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»

ОТЧЁТ ПО АНАЛИЗУ АЛГОРИТМОВ

к лабораторной работе №6 на тему:

«Муравьиный алгоритм»

Студент: Марабян К.В. ИУ7-55

Москва 2018

**Содержание**

[**Введение**](#_y1ea5kse8d8s) **2**

[**1 Аналитическая часть**](#_n91k6n79dzut) **3**

[**1.1 Постановка задачи**](#_3gqvaou3e8f5) **3**

[**1.2 Описание алгоритмов**](#_brhollpqxi2m)4

[**2 Конструкторский раздел**](#_5g4cmomvb0gf)7

[**2.1 Разработка алгоритмов**](#_codkpmvflfqb)7

[**3 Технологический раздел**](#_dgwh654ihhkp)8

[**3.1 Минимальные требования к программному обеспечению**](#_mncwt7me2c7x)8

[**3.2 Средства реализации**](#_80qaa7iwbkbc)8

[**3.3 Листинг кода**](#_l3quu9c8lbwe)9

[**4 Экспериментальная часть**](#_hlnbv34b5bsg) **18**

[**4.1**](#_445uz6nu0e23) **Постановка эксперимента 1**9

**Вывод 19**

[**Заключение**](#_1hl6wz3mwv0x)21

##### 

##### 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.

Среди так называемых “Soft computing techniques”, разработанных за последние десять лет для трудно решаемых задач дискретной оптимизации, числятся:

• Генетические алгоритмы (GAs)  
o Основываются на естественном отборе и генетике.

• Муравьиные алгоритмы (Ant Colony Optimization – ACO, Ant Systems – AS) o Моделируют поведение муравейника.

Имитация самоорганизации муравьиной̆ колонии составляет основу муравьиных алгоритмов оптимизации. Колония муравьев может рассматриваться как многоагентная система, в которой каждый агент (муравей) функционирует автономно по очень простым правилам. В противовес почти примитивному поведению агентов, поведение всей системы получается на удивление разумным.

Муравьиные алгоритмы серьезно исследуются европейскими учеными с середины 90-х годов. На сегодняшний день уже получены хорошие результаты для оптимизации таких сложных комбинаторных задач, как задача коммивояжера, задача оптимизации маршрутов грузовиков, задача раскраски графа, квадратичная задача о назначениях, задача оптимизации сетевых графиков, задача календарного планирования и многие другие. Особенно эффективны муравьиные алгоритмы при динамической оптимизации процессов в распределенных нестационарных системах, например, трафиков в телекоммуникационных сетях.

##### 1 Аналитическая часть

В данном разделе представлены теоретическое описание алгоритма и указание области их применения.

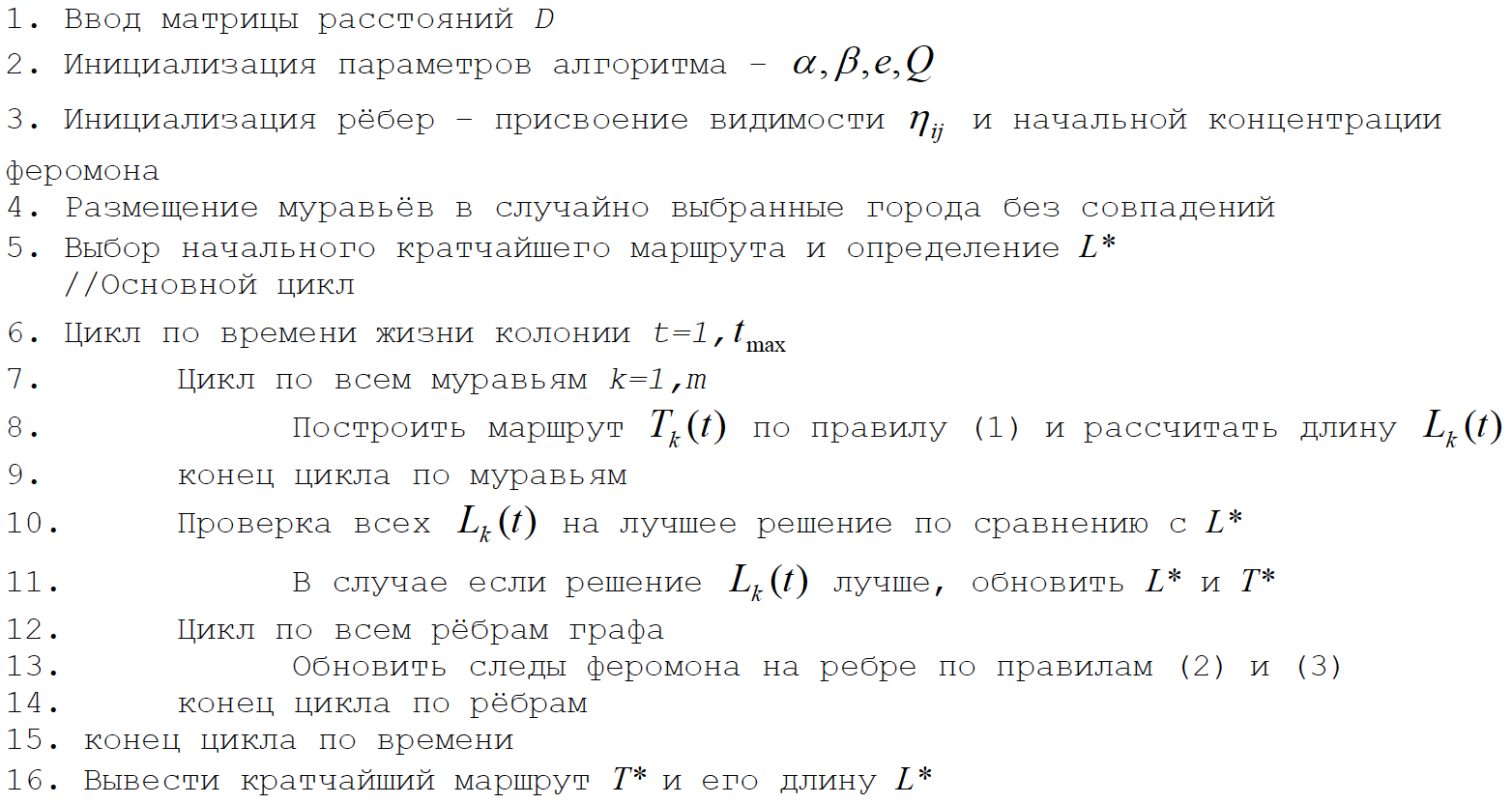
##### 1.1 Постановка задачи

1) Реализовать и исследовать муравьиный алгоритм приближенного решения задачи коммивояжера.

2) Задача коммивояжера – задача обхода всех вершин графа со взвешенными ребрами без повторного захода в одни и те же вершины.

##### 3) В данной задаче в начальный момент времени муравьи распределены по одному в каждой вершине.

##### 1.2 Муравьиный алгоритм для задачи коммивояжёра в псевдокоде



**1.3 Применение муравьиных алгоритмов для задачи коммивояжёра**

Задача 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 коммивояжёр, а веса рёбер отражают расстояния (длины) или стоимости проезда. Эта задача является NP-трудной, и точный переборный алгоритм её решения имеет факториальную сложность.

Моделирование поведения муравьёв связано с распределением феромона на тропе – ребре графа в задаче коммивояжёра. При этом вероятность включения ребра в маршрут отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его рёбрах, следовательно, большее количество муравьёв будет включать его в синтез собственных маршрутов. Моделирование такого подхода, использующего только положительную обратную связь, приводит к преждевременной сходимости – большинство муравьёв двигается по локально оптимальному маршруту. Избежать этого можно, моделируя отрицательную обратную связь в виде испарения феромона. При этом если феромон испаряется быстро, то это приводит к потере памяти колонии и забыванию хороших решений, с другой стороны, большое время испарения может привести к получению устойчивого локального оптимального решения.

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

1. Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещённых городов – список запретов. Обозначим через

*Ji*,*k* список городов, которые необходимо посетить муравью *k*, находящемуся в городе *i*.

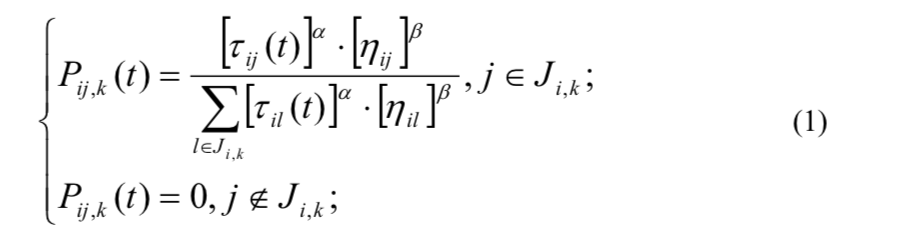
1. Муравьи обладают «зрением» – видимость есть эвристическое желание посетить город *j*, если муравей

находится в городе *i*. Будем считать, что видимость обратно пропорциональна расстоянию между городамиη*ij* =1/*Dij*.

1. 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

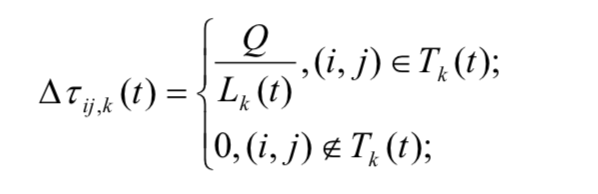
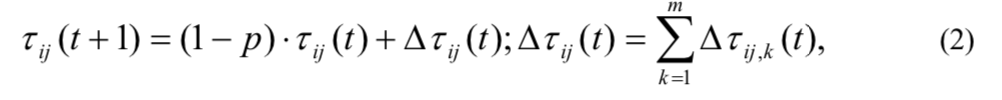
момент времени *t* обозначим через τ *ij* (*t*) .

1. На этом основании мы можем сформулировать вероятностно-пропорциональное правило, определяющее вероятность перехода *k*-ого муравья из города *i* в город *j*:



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 *Ji*,*k* бросается случайное число, которое и определяет выбор муравья. Правило (1) не изменяется в ходе алгоритма, но у двух разных муравьёв значение вероятности перехода будут 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 *Tk* (*t*) есть маршрут, пройденный муравьём *k* к моменту времени *t*, *Lk* (*t*) – длина этого маршрута, а *Q* – параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано в виде

  
Правила внешней среды определяют, в первую очередь, испарение феромона. Пусть *p*∈[0,1]есть коэффициент испарения, тогда правило испарения имеет вид:  
где *m* – количество муравьёв в колонии.

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

Дополнительная модификация алгоритма может состоять в ведении так называемых «элитных» муравьёв, которые усиливают рёбра наилучшего маршрута, найденного с начала работы алгоритма. Обозначим через *T\** наилучший текущий маршрут, через *L\** – его длину. Тогда если в колонии есть *e* элитных муравьёв, то рёбра маршрута получат дополнительное количество феромона



##### 2 Конструкторский раздел

В этом разделе приведены схемы алгоритмов.

##### 2.1 Разработка алгоритмов

##### Ниже приложены блок-схемы алгоритмов решения поставленных задач.

Рисунок 2.1 — Муравьиный алгоритм.

##### 3 Технологический раздел

Данный раздел содержит указание использованного языка программирования, а также листинг кода.

##### 3.1 Минимальные требования к программному обеспечению

Операционные системы: MacOS, Win7,10

Объем жесткого диска: 10 Мб.

Оперативная память: 1024 MB ОЗУ.

##### 3.2 Средства реализации

Для реализации программ мною был выбрал язык программирования - C++.

##### 3.3 Листинг кода

##### #include <iostream>

##### #include <vector>

##### #include <random>

##### #include <ctime>

##### #include <algorithm>

##### #include <limits>

##### #include <iomanip>

##### #include "ant.hpp"

##### #include "town.hpp"

##### #include "main.hpp"

##### float rand\_FloatRange(float a, float b)

##### {

##### return ((b - a) \* ((float)rand() / RAND\_MAX)) + a;

##### }

##### void initTowns(Town\* towns, int\*\* townsMatrix)

##### {

##### for (int i = 0; i < TOWN\_COUNT; i++)

##### {

##### towns[i] = Town(i); // Create town object

##### for (int j = 0; j < TOWN\_COUNT; j++)

##### {

##### if (i == j)

##### {

##### townsMatrix[i][j] = 0;

##### continue;

##### }

##### int distance = 1 + std::rand() % MAX\_DISTANCE;

##### townsMatrix[i][j] = distance; // Update matrix of distances between towns

##### townsMatrix[j][i] = distance;

##### }

##### }

##### }

##### float randomFloat(float min, float max) {

##### return (max - min) \* ((((float) rand()) / (float) RAND\_MAX)) + min ;

##### }

##### void initPheromons(float\*\* pheromonMatrix)

##### {

##### for (int i = 0; i < TOWN\_COUNT; i++)

##### {

##### for (int j = 0; j < TOWN\_COUNT; j++)

##### {

##### if (i == j)

##### {

##### pheromonMatrix[i][j] = 0;

##### continue;

##### }

##### pheromonMatrix[i][j] = 0.1;

##### }

##### }

##### }

##### template <typename T>

##### void print\_matrix(T \*\*matr, int size){

##### for(int i = 0; i < size; i++){

##### for (int j = 0; j < size; j++){

##### std::cout << matr[i][j] << " ";

##### }

##### std::cout << std::endl;

##### }

##### }

##### void initAnts(Ant\* ants, Town\* towns)

##### {

##### std::vector<int>leftTownsIDs(TOWN\_COUNT); // Create vector of towns' IDs

##### std::iota(std::begin(leftTownsIDs), std::end(leftTownsIDs), 0); //

##### for (int ant = 0; ant < ANT\_COUNT; ant++)

##### {

##### int townID = leftTownsIDs[std::rand() % leftTownsIDs.size()]; // Pick random town from towns, that are left

##### towns[townID].addAnt(ants[ant]); // Place ant in this town

##### ants[ant] = Ant(townID); // Add this town in ant's route

##### leftTownsIDs.erase(std::remove(leftTownsIDs.begin(), leftTownsIDs.end(), townID), leftTownsIDs.end()); // Delete occupied town

##### }

##### }

##### void showAnts(Ant\* ants)

##### {

##### for (int i = 0; i < ANT\_COUNT; i++)

##### {

##### ants[i].showAnt();

##### }

##### }

##### void showTowns(int\*\* townsMatrix)

##### {

##### for (int i = 0; i < TOWN\_COUNT; i++)

##### {

##### for (int j = 0; j < TOWN\_COUNT; j++)

##### {

##### std::cout << townsMatrix[i][j] << (j < TOWN\_COUNT - 1 ? " " : "\n");

##### }

##### }

##### }

##### void initParameters(float& a, float& b, int& e)

##### {

##### a = 0.7;

##### b = 0.3;

##### e = 1;

##### }

##### void planRoute(Ant& ant, int\*\* townsMatrix, float\*\* pheromonMatrix, float alpha, float beta)

##### {

##### float top = 0;

##### float bottom = 0;

##### int startID;

##### int bestID = -1;

##### float maxP = -1;

##### float rand;

##### float P\_min = 0;

##### float P\_max = 0;

##### float P;

##### std::vector<int> destinations = ant.getDestinations();

##### for (int j = 0; j < TOWN\_COUNT - 1; j++)

##### {

##### maxP = -1;

##### bestID = -1;

##### destinations = ant.getDestinations();

##### std::vector<int> route = ant.getRoute();

##### startID = route[route.size() - 1];

##### rand = rand\_FloatRange(0.3, 1);//std::rand() % 1;

##### //std::cout << "Rand = " << rand << std::endl;

##### P\_min = 0;

##### for (int i = 0; i < destinations.size(); i++)

##### {

##### int endID = destinations[i];

##### //std::cout << "start/end = " << startID << " " << endID << std::endl;

##### //std::cout << "coef" << alpha << " " << beta << std::endl;

##### //std::cout << "1 = " << float(pow(pheromonMatrix[startID][endID], alpha)) << std::endl;

##### //std::cout << "2 = " << float(pow(float(1 / float(townsMatrix[startID][endID])), beta)) << std::endl;

##### top = float(pow(pheromonMatrix[startID][endID], alpha)) \* float(pow(float(1 / float(townsMatrix[startID][endID])), beta));

##### //std::cout << "top = " << top << std::endl;

##### 

##### bottom = 0;

##### for (int k = 0; k < destinations.size(); k++)

##### {

##### int tempID = destinations[k];

##### float temp = (float)(1.0 / townsMatrix[startID][tempID]);

##### bottom += pow(pheromonMatrix[startID][tempID], alpha) \* pow(temp, beta);

##### }

##### //std::cout << "bottom = " << bottom << std::endl;

##### 

##### 

##### P = float(top / bottom);

##### P\_min += P;

##### P\_max = P + P\_min;

##### //std::cout << "P = " << P << std::endl;

##### //std::cout << "P\_min = " << P\_min << std::endl;

##### //if ((P\_min <= rand) && (rand <= P\_max)){

##### if (P\_min >= rand){

##### //std::cout << "!!!" << std::endl;

##### //std::cout << "PPP = " << P << std::endl;

##### //if (maxP < P)

##### //{

##### maxP = P;

##### bestID = destinations[i];

##### //}

##### }

##### 

##### }

##### //std::cout << "BestID = " << bestID << std::endl;

##### ant.updateRoad(bestID, townsMatrix[startID][bestID]);

##### //ant.showAnt();

##### }

##### //ant.showAnt();

##### }

##### void updatePheromon(float\*\* pheromonMatrix, Ant\* ants)

##### {

##### int found;

##### for (int i = 0; i < TOWN\_COUNT; i++)

##### {

##### for (int j = 0; j < TOWN\_COUNT; j++)

##### {

##### if (i == j)

##### {

##### continue;

##### }

##### float pheromonSum = 0;

##### for (int ant = 0; ant < ANT\_COUNT; ant++)

##### {

##### found = 0;

##### for (int k = 0; k < TOWN\_COUNT - 1; k++)

##### {

##### std::vector<int> antRoute = ants[ant].getRoute();

##### if (antRoute[k] == i && antRoute[k+1] == j)

##### {

##### float currentPheromon = ants[ant].getPheromon();

##### //std::cout << "Ant " << ant << " " << "Pheromon " << currentPheromon << std::endl;

##### pheromonSum += currentPheromon;

##### }

##### }

##### }

##### pheromonMatrix[i][j] = (1 - P\_COEF) \* pheromonMatrix[i][j] + pheromonSum;

##### }

##### }

##### }

##### int elit\_ants (float \*\* pheromonMatrix, Ant\* ants)

##### {

##### int minLenght = ants[0].getRouteLenght();

##### for (int i = 1; i < ANT\_COUNT; i++) {

##### if (minLenght > ants[i].getRouteLenght()) {

##### minLenght = ants[i].getRouteLenght();

##### }

##### }

##### //std::cout << minLenght << std::endl;

##### //std::cout<< "Pheromon: " << ELITE\_ANT \* (float(Q)/float(minLenght)) << std::endl;

##### for (int i = 1; i < ANT\_COUNT; i++) {

##### if (ants[i].getRouteLenght() == minLenght) {

##### for (int j = 0; j < ants[i].getRoute().size() - 1; j++) {

##### int m = ants[i].getRoute()[j];

##### int n = ants[i].getRoute()[j + 1];

##### 

##### pheromonMatrix[m][n] += (ELITE\_ANT \* (float(Q)/float(minLenght)));

##### pheromonMatrix[n][m] += (ELITE\_ANT \* (float(Q)/float(minLenght)));

##### }

##### }

##### }

##### return minLenght;

##### }

##### int antAlgorithm(Town\* towns, int\*\* townsMatrix, float\*\* pheromonMatrix, Ant\* ants, float alpha)

##### {

##### initTowns(towns, townsMatrix);

##### initPheromons(pheromonMatrix);

##### int currentLenght = 0;

##### int Lmin = 1e5;

##### for (int gen = 0; gen < GENERATIONS; gen++)

##### {

##### //std::cout << "GENERATION " << gen << "\n" << std::endl;

##### initAnts(ants, towns);

##### for (int ant = 0; ant < ANT\_COUNT; ant++)

##### {

##### //std::cout << "ANT " << ant << std::endl;

##### planRoute(ants[ant], townsMatrix, pheromonMatrix, alpha, 1 - alpha);

##### //ants[ant].showAnt();

##### //std::cout << std::endl;

##### }

##### updatePheromon(pheromonMatrix, ants);

##### //print\_matrix<float> (pheromonMatrix, TOWN\_COUNT);

##### currentLenght = elit\_ants(pheromonMatrix, ants);

##### //std::cout << "CURRENT LENGTH " << currentLenght << std::endl;

##### if (currentLenght < Lmin)

##### {

##### Lmin = currentLenght;

##### }

##### //std::cout << "Lmin: " << Lmin << std::endl;

##### //print\_matrix<float> (pheromonMatrix, TOWN\_COUNT);

##### }

##### return Lmin;

##### }

##### void printParameters(float a, float p, int e, int gen, int length)

##### {

##### std::cout.precision(4);

##### std::cout << std::setw(5) << float(a) << " | " << std::setw(5) << p << " | " << std::setw(5) << e << std::setw(5) << " | " << gen << " | " << std::setw(8) << length << std::endl;

##### }

##### void parametrization(Town\* towns, int\*\* townsMatrix, float\*\* pheromonMatrix, Ant\* ants)

##### {

##### float a = 0.0;

##### float p = 0.0;

##### int length;

##### std::cout << "Alpha | " << "Phero | " << "Elite | " << "Gen | " << "Length" << std::endl;

##### int minLen = 1e5;

##### float topA;

##### float topP;

##### float topE;

##### float topGen;

##### while (a <= 1)

##### {

##### while (p <= 2)

##### {

##### for (int e = 0; e < 5; e++)

##### {

##### for (int gen = 1; gen < 10; gen++)

##### {

##### length = antAlgorithm(towns, townsMatrix, pheromonMatrix, ants, float(a));

##### printParameters(float(a), p, e, gen, length);

##### if (minLen > length){

##### minLen = length;

##### topA = a;

##### topP = p;

##### topE = e;

##### topGen = gen;

##### }

##### }

##### }

##### p += float(0.5);

##### }

##### p = 0;

##### a = float(a + float(0.2));

##### }

##### std::cout << "\n\n\n";

##### std::cout << minLen << " | " << topA << " | " << topP << " | " << topE << " | " << topGen << std::endl;

##### }

##### int main(void)

##### {

##### srand(time(NULL));

##### Town\* towns = new Town[TOWN\_COUNT];

##### int\*\* townsMatrix = new int\*[TOWN\_COUNT];

##### for (int i = 0; i < TOWN\_COUNT; i++)

##### {

##### townsMatrix[i] = new int[TOWN\_COUNT];

##### }

##### //initTowns(towns, townsMatrix);

##### Ant\* ants = new Ant[ANT\_COUNT];

##### float alpha, beta;

##### int e;

##### initParameters(alpha, beta, e);

##### float\*\* pheromonMatrix = new float\*[TOWN\_COUNT];

##### for (int i = 0; i < TOWN\_COUNT; i++)

##### {

##### pheromonMatrix[i] = new float[TOWN\_COUNT];

##### }

##### int length = antAlgorithm(towns, townsMatrix, pheromonMatrix, ants, alpha);

##### std::cout << length << std::endl;

##### parametrization(towns, townsMatrix, pheromonMatrix, ants);

##### //initPheromons(pheromonMatrix);

##### /\* print\_matrix<float> (pheromonMatrix, TOWN\_COUNT);

##### int Lmax = std::numeric\_limits<int>::infinity();

##### for (int gen = 0; gen < GENERATIONS; gen++)

##### {

##### std::cout << "GENERATION " << gen << "\n" << std::endl;

##### float P = 0.0;

##### initAnts(ants, towns);

##### 

##### for (int ant = 0; ant < ANT\_COUNT; ant++)

##### {

##### std::cout << "ANT " << ant << std::endl;

##### planRoute(ants[ant], townsMatrix, pheromonMatrix, alpha, beta);

##### ants[ant].showAnt();

##### std::cout << std::endl;

##### }

##### updatePheromon(pheromonMatrix, ants);

##### print\_matrix<float> (pheromonMatrix, TOWN\_COUNT);

##### elit\_ants(pheromonMatrix, ants);

##### std::cout << std::endl;

##### print\_matrix<float> (pheromonMatrix, TOWN\_COUNT);

##### }

##### \*/

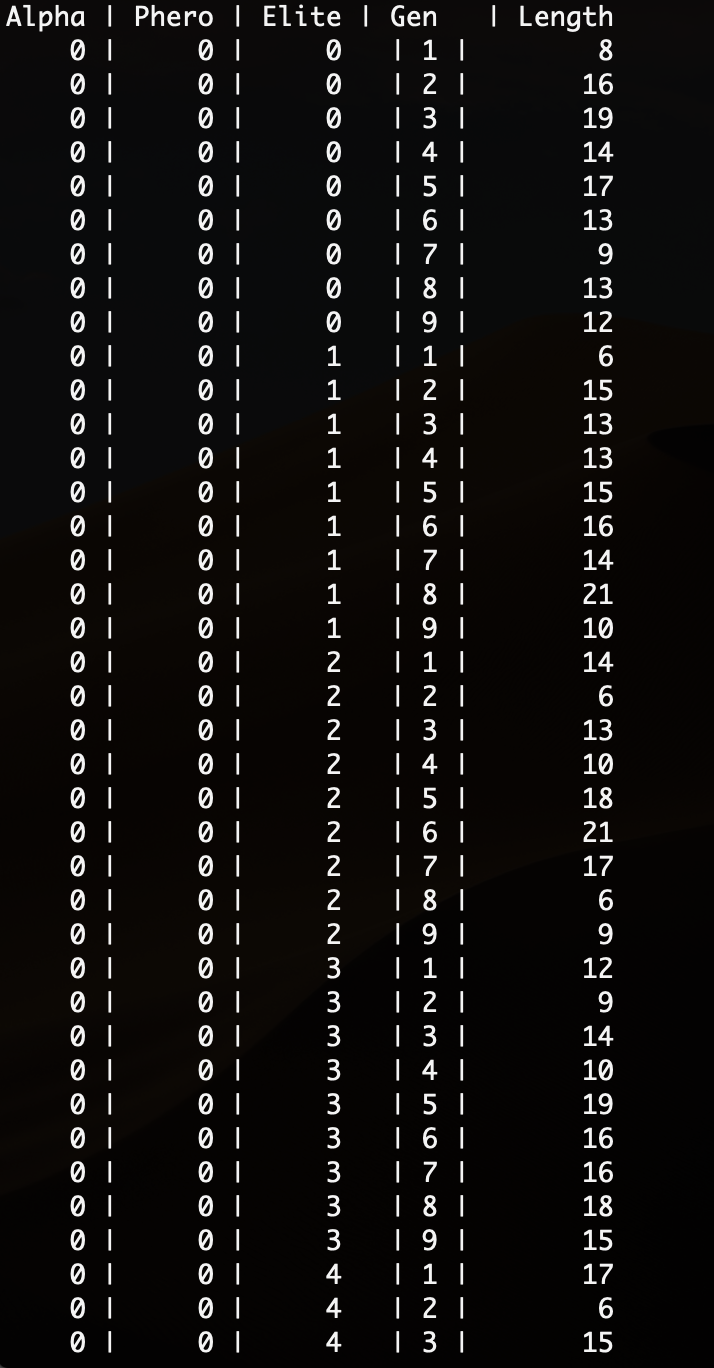
##### return 0;

##### }

##### 4 Экспериментальная часть

В данном разделе будут представлены результаты работы муравьиного алгоритма.

##### 4.1 Постановка эксперимента



**Вывод**

В ходе работы был реализован и исследован муравьиный алгоритм решения задачи коммивояжера.

Было проведено исследование зависимости оптимальности результата от параметров уравнения. Были получен следующий вывод:

Для каждого коэффициента целесообразно задать ряд значений и сравнить результаты работы программы при каждом наборе. Заранее определить оптимальные величины коэффициентов не представляется возможным, однако, точно можно сказать, что alpha и beta должны быть одного порядка и прочие коэффициенты не должны обращаться в ноль или принимать очевидно большие или малые значения.

##### Заключение

В ходе работы был изучен и реализован муравьиный алгоритм.