайший путь станет единственным маршрутом движения большинства муравьев. Моделирование испарения феромона - отрицательной обратной связи - гарантирует нам, что найденное локально оптимальное решение не будет единственным - муравьи будут искать и другие пути. Если мы моделируем процесс такого поведения на некотором графе, рёбра которого представляют собой возможные пути перемещения муравьёв, в течение определённого времени, то наиболее обогащённый феромоном путь по рёбрам этого графа и будет являться решением задачи, полученным с помощью муравьиного алгоритма.

Обобщенный алгоритм:

Любой муравьиный алгоритм, независимо от модификаций, представим в следующем виде

1. Пока (условия выхода не выполнены)
    1. Создаем муравьев
    2. Ищем решения
    3. Обновляем феромно
    4. Дополнительные действия (опционально)

Теперь рассмотрим каждый шаг в цикле более подробно

1. Создаем муравьёв

    Стартовая точка, куда помещается муравей, зависит ограничений, накладываемых условиями задачи. Потому что для каждой задачи способ размещения муравьёв является определяющим. Либо все они помещаются в одну точку, либо в разные с повторения, либо без повторений.

На этом же этапе задается начальный уровень феромона. Он инициализируется небольшим положительным числом для того, чтобы на начальном шаге вероятности перехода в следующую вершину не были нулевыми.

2. Ищем решения

Вероятность перехода из вершины i в вершину j определяется по следующей формуле

$$P_{i,j}(t) = \frac{\tau_{i,j}(t)^{\alpha} \left(\frac{1}{d_{i,j}}\right)^{\beta}}{\sum_{j \in allowednodes} \tau_{i,j}(t)^{\alpha} \left(\frac{1}{d_{i,j}}\right)^{\beta}}$$

где $\tau_{i,j}(t)$ - уровень феромона, $d_{i,j}$ - эврестическое расстояние, а α, β - константный параметры. При $\alpha = 0$ выбор ближайшего города наиболее вероятен, то есть алгоритм становится жадным. При $\beta = 0$ выбор происходит только на основании феромона, что приводит к субоптимальным решениям.

Поэтому необходим компромисс между этими величинами, который находится экспериментально.

3. Обновляем феромон

Уровень феромона обновляется в соответствии с приведённой формулой

$$\tau_{i,j}(t+1) = \left(1-\rho\right)\tau_{i,j}(t) + \sum_{k \in Colony\ that\ used\ edge(i,j)} \frac{Q}{L_k}$$

Где ρ - интенсивность испарения, $L_k(t)$ - цена текущего решения для k-ого муровья, а Q - параметр, имеющий значение порядка

цены оптимального решения, то есть $\frac{Q}{L_k(t)}$ - феромон,

откладываемый *k*-ым муравьём, использующим ребро (i, j).

4. Дополнительные действия

Обычно здесь используется алгоритм локального поиска, однако он может также появиться и после поиска всех решений.

Применение для задачи коммивояжёра:

Теперь с учетом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьев при выборе пути.

1. Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через $J_{i,k}$ список городов, которые необходимо посетить муравью *k*, находящемуся в городе *i*.
2. Муравьи обладают «зрением» - видимость есть эвристическое желание посетить город *j*, если муравей находится в городе *i*. Будем считать, что видимость обратно пропорциональна расстоянию между городами $\eta_{i,j} = \frac{1}{D_{i,j}}$.
3. Муравьи обладают «обонянием» - они могут улавливать след феромона, подтверждающий желание посетить город *j* из города *i* на основании опыта других муравьёв. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{i,j}(t)$
4. На этом основании мы можем сформулировать вероятностно-пропорциональное правило, определяющее вероятность перехода *k*-ого муравья из города *i* в город *j*:

$$\begin{cases} P_{ij,k}(t) = \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum_{l \in J_{i,k}} \left[\tau_{il}(t)\right]^{\alpha} \cdot \left[\eta_{il}\right]^{\beta}}, j \in J_{i,k} \\ P_{ij,k}(t) = 0, j \notin J_{i,k} \end{cases} \quad (1)$$

Где α,β - параметры, задающие веса следа феромона. При α=0 алгоритм вырождается до жадного алгоритма(будет выбран ближайший город). Заметим, что выбор города является вероятностным, правило (1) лишь определяет ширину зоны города j; в общую зону всех городов $J_{i,k}$ бросается случайное число, которое и определяет выбора муравья. Правило (1) не изменяется в ходе алгоритма, но у двух разных муравьев значение вероятности перехода будет отличаться, т.к они имеют разный список разрешённых городов.

5. Пройдя ребро (i, j), муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t, $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано в виде

$$\Delta\tau(t) = \begin{cases} \frac{Q}{L_k(t)}, (i, j) \in T_k(t) \\ 0, (i, j) \notin T_k(t) \end{cases}$$

Правила внешней среды определяют, в первую очередь, испарение феромона. Пусть $p \in [0,1]$ есть коэффициент испарения, тогда правило испарения имеет вид.

$$\tau_{ij}(t+1) = (1-p) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t); \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \quad (2)$$

Где m - количество муравьев в колонии.

В начале алгоритма количества феромона на рёбрах принимается равным небольшому положительному числу. Общее количество муравьёв остается постоянным и равным количеству городов, каждый муравей начинает маршрут из своего города.

2. Конструкторская часть

Для исследований будет реализован алгоритм муравья, далее будет произведена параметризация данного алгоритма.

2.1 Разработка алгоритма

2.1.1 Схема алгоритма муравья

Входные данные:

α - константный параметр

β - константный параметр

p - интенсивность испарения

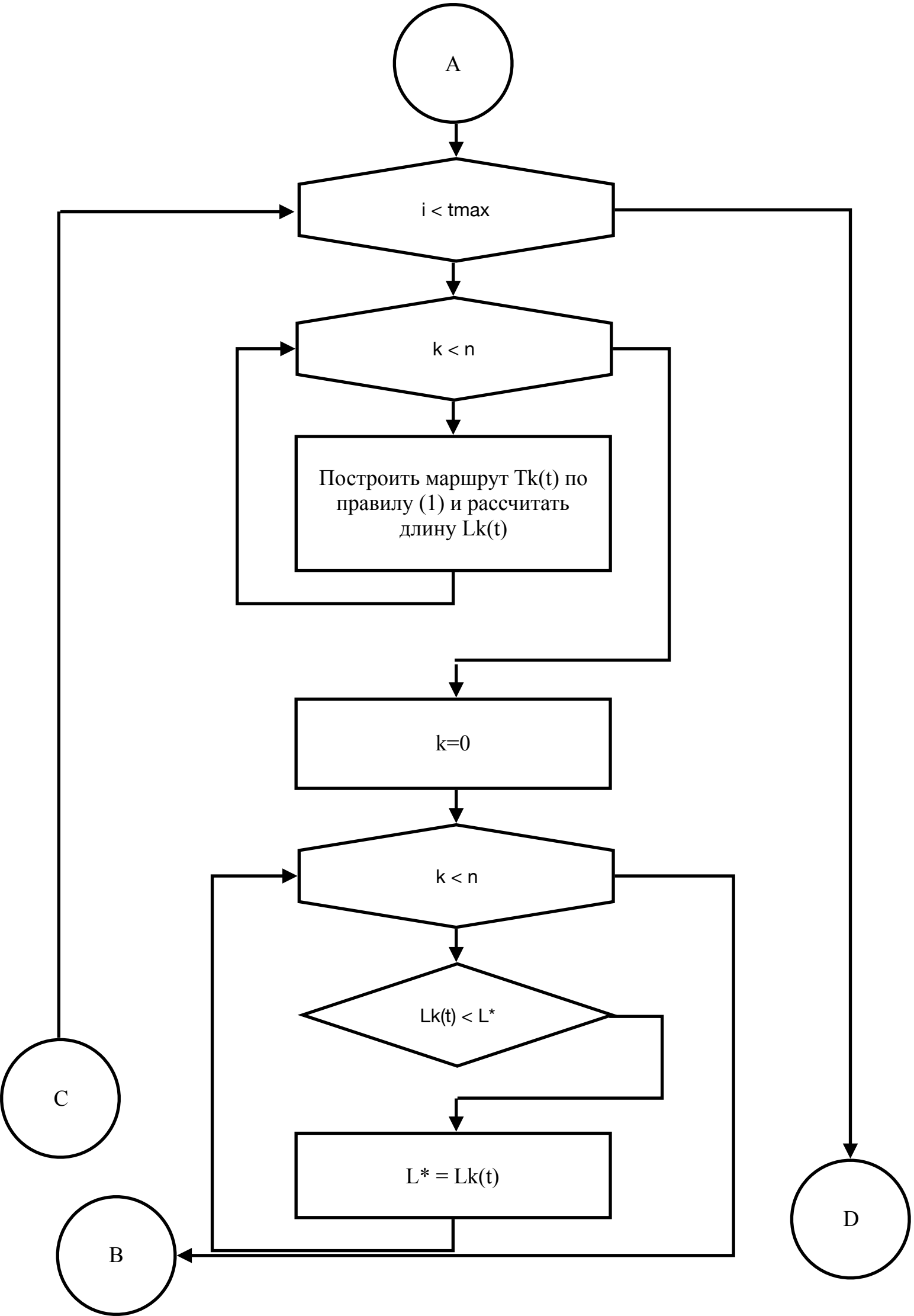
Q - параметр, имеющий значение порядка длины оптимального пути

t_{max} - параметр, задающий время жизни колонии

D - матрица расстояний

n - количество городов


Начало
Ввод матрицы расстояний D
Инициализация параметров алгоритма
Инициализация ребер - присвоение видимости и начальной концентрации феромона
Размещение муравьев в случайно выбранные города без совпадений
Выбор начального кратчайшего пути маршрута и определение L*
A


A
i < tmax
k < n
Построить маршрут Tk(t) по правилу (1) и рассчитать длину Lk(t)
k=0
k < n
Lk(t) < L*
L* = Lk(t)
C
B
D

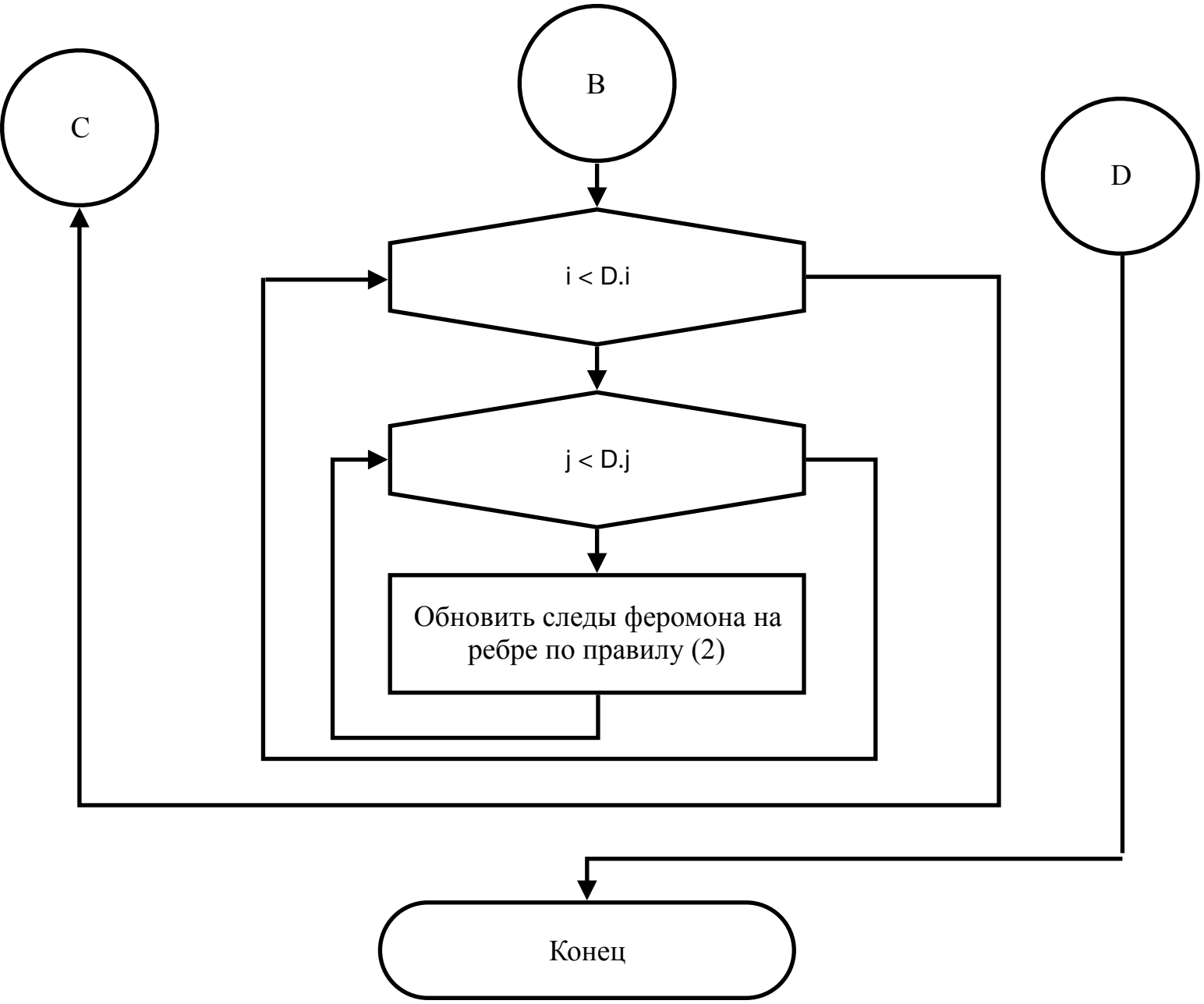


Рисунок 1. Схема алгоритма Муравья

3. Технологическая часть

В данном разделе будет представлено описание используемого языка программирования, а так же будет показан листинг кода функций, работающих согласно указанным выше алгоритмам.

3.1 Требования к программному обеспечению

Данна программа балы реализована на языке C++ компилятор для которого поддерживается многими операционными системами.

Компилятор: clang

3.2 Средства реализации

Программа была реализована на операционной система MacOS в среде разработки Clion

3.3 Листинги кода

Здесь я приведу листинги кода алгоритма муравья.

3.3.1 Листинг алгоритма муравья

Ant - структура представляющая собой одного муровья

```
void findBestTravel(std::vector<std::vector<double> > matrixGraph,
                    double alpha,
                    double beta,
                    double e,
                    double q,
                    int tmax) {

    // 3.Инициализация рёбер — присвоение видимости ηij и начальной концентрации феромона

    // Инициализировали область видимости
    std::vector<std::vector<double> > eta(matrixGraph.size());

    for (int i = 0; i < matrixGraph.size(); ++i) {
        eta[i].resize(matrixGraph.size());
        for (int j = 0; j < matrixGraph[i].size(); ++j) {
            (matrixGraph[i][j] != 0) ? (eta[i][j] = 1 / matrixGraph[i][j]) : eta[i][j] = 0;
        }
    }
```

```
    // Инициализировали феромоны единицами для "равновероятнотного"
начала
    std::vector<std::vector<double> > pheromones(matrixGraph.size());
    for (int i = 0; i < matrixGraph.size(); ++i) {
        pheromones[i].resize(matrixGraph.size());
        for (int j = 0; j < matrixGraph[i].size(); ++j) {
            pheromones[i][j] = 1;
        }
    }

    // 4.Размещение муравьёв в случайно выбранные города без
совпадений
    // Инициализируем массив муравьев
    std::vector<Ant> ants(matrixGraph.size());

    for (int i = 0; i < matrixGraph.size(); ++i) {
        Ant ant;
        ant.num_path = (int) matrixGraph.size();
        ant.start_town = (int) (i + 1);
        ant.curPath.push_back(ant.start_town);
        for (int j = 0; j < matrixGraph.size(); ++j) {
            if (j + 1 != i + 1) ant.path.push_back(j + 1);
        }
        ant.dpher.resize(matrixGraph.size());
        for (int j = 0; j < matrixGraph.size(); ++j) {
            ant.dpher[j].resize(matrixGraph.size());
            for (int k = 0; k < matrixGraph[i].size(); ++k) {
                ant.dpher[j][k] = 0;
            }
        }
        ants[i] = ant;
    }

    // 5.Выбор начального кратчайшего маршрута и определение L*
    double shortL = 1e5;
    std::vector<double> probabilities;

    // Основной цикл

    // 6.Цикл по времени жизни колонии t=1,tmax
    for (int t = 0; t < tmax; ++t) {
        // 7.Цикл по всем муравьям k=1,m
        for (int k = 0; k < ants.size(); ++k) {
            // 8.Построить маршрут Tk (t) по правилу (1) и рассчитать
длину Lk (t)
            while (!ants[k].path.empty()) {
                int from = *(ants[k].curPath.end() - 1);

                // Посчитаем вероятности для каждого города
                for (int j = 0; j < ants[k].path.size(); ++j) {

                    // Знаменатель вероятности
```

```
                    double sum = 0;
                    for (int i = 0; i < ants[k].path.size(); ++i) {
                        sum += (std::pow(pheromones[from - 1]
[ants[k].path[i] - 1], alpha)
                                * std::pow(eta[from - 1]
[ants[k].path[i] - 1], beta));
                    }

                    // вероятность
                    int to = ants[k].path[j];
                    double p = (std::pow(pheromones[from - 1][to -
1], alpha) * std::pow(eta[from - 1][to - 1], beta)) / sum;
                    probabilities.push_back(p);
                }
                // Выюберем город в который пойдем
                double coin = ((double) rand() / (double) RAND_MAX);
                CompSort(ants[k].path, probabilities,
probabilities.size());

                if (ants[k].path.size() != 1) {
                    for (int i = 0; i < probabilities.size(); ++i) {
                        if (probabilities[i] > coin) {

ants[k].curPath.push_back(ants[k].path[i]); // поместил
                            ants[k].path.erase(ants[k].path.begin() +
i); // удалил
                            ants[k].distance += matrixGraph[from - 1]
[*(ants[k].curPath.end() - 1) - 1];
                            ants[k].dpher[from - 1][(ants[k].path[i])
- 1] = q / ants[k].distance;
                            ants[k].dpher[(ants[k].path[i]) - 1][from
- 1] = q / ants[k].distance;
                            break;
                        }
                    }
                } else {
                    ants[k].curPath.push_back(ants[k].path[0]); //
поместил
                    ants[k].path.erase(ants[k].path.begin()); //
удалил
                    ants[k].distance += matrixGraph[from - 1]
[(ants[k].path[0]) - 1];
                    ants[k].dpher[from - 1][(ants[k].path[0]) - 1] =
q / ants[k].distance;
                    ants[k].dpher[(ants[k].path[0]) - 1][from - 1] =
q / ants[k].distance;
                }

                //Очистка данных о вероятностях
                probabilities.clear();
            }
```

```
            // Замыкаем путь
            int from = *(ants[k].curPath.end() - 1);
            int to = *(ants[k].curPath.begin());
            ants[k].distance += matrixGraph[from - 1][to - 1];
            ants[k].dpher[from - 1][to - 1] = q / ants[k].distance;
            ants[k].dpher[to - 1][from - 1] = q / ants[k].distance;
        }

        //debug_output_ants(ants);
        // Далее обновляем наш минимум
        for (int j = 0; j < ants.size(); ++j) {
            // Если минимальный путь нашелся - выводим муравья
нашедшего этот путь, дистанцию, которую он наше,
            // а так же сам путь
            if (shortL > ants[j].distance) {
                std::cout << "New best way find by ant: " << j + 1 <<
" its distance: " << ants[j].distance << " ";
                std::cout << "way: ";
                for (int k = 0; k < ants[j].curPath.size(); ++k) {
                    std::cout << ants[j].curPath[k] << " ";
                }
                std::cout << std::endl;
                shortL = ants[j].distance;
            }
        }

        //Обновляем феромоны
        for (int j = 0; j < pheromones.size(); ++j) {
            for (int k = 0; k < pheromones[j].size(); ++k) {
                double sum = 0;
                for (int z = 0; z < ants.size(); ++z) {
                    sum += ants[z].dpher[j][k];
                }

                pheromones[j][k] = (1 - e) * pheromones[j][k] + sum;
            }
        }
        ants.clear();
        ants.resize(matrixGraph.size());

        for (int i = 0; i < matrixGraph.size(); ++i) {
            Ant ant;
            ant.num_path = (int) matrixGraph.size();
            ant.start_town = (int) (i + 1);
            ant.curPath.push_back(ant.start_town);
            for (int j = 0; j < matrixGraph.size(); ++j) {
                if (j + 1 != i + 1) ant.path.push_back(j + 1);
            }
            ant.dpher.resize(matrixGraph.size());
            for (int j = 0; j < matrixGraph.size(); ++j) {
                ant.dpher[j].resize(matrixGraph.size());
                for (int k = 0; k < matrixGraph[i].size(); ++k) {
```

```
                    ant.dpher[j][k] = 0;
                }
            }
            ants[i] = ant;
        }
    }
}
```

4. Экспериментальная часть

В данном разделе будет представлено сравнение временных характеристик алгоритмов

4.1 Постановка эксперимента

В данном разделе будет произведена параметризация алгоритма муравья. И найдены оптимальные значения параметров α, β, p, Q. При этом $t_{max} = 100$. Эксперимент будет произведен на следующей матрице расстояний:

0	5	7	10	5
5	0	6	2	8
7	6	0	8	8
10	2	8	0	6
5	8	8	6	0

В первую очередь подберем оптимальные параметры α, β. Будем перебирать их от 0 до 1 с шагом 0.1. Для получения более точных результатов, было произведено 100 итераций $p = 0.5$, $Q = 1$.

α	β	t
0	0	0.00536
1.5	0	0.04985
1.6	0	0.004817
1.2	0	0.004742
1.5	0	0.004734
1.6	0	0.004731
1.7	0	0.004679
1.9	0	0.004669
1.5	0	0.004668
1.6	0	0.004597
1.9	0	0.04585
1.7	0	0.004563
1.9	0	0.004548

Исходя из данных результатов, можно увидеть, что наилучшее время получается при $\alpha = 1.9$ и $\beta = 0$

Далее зафиксируем α и β и подберем параметр p, будем перебирать его от 0 до 1 с шагом 0,1

p	t
0	0.0065
0.3	0.006413
0.5	0.005407
0.1	0.005029
0.5	0.004725
1	0.004489
1	0.004427
1	0.004419

Таким образом наилучшее время на данном наборе значений получается при $p = 1$

Далее подберем Q, будет перебирать его от 1 до 5 с шагом 1. Наилучший результат получился при $t = 2$ время при этом $t = 0.004363$

Заключение

В ходе лабораторной работы был изучен и реализован алгоритм муравья. Была проведена параметризация на конкретной матрице расстояний, что говорит, о том, что нельзя принимать данные параметры абсолютно оптимальными.