# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc153139771)

[1 ОНТОЛОГИЯ 5](#_Toc153139772)

[1.1 Постановка задачи 5](#_Toc153139773)

[1.2 Теоретический раздел 5](#_Toc153139774)

[1.3 Создание онтологии 5](#_Toc153139775)

[1.4 Программная реализация 10](#_Toc153139776)

[2 МЕТОД ИМИТАЦИИ ОТЖИГА 11](#_Toc153139777)

[2.1 Постановка задачи 11](#_Toc153139778)

[2.2 Теоретический раздел 11](#_Toc153139779)

[2.3 Ручной просчет алгоритма 12](#_Toc153139780)

[2.4 Программная реализация 15](#_Toc153139781)

[3 МЕТОД ОПТИМИЗАЦИИ РОЯ ЧАСТИЦ 19](#_Toc153139782)

[3.1 Постановка задачи 19](#_Toc153139783)

[3.2 Теоретический раздел 19](#_Toc153139784)

[3.3 Ручной просчет алгоритма 21](#_Toc153139785)

[3.4 Программная реализация 23](#_Toc153139786)

[4 МЕТОД ОПТИМИЗАЦИИ МУРАВЬИНОЙ КОЛОНИИ 24](#_Toc153139787)

[4.1 Постановка задачи 24](#_Toc153139788)

[4.2 Теоретический раздел 24](#_Toc153139789)

[4.3 Ручной просчет алгоритма 26](#_Toc153139790)

[4.4 Программная реализация 39](#_Toc153139791)

[5 АЛГОРИТМ ПЧЕЛИНОЙ КОЛОНИИ 40](#_Toc153139792)

[5.1 Постановка задачи 40](#_Toc153139793)

[5.2 Теоретический раздел 40](#_Toc153139794)

[5.3 Ручной просчет алгоритма 42](#_Toc153139795)

[5.4 Программная реализация 45](#_Toc153139796)

[ЗАКЛЮЧЕНИЕ 46](#_Toc153139797)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47](#_Toc153139798)

[ПРИЛОЖЕНИЯ 49](#_Toc153139799)

# ВВЕДЕНИЕ

В современном мире данные являются одним из самых ценных ресурсов. Они хранятся в различных системах, таких как базы данных, электронные таблицы, текстовые файлы и другие. Однако, для того чтобы эти данные были полезными, необходимо уметь анализировать и структурировать их.

Структурный анализ данных в системах поддержки принятия решений – это процесс, который позволяет выявить структуру данных, определить связи между ними и описать их в виде модели. Это позволяет лучше понимать данные и использовать их для принятия эффективных решений.

В данной курсовой работе будут рассмотрены различные подходы к анализу данных, такие как статистический анализ, машинное обучение, анализ текстов и другие. Будут также рассмотрены методы и инструменты, которые используются для обработки и структурирования информации.

# 1 ОНТОЛОГИЯ

## 1.1 Постановка задачи

Предметная область: IT-компания.

В данной практической работе необходимо создать формальную спецификацию, описывающую понятия, отношения и правила в предметной области. Основная цель – обеспечение эффективной обработки и использования информации в приложениях.

## 1.2 Теоретический раздел

Онтология — раздел философии, изучающий фундаментальные принципы устройства бытия, его начала, сущностные формы, свойства и категориальные распределения [4]. Формальная модель онтологии (O) может быть определена как упорядоченная тройка вида (1.1):

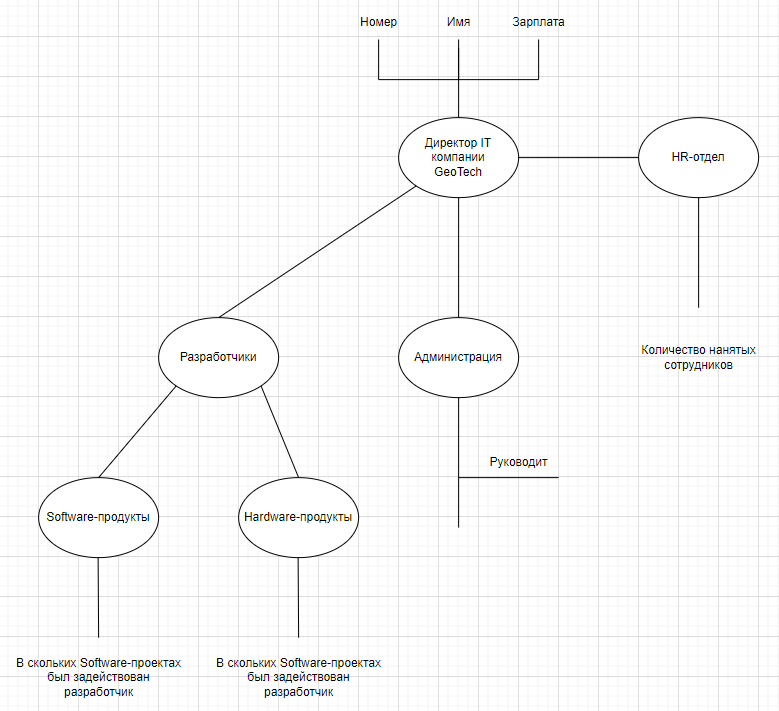
, где (1.1)

X – конечное множество концептов (понятий, терминов) предметной области, которую представляет онтология; R – конечное множество отношений между концептами (понятиями, терминами) заданной предметной области; F – конечное множество функций интерпретации (аксиоматизации), заданных на концептах и/или отношениях онтологии [3].

## 1.3 Создание онтологии

Первостепенная задача при создании онтологии, это изображение, описывающее схематически онтологию. В изображении должны быть отображены все объекты, их поля (слоты), и взаимосвязи между ними.

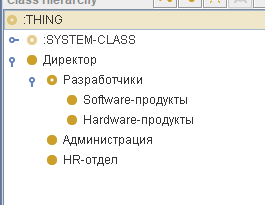
Изображение, описывающее онтологию, представлено на рисунке 1.1.



**Рисунок 1.1 – Изображение, описывающее онтологию**

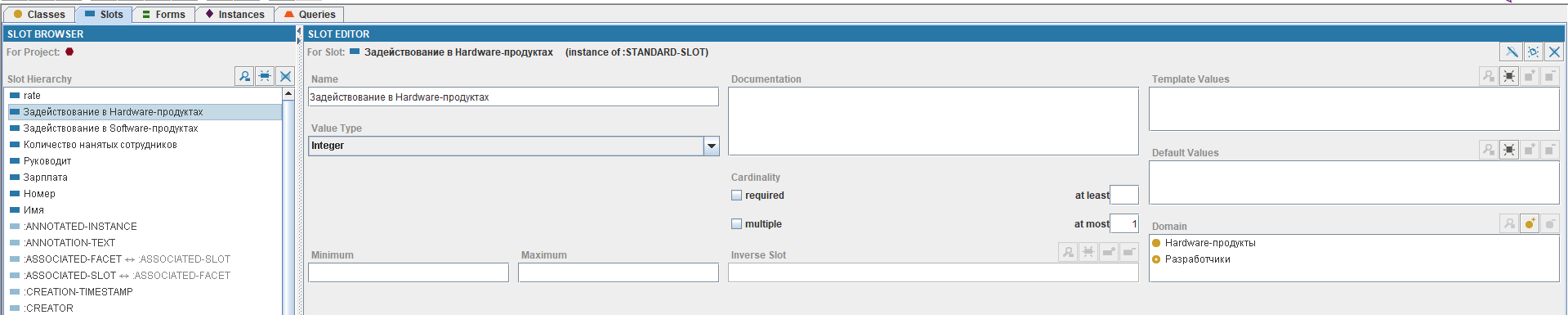
Для разработки онтологии использовалось программное обеспечение «Protege». В нем были разработаны классы, слоты, созданы сущности под каждый класс.

Созданные классы представлены на рисунке 1.2.



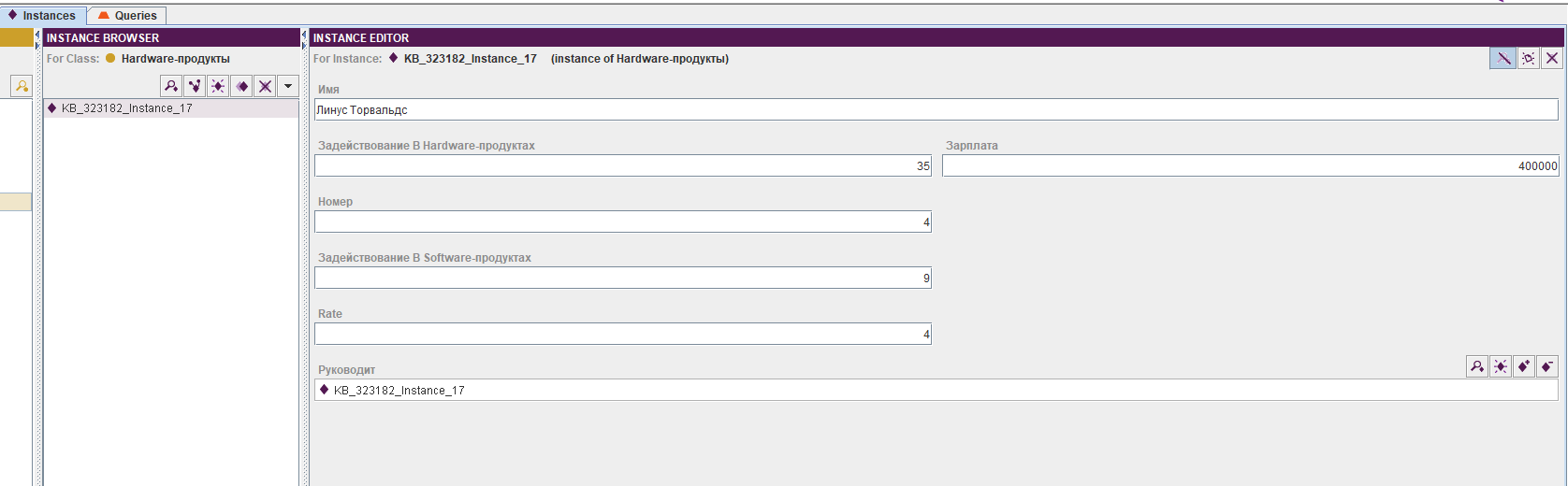
**Рисунок 1.2 – Созданные классы**

Созданные слоты представлены на рисунке 1.3.



**Рисунок 1.3 – Созданные классы**

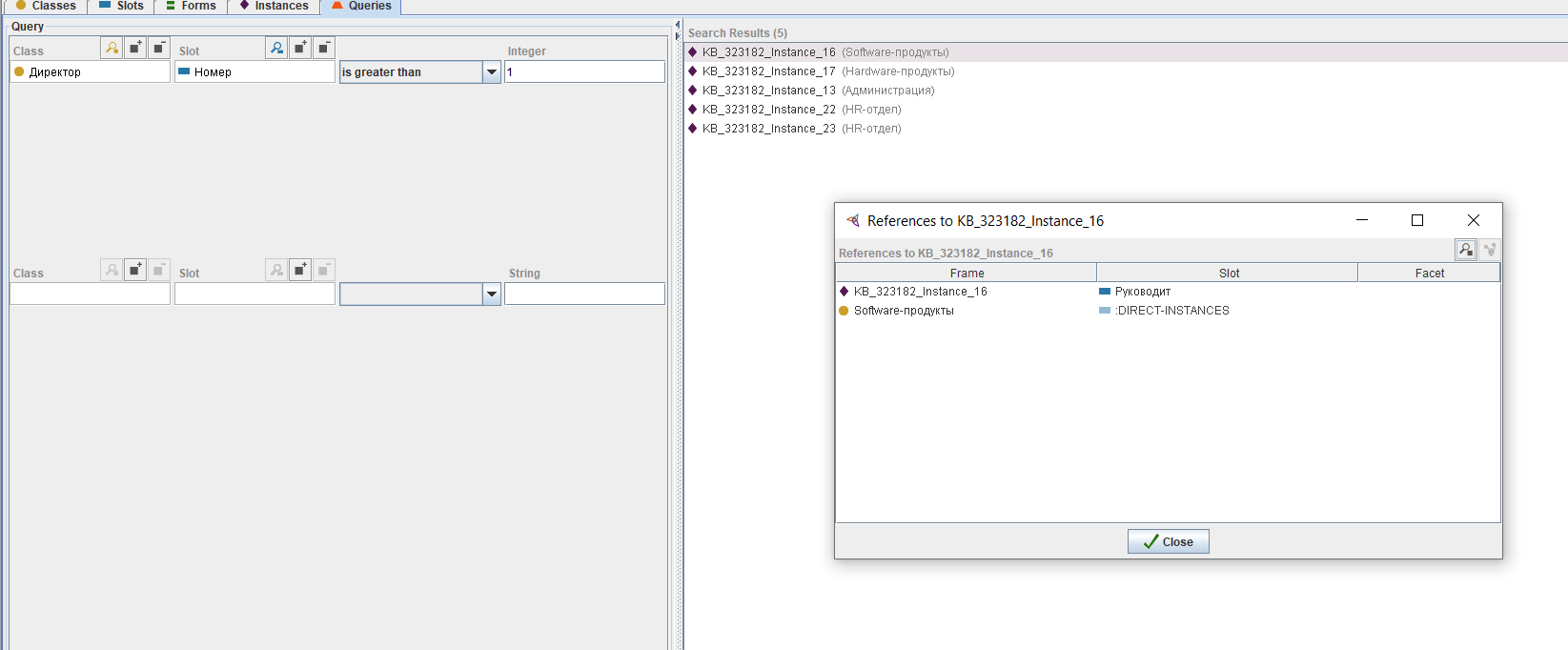
Созданные сущности представлены на рисунке 1.4



**Рисунок 1.4 – Созданные классы**

Для демонстрации правильности создания онтологии были созданы три запроса, выбирающие один из экземпляров класса. Для каждого из экземпляров показан экземпляр другого класса, с которым он связан.

Запрос представлен на рисунке 1.5.

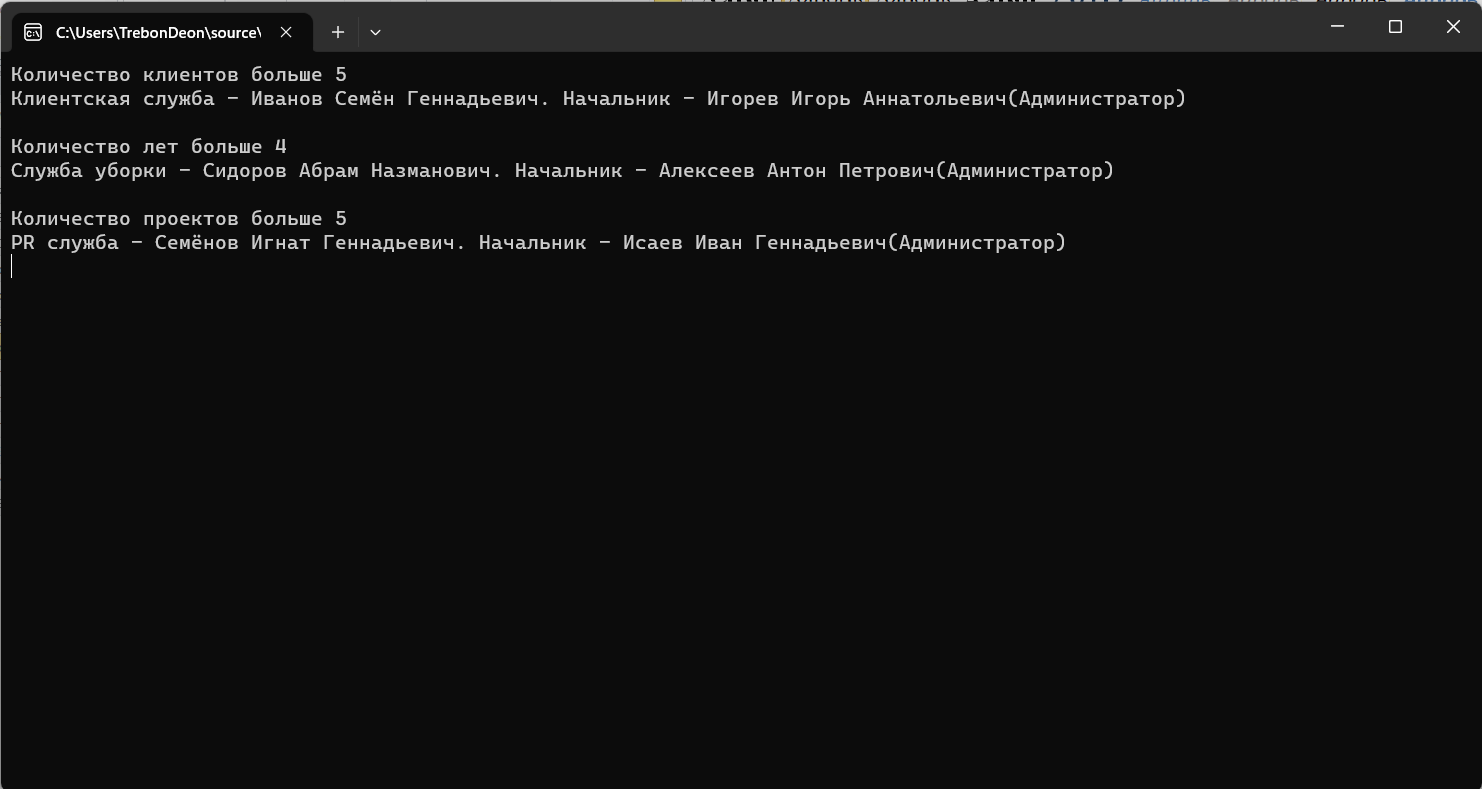


**Рисунок 1.5 – Первый запрос**

## 1.4 Программная реализация

Была реализована онтология на языке Python, используя принципы ООП.

Вывод работы программы представлен на рисунке 1.6.



**Рисунок 1.6 – Вывод работы программы**

Код реализации представлен в приложении А.

# 2 МЕТОД ИМИТАЦИИ ОТЖИГА

## 2.1 Постановка задачи

Очень голодный студент после 5 пар решил посетить все точки быстрого питания в радиусе 10 километров, но дома его ждет собака, которая хочет пойти на прогулку. Необходимо посчитать оптимальный путь студента между точками быстрого питания.

## 2.2 Теоретический раздел

Метод отжига служит для поиска глобального минимума некоторой функции 𝑓(𝑥), заданной для 𝑥 некоторого пространства 𝑆, дискретного или непрерывного. Элементы множества 𝑆 представляют собой состояние воображаемой физической системы («энергетические уровни), а значения функции 𝑓 в этих точках используется как энергия системы 𝐸=𝑓(𝑥). В каждый момент предполагается заданная температура системы 𝑇, как правило, уменьшающая с течением времени. После попадания в состояние 𝑥 при температуре 𝑇, следующее состояние системы выбирается в соответствии с заданным порождающим семейством вероятностных распределений 𝒢(𝑥,𝑇), которое при фиксированных 𝑥 и 𝑇 задает случайный элемент 𝐺(𝑥,𝑇) со значениями в пространстве 𝑆. После генерации нового состояния 𝑥′=𝐺(𝑥,𝑇), система с вероятностью ℎ(Δ𝐸,𝑇) переходит к следующему состоянию 𝑥′, в противном случае процесс генерации 𝑥′ повторяется. Здесь Δ𝐸 обозначает приращение функции 𝑓(𝑥′)−𝑓(𝑥). Величина ℎ(Δ𝐸,𝑇) называется вероятностью принятия нового состояния.

Как правило, в качестве функции ℎ(Δ𝐸,𝑇) выбирается либо точное значение соответствующей физической величины, представленной в формуле (2.1).

ℎ(Δ𝐸,𝑇)=1/1+exp(Δ𝐸/𝑇), (2.1)

либо приближенное значение, представленное в формуле (2.2).

ℎ(Δ𝐸,𝑇)=exp(−Δ𝐸/𝑇). (2.2)

Вторая формула используется наиболее часто. При ее использовании ℎ(Δ𝐸,𝑇) оказывается больше единицы в случае Δ𝐸<0, и тогда соответствующая вероятность считается равной 1[10]. Таким образом, если новое состояние дает лучшее значение оптимизируемой функции, то переход в это состояние перейдет в любом случае. Описание представлено в формуле (2.3).

. (2.3)

Конкретная схема метода отжига задается следующими параметрами:

* Выбором закона изменения температуры 𝑇(𝑘), где 𝑘 – номер шага.
* Выбором порождающего семейства распределений 𝒢(𝑥,𝑇).
* Выбором функции вероятности принятия ℎ(Δ𝐸,𝑇).

## 2.3 Ручной просчет алгоритма

Для решения задачи были выбраны следующие точки:

* 1(21, 80);
* 2(54,53);
* 3(18,89);
* 4(90, 78);
* 5(43, 43).

Были выбраны следующие переходы:

* 1->4;
* 2->5;
* 5->1.

По формуле (2.4) были рассчитаны расстояния между точками:

, (2.4)

* 1-2(42,63);
* 2-3(50,91);
* 3-4(72,83);
* 4-5(58,6);
* 5-1(43,04).

Сумма расстояний – 268,032.

В качестве случайных точек были выбраны следующие точки:

* 50,91;
* 58,60;
* 42,63.

Так как расстояние улучшилось, нужно записать новое лучшее расстояние, после чего сделать перестановки.

Точки и их последовательность после перехода:

* 4(90, 78);
* 2(54,53);
* 3(18,89);
* 1(21, 80);
* 5(43, 43).

По формуле (2.4) были рассчитаны расстояния между точками:

* 4-2(43,83);
* 2-3(50,91);
* 3-1(9,49);
* 1-5(43,04);
* 5-4(58, 60).

Сумма расстояний – 205,87.

Так как расстояние меньше, нужно выбрать его как лучшее, и уменьшить температуру по правилу (), и сделать перестановки.

Точки их последовательность после перехода:

* 4(90, 78);
* 5(43,43);
* 3(18,89);
* 1(21, 80);
* 2(54, 53).

По формуле (2.4) были рассчитаны расстояния между точками:

* 4-5(58,60);
* 5-3(52,35);
* 3-1(9,49);
* 1-2(42,64);
* 2-4(43,83).

Сумма расстояний – 206,909.

Расстояние хуже, поэтому надо сверить его с вероятностью. Необходимо уменьшить температуру. Вероятность расстояния равна 97,95 (). Случайной вероятностью можно выбрать любое число, например, 58,60. Вероятность расстояния больше случайной, поэтому лучшее расстояние будет текущим[5]. После этого были снова сделаны перестановки.

Точки их последовательность после перехода:

* 4(90, 78);
* 1(21,80);
* 3(18,89);
* 5(43, 43);
* 2(54, 53).

По формуле (2.4) были рассчитаны расстояния между точками:

* 4-1(69,03);
* 1-3(9,49);
* 3-5(52,35);
* 5-2(14,87);
* 2-4(43,83).

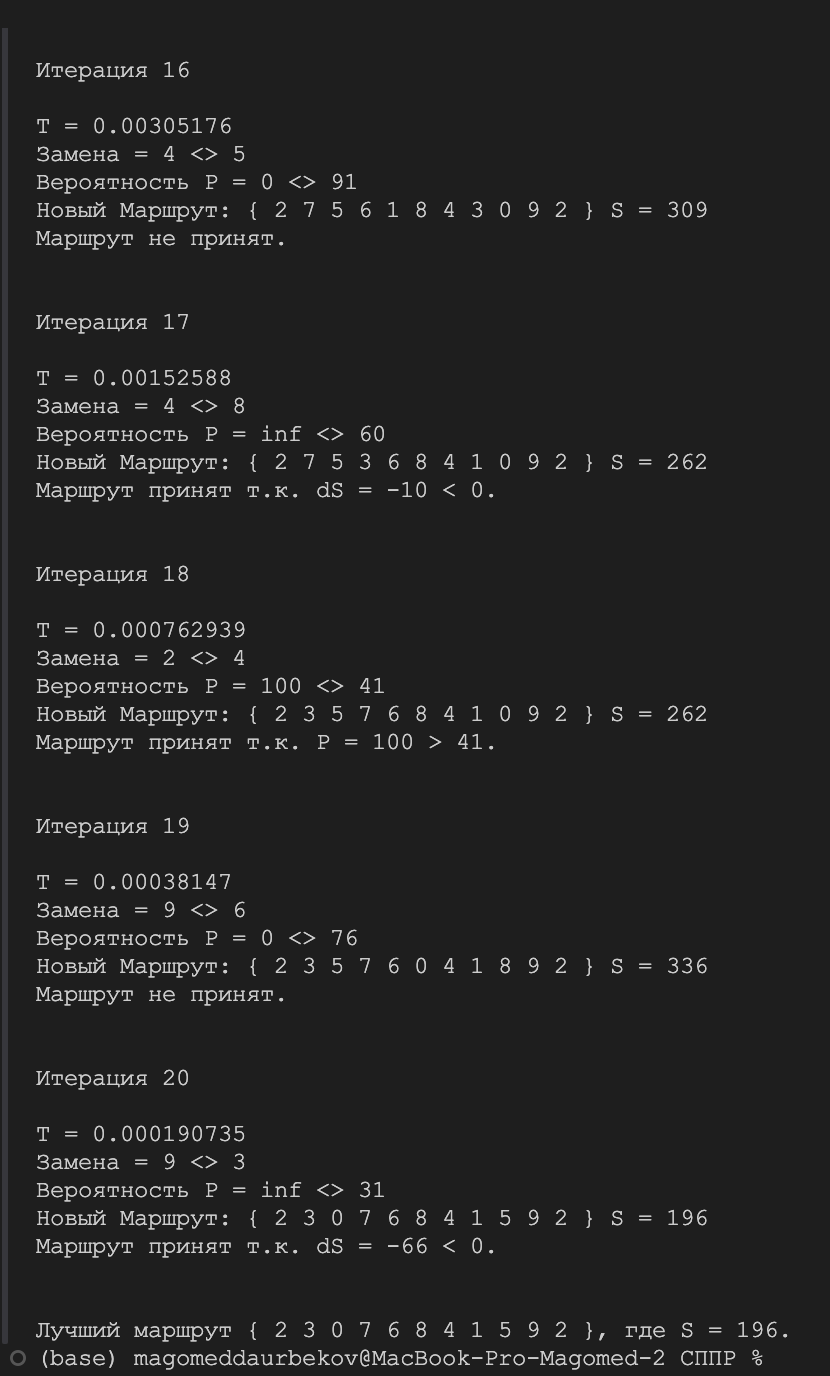
Сумма расстояний – 189,57.

Расстояние лучше предыдущего лучшего, поэтому лучшим становится текущее.

## 2.4 Программная реализация

Программа была реализована на языке C++.

Вывод работы программы представлен на рисунке 2.1.



**Рисунок 2.1 – Вывод работы программы**

Код реализации представлен в приложении Б.

# 3 МЕТОД ОПТИМИЗАЦИИ РОЯ ЧАСТИЦ

## 3.1 Постановка задачи

Найти минимум функции (3.1)

(3.1)

## 3.2 Теоретический раздел

Роевой алгоритм (РА) использует рой частиц, где каждая частица представляет потенциальное решение проблемы. Поведение частицы в гиперпространстве поиска решения все время подстраивается в соответствии со своим опытом и опытом своих соседей. Кроме этого, каждая частица помнит свою лучшую позицию с достигнутым локальным лучшим значением целевой (фитнесс-) функции и знает наилучшую позицию частиц - своих соседей, где достигнут глобальный на текущий момент оптимум. В процессе поиска частицы роя обмениваются информацией о достигнутых лучших результатах и изменяют свои позиции и скорости по определенным правилам на основе имеющейся на текущий момент информации о локальных и глобальных достижениях. При этом глобальный лучший результат известен всем частицам и немедленно корректируется в том случае, когда некоторая частица роя находит лучшую позицию с результатом, превосходящим текущий глобальный оптимум. Каждая частица сохраняет значения координат своей траектории с соответствующими лучшими значениями целевой функции, которые обозначим , которая отражает когнитивную компоненту. Аналогично значение глобального оптимума, достигнутого частицами роя, будем обозначать , которое отражает социальную компоненту. Таким образом, каждая частица роя подчиняется достаточно простым правилам поведения (изложенным ниже формально), которые учитывают локальный успех каждой особи и глобальный оптимум всех особей (или некоторого множества соседей) роя.

Каждая i-я частица характеризуется в момент времени t своей позицией в гиперпространстве и скоростью движения . Позиция частицы изменяется в соответствии с формулой (3.2):

(3.2)

Вектор скорости управляет процессом поиска решения и его компоненты определяются с учетом когнитивной и социальной составляющей по формуле (3.3):

(3.3)

Здесь - 𝑗-ая компонента скорости (𝑗=1,… ,𝑛𝑥) 𝑖-ой частицы в момент времени 𝑡, - 𝑗-я координата позиции 𝑖 -й частицы, 𝑐1 и 𝑐2 – положительные коэффициенты ускорения (часто полагаемые 2), регулирующие вклад когнитивной и социальной компонент, и ~ (0,1) - случайные числа из диапазона [0,1], которые генерируются в соответствии с нормальным распределением и вносят элемент случайности в процесс поиска. Кроме этого - персональная лучшая позиция по 𝑗 -й координате 𝑖-ой частицы, а –лучшая глобальная позиция роя, где целевая функция имеет экстремальное значение.

При решении задач минимизации персональная лучшая позиция в следующий момент времени (𝑡+1) определяется по формуле (3.4):

(3.4)

где фитнесс-функция. Как и в эволюционных алгоритмах, фитнесс-функция измеряет близость текущего решения к оптимуму [2].

Существует два основных подхода в оптимизации роя частиц, под названиями lbest и gbest, отличающиеся топологией соседства, используемой для обмена опытом между частицами. Для модели gbest лучшая частица определяется из всего роя. Глобальная лучшая позиция (gbest) в момент 𝑡 определяется в соответствии с формулой (3.5)

(3.5)

где – общее число частиц в рое.

В процессе поиска решения описанные действия выполняются для каждой частицы роя.

## 3.3 Ручной просчет алгоритма

Для решения задачи были выбраны следующие переменные:

* Количество частиц – 2;
* Количество итераций – 1.
* Верхняя граница переменных x и y – 5;
* Нижняя граница переменных x и y – -5;
* Верхняя граница вектора изменения скорости – 10;
* Нижняя граница вектора изменения скорости – -10.

Случайным образом генерируются точки x (координата по x), y (координата по y), vx (изменение скорости координаты x), vy (изменение скорости координаты y):

* x = [3.74, -2.25];
* y = [1.75, -3.34];
* vx = [4.76, 7.56];
* vy = [2.34, -4.32].

По формуле (3.1) высчитываются значения функции и записываются в массив результатов.

(3.74\*3.74+1.75-11)\* (3.74\*3.74+1.75-11)+(3.74+1.75\*1.75-7) \*(3.74+1.75\*1.75-7)=22.48

(-2.25\*(-2.25)+(-3.34)-11)\* (-2.25\*(-2.25)+(-3.34)-11)+(-2.25+(-3.34) \*(-3.34)-7)\*( -2.25+(-3.34)\*(-3.34)-7)=89.70

По формуле (3.6) рассчитывается изменение скорости:

, (3.6)

где w=0.6,

v – соответствующая координата,

a1, a2 = 2,

r1,r2 – случайное число от 0 до 1,

pbest – лучшее значение текущей точки,

coord – текущая соответствующая координата,

gbest – лучшее текущее значение функции.

(0.6\*3.74+(2\*0.8\*(3.74-3.74))+(2\*0.75\*(22.48-3.74)))=30.354

(0.6\*1.75+(2\*0.8\*(1.75-1.75))+(2\*0.75\*(22.48-1.75)))=33.34

Записываются новые координаты точки

3.74+30.354=34.10

1.75+33.34=35.09

Необходимо проверить, значение функции (3.1) от новых переменных

(34.10\*34.10+35.09-11)\* (34.10\*34.10+35.09-11) + (34.10+35.09\*35.09-7) \* (34.10+35.09\*35.09-7) = 2992322.56

Так как новое значение функции больше предыдущего, то ничего не меняется.

Значения не поменялись, поэтому текущее лучшее значение функции тоже не поменялось.

Просчитываются новые значения точек для второй частицы

(0.6\*(-2.25)+(2\*0.4\*((-2.25)- (-2.25)))+(2\*0.32\*(22.48-(-2.25))))=14.48

(0.6\*(-3.34)+(2\*0.3\*((-3.34)- (-3.34)))+(2\*0.94\*(22.48-(-3.34))))=46.54

Записываются новые координаты точки

-2.25+14.48=12.23

-3.34+46.54=43.2

Необходимо проверить, значение функции от новых переменных

(12.23\*12.23+43.2-11)\* (12.23\*12.23+43.2-11) + (12.23+43.2\*43.2-7) \* (12.23+43.2\*43.2-7) = 3548541.64

Так как новое значение функции больше предыдущего, то ничего не меняется.

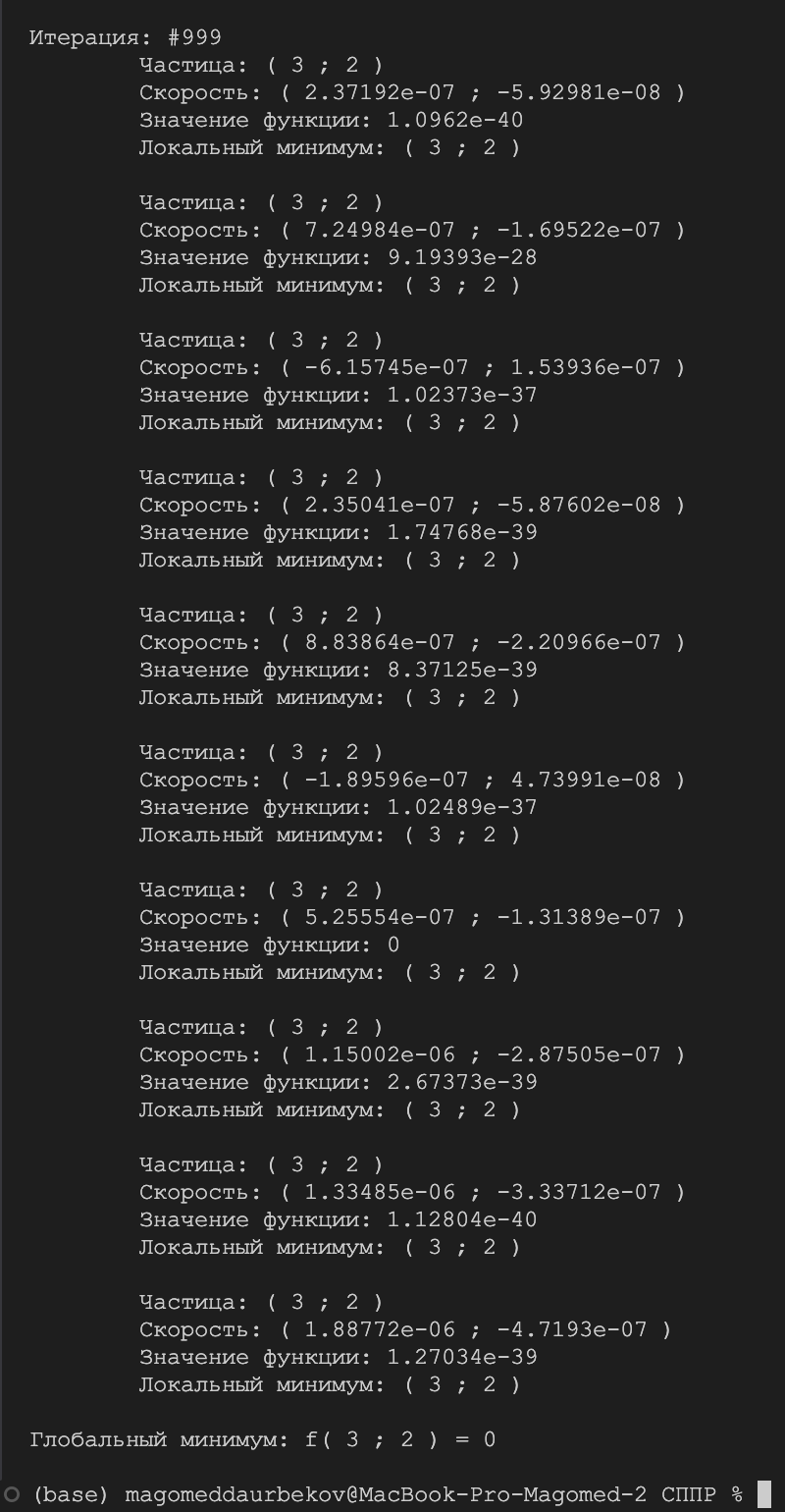
Значения не поменялись, поэтому текущее лучшее значение функции тоже не поменялось.

Лучшее значение функции было 22.48 в координатах 3.74;1.75.

## 3.4 Программная реализация

Реализован код оптимизации роя частиц на языке C++.

Вывод работы программы представлен на рисунке 3.1.



**Рисунок 3.1 – Вывод работы программы**

Код реализации представлен в приложении В.

# 4 МЕТОД ОПТИМИЗАЦИИ МУРАВЬИНОЙ КОЛОНИИ

## 4.1 Постановка задачи

Составить маршрут через граф, проходящий по всем его вершинам с минимальным пройденным путем.

## 4.2 Теоретический раздел

В качестве иллюстрации возьмем задачу поиска кратчайшего пути между двумя узлами графа 𝐺 = (𝑉, 𝐸), где 𝑉– множество узлов (вершин), а 𝐸 – матрица, которая представляет связи между узлами. Пусть = |𝑉| - число узлов в графе.

Обозначим - длину пути в графе, пройденного 𝑘-м муравьем, которая равна числу пройденных дуг (ребер) от первой до последней вершины пути.

С каждой дугой, соединяющей вершины (𝑖,𝑗), ассоциируем концентрацию феромона . Строго говоря, в начальный момент времени концентрация феромона для каждой дуги графа нулевая, но мы для удобства каждой дуге присвоим небольшое случайное число (0). Муравей выбирает следующую дугу пути случайным образом[1]. Множество муравьев k={1,…,n\_k} помещаются в начальную вершину. В каждой итерации ПМА каждый муравей пошагово строит путь до конечной вершины.

При этом в каждой вершине каждый муравей должен выбрать следующую дугу пути. Если j-й муравей находится в -ой вершине,то он выбирает следующую вершину на основе вероятностей перехода (4.1):

(4.1)

Здесь представляет множество возможных вершин, связанных с j-й вершиной, для -го муравья. Если для любого -го узла и 𝑘-го муравья , тогда предшественник узла 𝑖 включается в . В этом случае в пути возможны петли. Эти петли удаляются при достижении конечного города пути. 𝛼 - положительная константа, которая определяет влияние концентрации феромона. Очевидно, большие значения ∝ повышают влияние концентрации феромона. Это особенно существенно в начальной стадии для начальных случайных значений концентрации, что может привести к преждевременной сходимости к субоптимальным решениям. Когда все муравьи построили полный путь от начальной до конечной вершины, удаляются петли в путях, и каждый муравей помечает свой построенный путь, откладывая для каждой дуги феромон в соответствии со следующей формулой (4.2):

(4.2)

Здесь – длина пути, построенного 𝑘-м муравьем в момент времени 𝑡.

Таким образом, для каждой дуги графа концентрация феромона определяется следующим образом (4.3):

(4.3)

где - число муравьев. Из (4.2) следует, что общая концентрация феромона для данной дуги пропорциональна «качеству» путей, в которые входит эта дуга, поскольку откладываемое количество феромона согласно (4.2) отражает «качество» соответствующего пути. В данном случае «качество» обратно пропорционально длине пути (числу дуг, вошедших в путь). Но в общем случае может быть использована и другая мера качества (например, стоимость проезда по данному пути или геометрическое расстояние и т.п.). Пусть обозначает решение в момент 𝑡, и некоторая функция выражает качество решения.

Если не пропорционально качеству решения и все муравьи откладывают одинаковое количество феромона , то существует только один фактор, который зависит от длины пути и способствует выбору коротких путей. Это ведет к двум основным способам оценки качества решений, которые используются в МА:

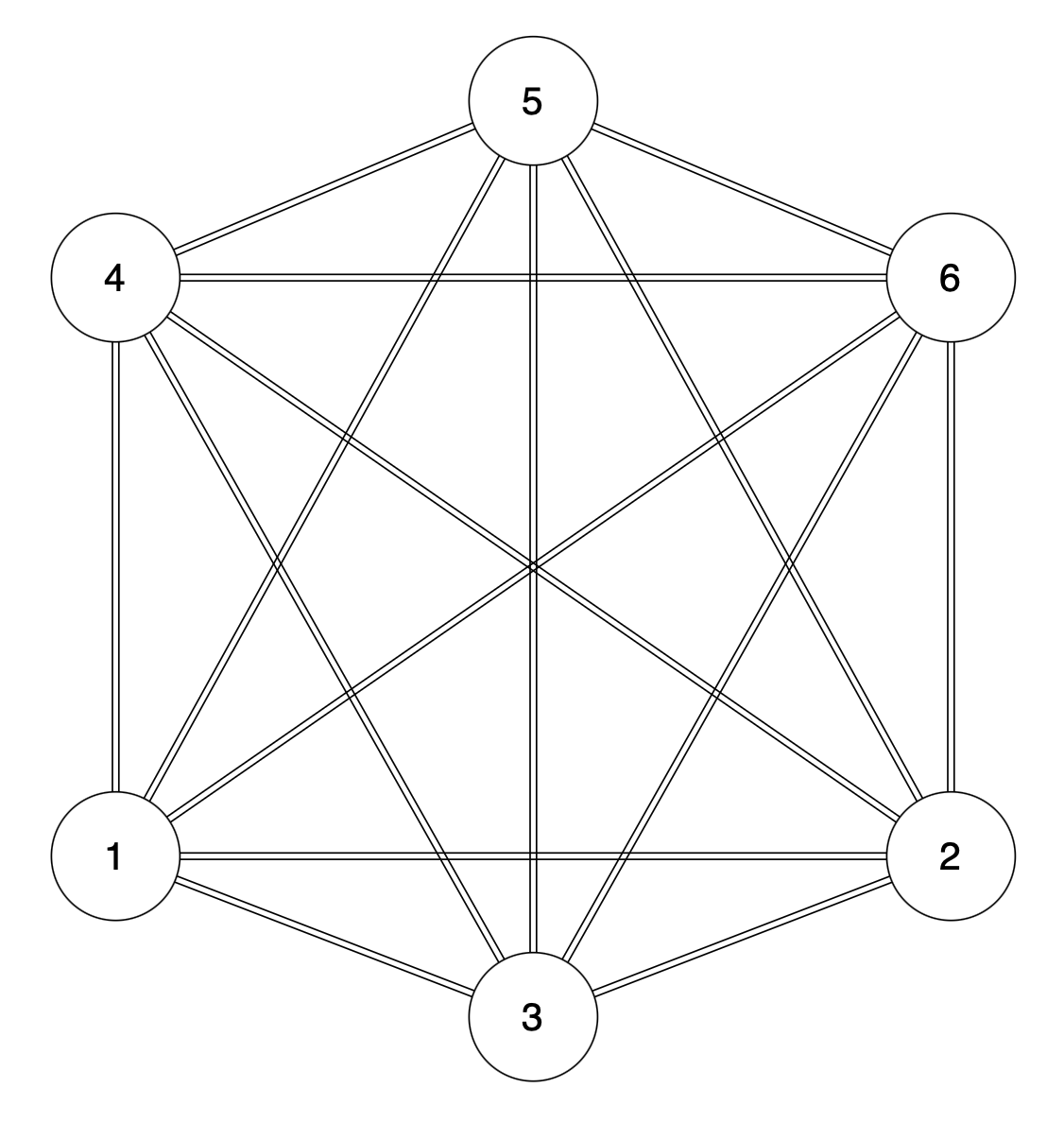
* неявная оценка, где муравьи используют отличие в длине путей

относительно построенных путей другими муравьями;

* явная оценка, количество феромона пропорционально некоторой мере качества построенного решения[13].

## 4.3 Ручной просчет алгоритма

Граф алгоритма представлен на рисунке 4.1. Необходимо пройти весь граф за минимальное пройденное расстояние.



**Рисунок 4.1 – Граф алгоритма**

Матрица смежности для графа выглядит следующим образом:

[[0, 12, 0, 0, 0, 16, 3, 0],

[12, 0, 8, 0, 0, 0, 6, 0],

[0, 8, 0, 44, 0, 0, 8, 4],

[0, 0, 44, 0, 14, 0, 0, 3],

[0, 0, 0, 14, 0, 28, 11, 5],

[16 ,0, 0, 0, 28, 0, 13, 0],

[3, 6, 8, 0, 11, 13, 0, 0],

[0, 0, 4, 3, 5, 0, 0, 0]].

Для ручного просчета генерируется 2 муравья и 2 итерации.

Константа для концентрации равна 2. Коэффициент испарения феромона равен 0.6.

Массив феромонов выглядит следующим образом:

[[0, 1, 0, 0, 0, 1, 1, 0],

[1, 0, 1, 0, 0, 0, 1, 0],

[0, 1, 0, 1, 0, 0, 1, 1],

[0, 0, 1, 0, 1, 0, 0, 1],

[0, 0, 0, 1, 0, 1, 1, 1],

[1, 0, 0, 0, 1, 0, 1, 0],

[1, 1, 1, 0, 1, 1, 0, 0],

[0, 0, 1, 1, 1, 0, 0, 0]].

Считаются переходы для 1 муравья. Из текущей нулевой вершины:

0->1:

0->5:

0->6:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.33, то переход в 1 вершину. Если число больше 0.33 и меньше 0.66, переход в 5 вершину. Если число больше 0.66, переход в 6 вершину. Сгенерированное число равно 0.78. Переход в 6 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 6:

6->1:

6->2:

6->4:

6->5:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.25, то переход в 1 вершину. Если число больше 0.25 и меньше 0.5, переход в 2 вершину. Если число больше 0.5 и меньше 0.75, переход в 4 вершину. Если число больше 0.75, переход в 5 вершину. Сгенерированное число равно 0.6. Переход в 4 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 4:

4->5:

4->3:

4->7:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.33, то переход в 5 вершину. Если число больше 0.33 и меньше 0.66, переход в 3 вершину. Если число больше 0.66, переход в 7 вершину. Сгенерированное число равно 0.81. Переход в 7 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 7:

7->2:

7->3:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.5, то переход в 2 вершину. Если число больше 0.5, переход в 3 вершину. Сгенерированное число равно 0.64. Переход в 3 вершину.

Муравей дошел до нужной точки.

Переходы муравья : 0->6->4->7->3

Таким же образом считаются переходы для 2 муравья.

Считаются переходы для 2 муравья. Из текущей нулевой вершины:

0->1:

0->5:

0->6:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.33, то переход в 1 вершину. Если число больше 0.33 и меньше 0.66, переход в 5 вершину. Если число больше 0.66, переход в 6 вершину. Сгенерированное число равно 0.4. Переход в 5 вершину.

Считаются переходы для 2 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 5:

5->6:

5->4:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.5, то переход в 6 вершину. Если число больше 0.5, переход в 4 вершину. Сгенерированное число равно 0.64. Переход в 4 вершину.

Считаются переходы для 2 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 4:

4->6:

4->3:

4->7:

Генерируется случайное число [0,1] для получения точки перехода[12]:

Если число меньше 0.33, то переход в 6 вершину. Если число больше 0.33 и меньше 0.66, переход в 3 вершину. Если число больше 0.66, переход в 7 вершину. Сгенерированное число равно 0.81. Переход в 7 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 7:

7->2:

7->3:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.5, то переход в 2 вершину. Если число больше 0.5, переход в 3 вершину. Сгенерированное число равно 0.64. Переход в 3 вершину.

Муравей дошел до нужной точки.

Переходы муравья : 0->5->4->7->3

Все муравьи добрались до конечной точки. Необходимо выветрить феромон на всех дугах графа: . Обновленный массив феромонов выглядит следующим образом:

[[0, 0.4, 0, 0, 0, 0.4, 0.4, 0],

[0.4, 0, 0.4, 0, 0, 0, 0.4, 0],

[0, 0.4, 0, 0.4, 0, 0, 0.4, 0.4],

[0, 0, 0.4, 0, 0.4, 0, 0, 0.4],

[0, 0, 0, 0.4, 0, 0.4, 0.4, 0.4],

[0.4, 0, 0, 0, 0.4, 0, 0.4, 0],

[0.4, 0.4, 0.4, 0, 0.4, 0.4, 0, 0],

[0, 0, 0.4, 0.4, 0.4, 0, 0, 0]].

Каждый муравей возвращается обратно тем же маршрутом, что он пришел в конечную точку маршрута, оставляя при этот часть феромона на той дуге, по которой он прошел[6]. На нулевой итерации для первого муравья можно рассчитать одно значение для каждой дуги, поскольку концентрация феромона на всех дугах одинакова:

Для дуг 1 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.4, 0, 0, 0, 0.4, 0.44, 0],

[0.4, 0, 0.4, 0, 0, 0, 0.4, 0],

[0, 0.4, 0, 0.4, 0, 0, 0.4, 0.4],

[0, 0, 0.4, 0, 0.4, 0, 0, 0.4],

[0, 0, 0, 0.4, 0, 0.4, 0.4, 0.44],

[0.4, 0, 0, 0, 0.4, 0, 0.4, 0],

[0.4, 0.4, 0.4, 0, 0.44, 0.4, 0, 0],

[0, 0, 0.4, 0.44, 0.4, 0, 0, 0]].

Для дуги 0->5 2 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.4, 0, 0, 0, 0.42, 0.44, 0],

[0.4, 0, 0.4, 0, 0, 0, 0.4, 0],

[0, 0.4, 0, 0.4, 0, 0, 0.4, 0.4],

[0, 0, 0.4, 0, 0.4, 0, 0, 0.4],

[0, 0, 0, 0.4, 0, 0.4, 0.4, 0.44],

[0.4, 0, 0, 0, 0.4, 0, 0.4, 0],

[0.4, 0.4, 0.4, 0, 0.44, 0.4, 0, 0],

[0, 0, 0.4, 0.44, 0.4, 0, 0, 0]].

Для дуги 5->4 2 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.4, 0, 0, 0, 0.42, 0.44, 0],

[0.4, 0, 0.4, 0, 0, 0, 0.4, 0],

[0, 0.4, 0, 0.4, 0, 0, 0.4, 0.4],

[0, 0, 0.4, 0, 0.4, 0, 0, 0.4],

[0, 0, 0, 0.4, 0, 0.4, 0.4, 0.44],

[0.4, 0, 0, 0, 0.42, 0, 0.4, 0],

[0.4, 0.4, 0.4, 0, 0.44, 0.4, 0, 0],

[0, 0, 0.4, 0.44, 0.4, 0, 0, 0]].

Для дуги 4->7 2 муравья:

Для дуги 7->3 значение будет таким же.

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.4, 0, 0, 0, 0.42, 0.44, 0],

[0.4, 0, 0.4, 0, 0, 0, 0.4, 0],

[0, 0.4, 0, 0.4, 0, 0, 0.4, 0.4],

[0, 0, 0.4, 0, 0.4, 0, 0, 0.4],

[0, 0, 0, 0.4, 0, 0.4, 0.4, 0.46],

[0.4, 0, 0, 0, 0.42, 0, 0.4, 0],

[0.4, 0.4, 0.4, 0, 0.44, 0.4, 0, 0],

[0, 0, 0.4, 0.46, 0.4, 0, 0, 0]].

Первая итерация закончилась.

Начиная вторую итерацию, необходимо снова рассчитывать переходы каждого муравья.

Считаются переходы для 1 муравья. Из текущей нулевой вершины:

0->1:

0->5:

0->6:

Уже на данном этапе заметно, что муравей с большей вероятностью выберет переход в 6 вершину. Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.30, то переход в 1 вершину. Если число больше 0.30 и меньше 0.63, переход в 5 вершину. Если число больше 0.63, переход в 6 вершину. Сгенерированное число равно 0.66. Переход в 6 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 6:

6->1:

6->2:

6->4:

6->5:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.24, то переход в 1 вершину. Если число больше 0.24 и меньше 0.48, переход в 2 вершину. Если число больше 0.48 и меньше 0.77, переход в 4 вершину. Если число больше 0.77, переход в 5 вершину. Сгенерированное число равно 0.5. Переход в 4 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 4:

4->5:

4->3:

4->7:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.31, то переход в 5 вершину. Если число больше 0.31 и меньше 0.62, переход в 3 вершину. Если число больше 0.62, переход в 7 вершину. Сгенерированное число равно 0.93. Переход в 7 вершину.

Считаются переходы для 1 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 7:

7->2:

7->3:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.43, то переход в 2 вершину. Если число больше 0.43, переход в 3 вершину. Сгенерированное число равно 0.48. Переход в 3 вершину.

Муравей дошел до нужной точки.

Переходы муравья : 0->6->4->7->3

Таким же образом считаются переходы для 2 муравья.

Считаются переходы для 2 муравья. Из текущей нулевой вершины:

0->1:

0->5:

0->6:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.30, то переход в 1 вершину. Если число больше 0.30 и меньше 0.63, переход в 5 вершину. Если число больше 0.63, переход в 6 вершину. Сгенерированное число равно 0.4. Переход в 5 вершину[15].

Считаются переходы для 2 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 5:

5->6:

5->4:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.48, то переход в 6 вершину. Если число больше 0.48, переход в 4 вершину. Сгенерированное число равно 0.64. Переход в 4 вершину.

Считаются переходы для 2 муравья. Муравей может пойти в любую вершину кроме той, из которой он пришел. Из текущей вершины 4:

4->3:

4->6:

4->7:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.30, то переход в 3 вершину. Если число больше 0.30 и меньше 0.60, переход в 6 вершину. Если число больше 0.60, переход в 7 вершину. Сгенерированное число равно 0.81. Переход в 7 вершину.

7->2:

7->3:

Генерируется случайное число [0,1] для получения точки перехода:

Если число меньше 0.43, то переход в 2 вершину. Если число больше 0.43, переход в 3 вершину. Сгенерированное число равно 0.64. Переход в 3 вершину.

Муравей дошел до нужной точки.

Переходы муравья : 0->5->4->7->3

Все муравьи добрались до конечной точки. Необходимо выветрить феромон на всех дугах графа:

Для дуг равных 0.4: .

Для дуг равных 0.42: .

Для дуг равных 0.44: .

Для дуг равных0.46: .

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.168, 0.176, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.184],

[0.16, 0, 0, 0, 0.168, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.176, 0.16, 0, 0],

[0, 0, 0.16, 0.184, 0.16, 0, 0, 0]].

Каждый муравей возвращается обратно тем же маршрутом, что он пришел в конечную точку маршрута, оставляя при этот часть феромона на той дуге, по которой он прошел.

Для дуги 0->6 1 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.168, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.184],

[0.16, 0, 0, 0, 0.168, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.176, 0.16, 0, 0],

[0, 0, 0.16, 0.184, 0.16, 0, 0, 0]].

Для дуги 6->4 1 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.168, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.184],

[0.16, 0, 0, 0, 0.168, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.184, 0.16, 0, 0, 0]].

Для дуги 4->7 1 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.168, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.230],

[0.16, 0, 0, 0, 0.168, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.184, 0.16, 0, 0, 0]].

Для дуги 7->3 1 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.168, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.230],

[0.16, 0, 0, 0, 0.168, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.230, 0.16, 0, 0, 0]].

Для дуги 0->5 2 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.187, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.230],

[0.16, 0, 0, 0, 0.168, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.230, 0.16, 0, 0, 0]].

Для дуги 5->4 2 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.187, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.230],

[0.16, 0, 0, 0, 0.187, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.230, 0.16, 0, 0, 0]].

Для дуги 4->7 2 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.187, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.25],

[0.16, 0, 0, 0, 0.187, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.230, 0.16, 0, 0, 0]].

Для дуги 7->3 2 муравья:

Обновленный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.187, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.25],

[0.16, 0, 0, 0, 0.187, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

[0, 0, 0.16, 0.25, 0.16, 0, 0, 0]].

Финальный массив феромонов выглядит следующим образом:

[[0, 0.16, 0, 0, 0, 0.187, 0.221, 0],

[0.16, 0, 0.16, 0, 0, 0, 0.16, 0],

[0, 0.16, 0, 0.16, 0, 0, 0.16, 0.16],

[0, 0, 0.16, 0, 0.16, 0, 0, 0.16],

[0, 0, 0, 0.16, 0, 0.16, 0.16, 0.25],

[0.16, 0, 0, 0, 0.187, 0, 0.16, 0],

[0.16, 0.16, 0.16, 0, 0.221, 0.16, 0, 0],

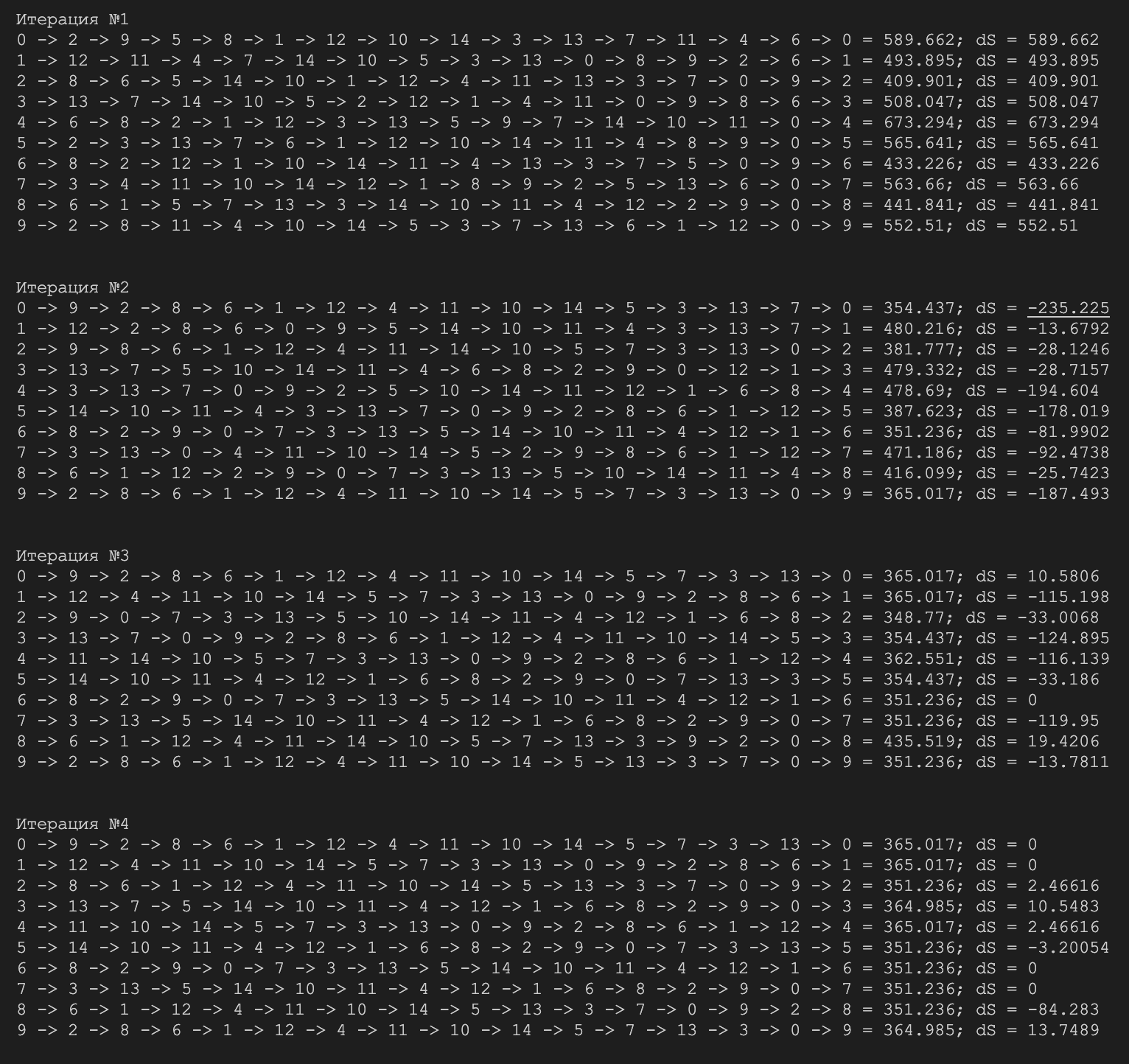
[0, 0, 0.16, 0.25, 0.16, 0, 0, 0]].

Наиболее вероятен переход с большей концентрацией феромона. Основываясь на массиве феромона, можно предположить, что на данной итерации лучшим маршрутом будет 0->6(0.221)->4(0.221)->7(0.25)->3(0.25).

## 4.4 Программная реализация

Реализован код оптимизации муравьиной колонии на языке C++.

Фрагмент вывода работы программы представлен на рисунке 4.2.



**Рисунок 4.2 – Вывод работы программы**

На рисунке 4.2 показаны первые 4 итерации из 10, в которых виднен процесс поиска муравьиной колоней оптимального маршрута обхода всего графа за минимальный пройденный путь. За 4 итерации видно, что все больше муравьев сводятся к решению, которое имеет пройденный путь равный 351.236. На 10 итерации вся колония будет следовать этому маршруту, что и делает его самым оптимальным.

Код реализации представлен в приложении Г.

# 5 АЛГОРИТМ ПЧЕЛИНОЙ КОЛОНИИ

## 5.1 Постановка задачи

Найти минимум функции (5.1)

(5.1)

## 5.2 Теоретический раздел

В алгоритме каждое решение представляется в виде пчелы, которая знает (хранит) расположение (координаты или параметры многомерной функции) какого-то участка поля, где можно добыть нектар.

В начале алгоритма в точки, описываемые случайными координатами, отправляется некоторое количество пчел-разведчиков (пусть будет S пчел, от слова scout). Таким образом:

1 шаг: Необходимо задать количество пчел-разведчиков S. В точки со случайными координатами , отправляются пчелы-разведчики, где β – номер пчелы разведчика, , а 0 обозначает номер итерации в данный момент времени. Считаются значения целевой функции 𝐹(𝑋) в этих точках.

2 шаг: В области 𝐷 с помощью полученных значений выделяют два вида участков (подобластей) .

Первый вид содержит 𝑛 лучших участков, которые соответствуют наибольшим или наименьшим значениям целевой функции, в зависимости от того решается задача на минимум или на максимум функции[14].

Второй 𝑚 перспективных участков, соответствующих значениям целевой функции, наиболее близким к наилучшим значениям.

Подобласть является подобластью локального поиска, представляющая собой гиперкуб в пространстве с центром в точке . Длина его сторон равна , где – параметр, называемый размером области локального поиска.

3 шаг: Сравнивается евклидово расстояние между двумя агентами-разведчиками. Для точек и евклидово расстояние считается по формуле (5.2)

(5.2)

Если евклидово расстояние оказывается меньше фиксированной величины, то возможны два следующих варианта метода:

* поставить в соответствие этим агентам два различных пересекающихся участка (лучших и/или перспективных);
* поставить в соответствие тем же агентам один участок, центр которого находится в точке, соответствующей агенту с большим значением целевой функции. Из этих двух вариантов в работе используется второй вариант.

4 шаг: В каждый из лучших и перспективных участков посылается по N и по M агентов, соответственно. Координаты этих агентов в указанных участках определяются случайным образом[8].

5 шаг: В полученных точках снова считается значение целевой функции 𝐹(𝑋), снова выбирается наибольшее или наименьшее значение. Точка, в которой значение функции является максимальным, становится центром новой подобласти.

6 шаг: Шаги 4 и 5 повторяются до тех пор, пока не будет получен искомый результат, если такой известен, либо до тех пор, пока полученные значения координат экстремумов и значений функции в них не повторятся раз, где — параметр останова.

## 5.3 Ручной просчет алгоритма

Для решения задачи были выбраны следующие переменные:

* Количество итераций: 2;
* Количество пчел-разведчиков: 6;
* Количество пчел, отправляемых на лучшие участки: 2;
* Количество пчел, отправляемых на другие выбранные участки: 1;
* Количество лучших участков: 2:
* Количество выбранных участков: 1;
* Размер области для каждого участка: 10;

Случайным образом генерируются точки x (координата по x), y (координата по y) для каждой из пчёл разведчиков и высчитывается значение функции для этой точки

* x = [3.74, -2.25, 4.28, -3.48, 1.27, -0.75];
* y = [1.75, -3.34, 2.44, 1.49, -5.0, 0].

По формуле (1) высчитываются значения функции для каждой пчелы.

(3.74\*3.74+1.75-11)\* (3.74\*3.74+1.75-11)+(3.74+1.75\*1.75-7) \*(3.74+1.75\*1.75-7)=22.48,

(-2.25\*(-2.25)+(-3.34)-11)\* (-2.25\*(-2.25)+(-3.34)-11)+(-2.25+(-3.34) \*(-3.34)-7)\*( -2.25+(-3.34)\*(-3.34)-7)=89.70,

(4.28\*(4.28)+(2.44)-11)\* (4.28\*(4.28)+( 2.44)-11)+(4.28+(2.44)\*( 2.44)-7)\*(4.28+(2.44)\*( 2.44)-7)=105.683,

(-3.48\*(-3.48)+( 1.49)-11)\* (-3.48\*(-3.48)+( 1.49)-11)+( -3.48+(1.49) \*(1.49)-7)\*( -3.48+(1.49)\*( 1.49)-7)=74.988,

(-0.75\*(-0.75)+(0)-11)\* (-0.75\*(-0.75)+(0)-11)+( -0.75+(0) \*(0)-7)\*( -0.75+(0)\*(0)-7)=169.004,

(1.27\*(1.27)+(-5)-11)\* (1.27\*(1.27)+(-5)-11)+(1.27+(-5)\*(-5)-7)\*( 1.27+(-5)\*(-5)-7)=578.322.

Таким образом, имеется:

F(3.74, 1.75) = 22.48,

F(-3.48, 1.49) = 74.99,

F(-2.25, -3.34) = 89.70,

F(4.28, 2.44) = 105.68,

F(-0.75, 0) = 169.01,

F(1.27, -5) = 578.322.

Значения отсортированы по возрастанию целевой функции

Согласно установленным параметрам выбираются две лучшие точки:

F(3.74, 1.75) = 22.48,

F(-3.48, 1.49) = 74.99.

А также определяется один перспективный участок:

F(-2.25, -3.34) = 89.70.

В окрестности лучших точек будет отправлено по две пчелы.

Определяются границы участка для лучших и перспективных точек:

[3.74 – 10 = -6.26; 3.74+10=13.74],

[1.75-10 = -8.25;1.75+10=11.75],

[-3.48-10=-13.48;-3.48+10=6.52],

[1.49-10=-8.51;1.49+10=11.49],

[-2.25-10=-12.25;-2.25+10=7.75],

[-3.34-10=-13.34;-3.34+10=6.66].

В каждый из лучших интервалов отправляем по две пчелы, а в перспективный одну.

F(10.28, -6.34)=9693.80,

F(2.38,8.34)=4225.66,

F(-8.34,7.24)=5703.81,

F(4.44,2.35)=131.18,

F(5.55, -7.34)=2903.76.

На этом, первая итерация закончена. Текущий список значений функции берется как новый для следующей итерации.

Таким образом, имеется:

F(4.44,2.35)=131.18,

F(5.55, -7.34)=2903.76,

F(2.38,8.34)=4225.66,

F(-8.34,7.24)=5703.81,

F(10.28, -6.34)=9693.80.

Значения отсортированы по возрастанию целевой функции.

Согласно установленным параметрам выбираются две лучшие точки:

F(4.44,2.35)=131.18,

F(5.55, -7.34)=2903.76,

А также определяется один перспективный участок:

F(2.38,8.34)=4225.66,

В окрестности лучших точек будет отправлено по две пчелы.

Определяются границы участка для лучших и перспективных точек:

[4.44-10=-5.56;4.44+10=14.44],

[2.35-10=-7.65;2.35+10=12.35],

[5.55-10=-4.45;5.55+10=15.55],

[-7.34-10=-17.34;-7.34+10=2.66],

[2.38-10=-7.62;2.38+10=12.38],

[8.34-10=-1.66;8.34+10=18.34].

В каждый из лучших интервалов отправляем по две пчелы, а в перспективный одну.

F(3.75, 2.44)= 37.59,

F(-3.28,-3)=12.15,

F(11.24,2)=13836.01,

F(3.24,2.66)=15.65,

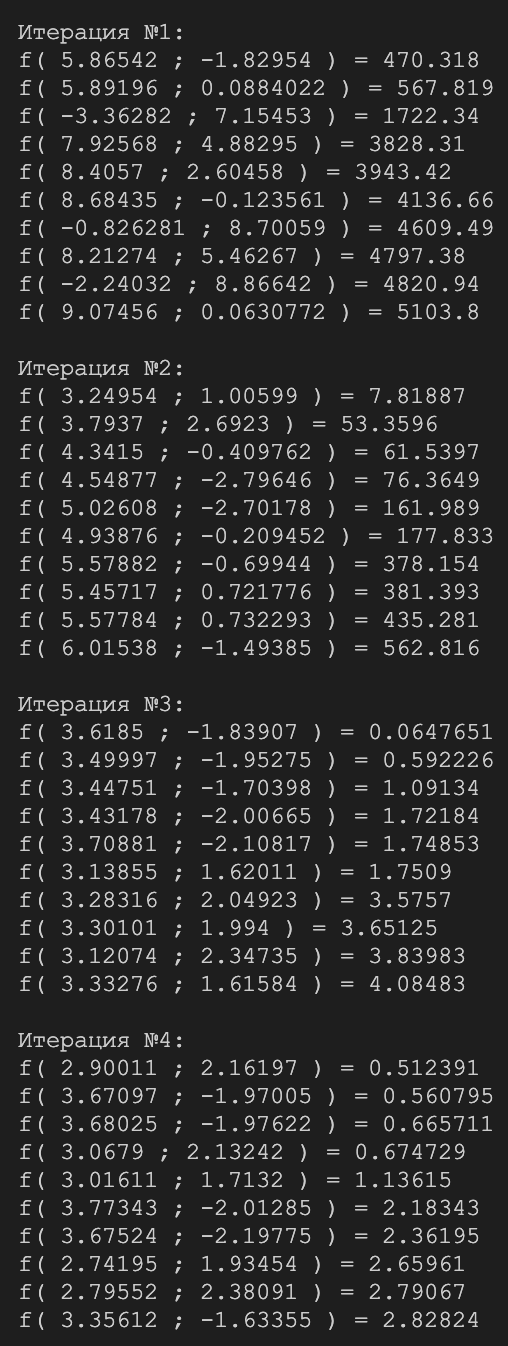
F(6, 12)=21818.

Таким образом, лучшее решение на данной итерации это -3.28, -3.

## 5.4 Программная реализация

Реализован код оптимизации пчелиной колонии на языке C++.

Вывод работы программы представлен на рисунке 5.1.



**Рисунок 5.1 – Вывод работы программы**

Код реализации представлен в приложении Д.

# ЗАКЛЮЧЕНИЕ

Структурный анализ данных - это процесс изучения данных, чтобы выявить в них закономерности, паттерны и связи между ними. Это важный инструмент для принятия решений в современном мире, где данные играют ключевую роль во многих областях, таких как бизнес, наука, медицина и т.д.

В данной курсовой работе рассмотрены различные подходы к анализу данных, такие как статистический анализ, машинное обучение и искусственный интеллект. Каждый из этих подходов имеет свои преимущества и недостатки и может быть использован в зависимости от конкретной задачи.

Также рассмотрены различные методы и инструменты для обработки и структурирования данных. Каждый из этих инструментов имеет свои особенности и может быть использован в зависимости от конкретной задачи.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сорокин А. Б. Введение в профессиональную деятельность. [Электронный ресурс] учебное пособие / А. Б. Сорокин, Л. М. Железняк . — М.: РТУ МИРЭА, 2022.
2. Сорокин А. Б. Введение в роевой интеллект: теория, расчет и приложения. [Электронный ресурс] учебно-метод. пособие / А. Б. Сорокин . — М.: РТУ МИРЭА, 2019.
3. Сорокин А. Б. Сверточные нейронные сети: примеры реализаций. [Электронный ресурс] учебно-метод. пособие / А. Б. Сорокин, Л. М. Железняк, Е. А. Зикеева . — М.: РТУ МИРЭА, 2020.
4. С чего начинаются онтологии — https://habr.com/ru/articles/140696/ (Дата обращения: 07.10.2023).
5. Просто о сложном: Систематизация знаний с помощью онтологий на примере борща и бетона — https://habr.com/ru/companies/teamly/articles/749366/ (Дата обращения: 06.10.2023).
6. С чего начинаются онтологии — https://habr.com/ru/articles/140696/ (Дата обращения: 07.10.2023).
7. Просто о сложном: Систематизация знаний с помощью онтологий на примере борща и бетона — https://habr.com/ru/companies/teamly/articles/749366/ (Дата обращения: 06.10.2023).
8. Естественные алгоритмы. Алгоритм поведения роя пчёл — https://habr.com/ru/articles/104055/ (Дата обращения: 29.11.2023).
9. Ползай, как муравей, летай, как пчела: алгоритмы, которые придумала сама природа — https://skillbox.ru/media/code/polzay-kak-muravey-letay-kak-pchela/?ysclid=lppwuzbdjd132748989 (Дата обращения: 01.12.2023).
10. Муравьиный алгоритм | Задача коммивояжёра — https://habr.com/ru/companies/timeweb/articles/754462/ (Дата обращения: 16.11.2023).
11. Муравьи и Python: ищем самые короткие пути — https://vc.ru/newtechaudit/353372-muravi-i-python-ishchem-samye-korotkie-puti (Дата обращения: 15.11.2023).
12. Алгоритм роя частиц. Описание и реализации на языках Python и C# — https://jenyay.net/Programming/ParticleSwarm (Дата обращения: 26.10.2023).
13. Алгоритм роя частиц — https://habr.com/ru/articles/105639/ (Дата обращения: 24.10.2023).
14. Введение в оптимизацию. Имитация отжига — https://habr.com/ru/articles/209610/ (Дата обращения: 15.10.2023).
15. Метод отжига — https://algorithmica.org/ru/annealing (Дата обращения: 15.10.2023).

# ПРИЛОЖЕНИЯ

**Приложение А - Код создания онтологии**

**Приложение Б – Код метода имитации отжига**

**Приложение В – Код метода оптимизации роя частиц**

**Приложение Г – Код метода оптимизации муравьиной колонии**

**Приложение Д – Код алгоритма пчелиной колонии**

**Приложение А**

*Листинг А.1 - Код онтологии*

|  |
| --- |
| using System;  using System.Collections.Generic;  namespace ConsoleApp1  {  public class Director  {  private int ID;  private string name;  private float money;  public Director(int ID, string name, float money)  {  this.ID = ID;  this.name = name;  this.money = money;  }  public string getName()  {  return this.name;  }  public int getID()  {  return this.ID;  }  public float getMoney()  {  return this.money;  }  }  public class Serv : Director  {  public Serv(int ID, string name, float money) : base(ID, name, money)  {  }  }  public class Client : Serv  {  private int counts\_of\_clients;  public Client(int ID, string name, float money,int counts\_of\_clients) : base(ID, name, money)  {  this.counts\_of\_clients = counts\_of\_clients;  }  public int getCountClients()  {  return this.counts\_of\_clients;  }  }  public class Cleaning : Serv  {  private int years;  public Cleaning(int ID, string name, float money, int years) : base(ID, name, money)  {  this.years = years; |

*Продолжение листинга А.1*

|  |
| --- |
| }  public int getYears()  {  return this.years;  }  }  public class Administartor : Director  {  private object heads\_of;  public Administartor(int ID, string name, float money, object heads\_of) : base(ID, name, money)  {  this.heads\_of = heads\_of;  }  public object getAd()  {  return this.heads\_of;  }  }  public class PR : Director  {  private int counts\_of\_project;  public PR(int ID, string name, float money, int counts\_of\_project) : base(ID, name, money)  {  this.counts\_of\_project = counts\_of\_project;  }  public int getCountProject()  {  return this.counts\_of\_project;  }  }  class Program  {  static void Main(string[] args)  {  Director Petrov = new Director(1, "Петров Иван Иванович", 50000);  Client Ivanov = new Client(2, "Иванов Семён Геннадьевич", 20000, 10);  Cleaning Sidorov = new Cleaning(3, "Сидоров Абрам Назманович", 10000, 5);  PR Semenov = new PR(4, "Семёнов Игнат Геннадьевич", 35000, 6);  Administartor Igorev = new Administartor(5,"Игорев Игорь Аннатольевич", 60000, Ivanov);  Administartor Alekseev = new Administartor(6, "Алексеев Антон Петрович", 60000, Sidorov);  Administartor Isaev = new Administartor(7, "Исаев Иван Геннадьевич", 60000, Semenov);  List<Administartor> administrators = new List<Administartor>();  administrators.Add(Igorev);  administrators.Add(Alekseev);  administrators.Add(Isaev);  List<Client> c = new List<Client>();  List<Director> d = new List<Director>();  List<Cleaning> cle = new List<Cleaning>(); |

*Продолжение листинга А.1*

|  |
| --- |
| List<PR> pr = new List<PR>();  d.Add(Petrov);  c.Add(Ivanov);  cle.Add(Sidorov);  pr.Add(Semenov);  Console.WriteLine("Количество клиентов больше 5");  foreach(Client cli in c)  {  if(cli.getCountClients()>5)  {  Console.Write("Клиентская служба - " + cli.getName() + ". Начальник - ");  foreach (Administartor a in administrators)  {  if (a.getAd() == cli)  {  Console.WriteLine(a.getName()+"(Администратор)");  }  }  }  }  Console.WriteLine("\nКоличество лет больше 4");  foreach (Cleaning cl in cle)  {  if (cl.getYears() > 4)  {  Console.Write("Служба уборки - " + cl.getName() + ". Начальник - ");  foreach (Administartor a in administrators)  {  if (a.getAd() == cl)  {  Console.WriteLine(a.getName() + "(Администратор)");  }  }  }  }  Console.WriteLine("\nКоличество проектов больше 5");  foreach (PR p in pr)  {  if (p.getCountProject() > 5)  {  Console.Write("PR служба - " + p.getName() + ". Начальник - ");  foreach (Administartor a in administrators)  {  if (a.getAd() == p)  {  Console.WriteLine(a.getName() + "(Администратор)");  foreach (Administartor a in administrators) |

*Окончание листинга А.1*

|  |
| --- |
| {  if (a.getAd() == p)  {  Console.WriteLine(a.getName() + "(Администратор)");  }  }  }  }  Console.ReadLine();  }  }  } |

**Приложение Б**

*Листинг Б - Код метода имитации отжига*

|  |
| --- |
| #include <algorithm>  #include <random>  #include <iostream>  #include <vector>  #include <numeric>  #include <ctime>  struct SwapIter { int from; int to; };  double rd() { return double(rand()) / RAND\_MAX; }  std::vector<int> SetupPorbabilities(int k)  {  std::vector<int> probabilities(k);  for(int i = 0; i < k; i++)  probabilities[i] = (rd() \* 100);  return probabilities;  }  std::vector<SwapIter> SetupSwapTable(int n, int k)  {  std::vector<SwapIter> swap\_table(k);  for(int i = 0; i < k; i++)  swap\_table[i] = {1 + (rand() % (n - 2)), 1 + (rand() % (n - 2))};  return swap\_table;  }  std::vector<std::vector<int>> SetupEdges(int n)  {  std::vector<std::vector<int>> l(n);  for(int i = 0; i < n; i++)  {  l[i].reserve(n);  for(int j = 0; j < n; j++)  {  if(j < i)  {  l[i][j] = l[j][i];  continue;  }  else if(j == i)  {  l[i][j] = 0;  continue;  }  l[i][j] = 50 \* (rd() + 0.1);  }  }  return l;  }  int f(const std::vector<int>& path, const std::vector<std::vector<int>> &l)  {  int res = 0;  for(int i = 1; i < path.size(); i++)  res += l[path[i - 1]][path[i]];  return res;  }  int main()  {  system("clear");  srand(time(0));  const int n = 10;  const int k = 20;  const double y = 0.5;  double t = 100.0;  std::vector<std::vector<int>> l = SetupEdges(n);  std::vector<int> probabilities = SetupPorbabilities(k);  std::vector<SwapIter> swap\_table = SetupSwapTable(n, k);  std::vector<int> v(n); std::iota(v.begin(), v.end(), 0);  std::shuffle(v.begin(), v.end(), std::default\_random\_engine(time(0)));  v.push\_back(v[0]);  std::cout << "\nДлины граней:\n";  for(int i = 0; i < n; i++)  for(int j = i + 1; j < n; j++)  std::cout << i + 1 << " -> " << j + 1 << " = " << l[i][j] << "\n";  std::cout << "\nВероятности: {";  for(int prob : probabilities)  std::cout << " " << prob;  std::cout << " }\n";  std::cout << "\nЗамены:\n";  for(const auto& iter : swap\_table)  std::cout << iter.from + 1 << " <> " << iter.to + 1 << "\n";  int best\_s = f(v, l);  std::cout << "\nНачальный Маршрут: {";  for(auto i : v)  std::cout << " " << i;  std::cout << " } S = " << best\_s << "\n";  for(int i = 0; i < k; i++)  {  std::cout << "\nИтерация " << i + 1 << "\n\n";  std::cout << "T = " << t << "\n";    std::vector<int> u = v;  SwapIter swap\_iter = swap\_table[i];  std::cout << "Замена = " << swap\_iter.from + 1 << " <> " << swap\_iter.to + 1 << "\n";  std::swap(u[swap\_iter.from], u[swap\_iter.to]);  int s = f(u, l);  double ds = s - best\_s;  double p = 100.0 \* exp(-ds / t);  std::cout << "Вероятность P = " << p << " <> " << probabilities[i] << "\n";  std::cout << "Новый Маршрут: {";  for(auto idx : u)  std::cout << " " << idx;  std::cout << " } S = " << s << "\n";  if(ds < 0 || p > probabilities[i])  {  best\_s = s;  v = u;  if(ds < 0)  std::cout << "Маршрут принят т.к. dS = " << ds << " < 0.\n\n";  else  std::cout << "Маршрут принят т.к. P = " << p << " > " << probabilities[i] <<".\n\n";  }  else  std::cout << "Маршрут не принят.\n\n";  t \*= y;  }  std::cout << "\nЛучший маршрут {";  for(auto idx : v)  std::cout << " " << idx;  std::cout << " }, где S = " << best\_s << ".\n";  return 0;  } |

**Приложение В**

*Листинг В - Код метода оптимизации роя частиц*

|  |
| --- |
| from math import \*  #include <vector>  #include <iostream>  double f(double, double);  struct Vector2D  {  double x{};  double y{};  double calc() { return f(x, y); }  void operator=(const Vector2D& other)  {  x = other.x;  y = other.y;  }  friend std::ostream& operator<<(std::ostream& to, const Vector2D& from)  {  return to << "( " << from.x << " ; " << from.y << " )";  }  };  struct Velocity : Vector2D  {  };  struct Point : Vector2D  {  Vector2D local\_min{};  Velocity velocity{};  void operator+=(const Velocity& other)  {  x += other.x;  y += other.y;  }  }; Point\* global\_min;  double rn() { return double(rand()) / RAND\_MAX; }  double rn(double min, double max) {return min + (max - min) \* rn(); }  double f(double x, double y)  {  return pow(x \* x + y - 11, 2) \* pow(x + y \* y - 7, 2);  }  std::vector<Velocity> GenerateVelocity(int n)  {  std::vector<Velocity> velocity(n);  for(int i = 0; i < n; i++)  velocity[i] = { rn(-10, 10), rn(-10, 10) };  return velocity;  }  std::vector<Point> GeneratePoints(int n)  {  auto velocities = GenerateVelocity(n);  std::vector<Point> points(n);  for(int i = 0; i < n; i++)  {  auto& point = points[i];  point = { rn(-5, 5), rn(-5, 5) };    point.local\_min = point;  point.velocity = velocities[i];  }  return points;  }  Velocity CalcVelocity(double W, double C, Point& point, const Point\* best\_point)  {  double R1 = rn(), R2 = rn();  double x = W \* point.velocity.x + C \* R1 \* (point.local\_min.x - point.x) + C \* R2 \* (best\_point->x - point.x);  double y = W \* point.velocity.y + C \* R1 \* (point.local\_min.y - point.y) + C \* R2 \* (best\_point->y - point.y);  return {x, y};  }  int main()  {  srand(time(0));  const double W = 0.5;  const auto C = 2;  const int n = 10;  const int k = 1000;  auto points = GeneratePoints(n);  global\_min = &points[0];  for(int i = 0; i < k; i++)  {  std::cout << "Итерация: #" << i << std::endl;  for(auto& point : points)  {  point.velocity = CalcVelocity(W, C, point, global\_min);  point += point.velocity;  double fxy = f(point.x, point.y);  if(fxy <= point.local\_min.calc())  {  point.local\_min = point;  }  if(fxy <= global\_min->calc())  {  global\_min = &point;  }  std::cout << "\tЧастица: " << point << std::endl;  std::cout << "\tCкорость: " << point.velocity << std::endl;  std::cout << "\tЗначение функции: " << point.calc() << std::endl;  std::cout << "\tЛокальный минимум: " << point.local\_min << std::endl << std::endl;  }  std::cout << "Глобальный минимум: f" << \*global\_min << " = " << global\_min->calc() << std::endl << std::endl;;  }  return 0;  } |

**Приложение Г**

*Листинг Г - Код муравьиного алгоритма*

|  |
| --- |
| #include <iostream>  #include <vector>  #include <unistd.h>  #include <ctime>  #include <iomanip>  struct Vector2D  {  double x{};  double y{};  void operator=(const Vector2D& other)  {  x = other.x;  y = other.y;  }  friend std::ostream& operator<<(std::ostream& to, const Vector2D& from)  {  return to << "( " << from.x << " ; " << from.y << " )";  }  double distance(const Vector2D& other)  {  return sqrt(pow(other.x - x, 2) + pow(other.y - y, 2));  }  };  struct Point : Vector2D  {  };  double rn() { return double(rand()) / RAND\_MAX; }  double rn(const double& min, const double& max) {return min + (max - min) \* rn(); }  double rn(const int& min, const int& max) { return min + double(max - min) \* rn(); }  const int max\_ants = 10;  const int max\_cities = 15;  const int max\_distance = 100;  const double alpha = 2.5;  const double beta = 2.0;  const double rho = 0.5;  const double qval = 100.0;  const double default\_pheromone = 1.0 / max\_cities;  struct Ant  {  int city\_index;  bool seen\_cities[max\_cities];  std::vector<int> path;  double tour\_length;  double prev\_tour\_length;  void SelectNextCity();  bool Simulate();  static double Product(int, int);  friend std::ostream& operator<<(std::ostream& s, const Ant& ant)  {  s << ant.path[0];  for(int i = 1; i < ant.path.size(); i++)  s << " -> " << ant.path[i];  s << " = " << ant.tour\_length << "; dS = " << ant.tour\_length - ant.prev\_tour\_length;  return s;  }  };  Point cities[max\_cities];  Ant ants[max\_ants];  double distances[max\_cities][max\_cities];  double pheromones[max\_cities][max\_cities];  bool obstacles[max\_cities][max\_cities];  double Ant::Product(int from, int to)  {  return pow(pheromones[from][to], alpha) \* pow(1.0 / distances[from][to], beta);  }  void Ant::SelectNextCity()  {  double denom = 0.0;  for(int to = 0; to < max\_cities; to++)  if(!seen\_cities[to])  denom += Product(city\_index, to);  assert(denom != 0.0);  int to = 0;  while(1)  {  to = (to + 1) % max\_cities;  if(seen\_cities[to]) continue;  double p = Product(city\_index, to) / denom;  if(p > rn()) break;  }  tour\_length += distances[city\_index][to];  city\_index = to;  seen\_cities[to] = true;  path.push\_back(to);  }  bool Ant::Simulate()  {  if(path.size() == max\_cities + 1) return true;  SelectNextCity();  if(path.size() == max\_cities)  {  path.push\_back(path[0]);  tour\_length += distances[city\_index][path[0]];  return true;  }  return false;  }  void ResetAnts()  {  for(int i = 0; i < max\_ants; i++)  {  ants[i].prev\_tour\_length = ants[i].tour\_length;  ants[i].city\_index = i % max\_cities;  ants[i].tour\_length = 0.0;  for(int j = 0; j < max\_cities; j++)  ants[i].seen\_cities[j] = false;  ants[i].seen\_cities[ants[i].city\_index] = true;  ants[i].path.clear();  ants[i].path.push\_back(ants[i].city\_index);  }  }  void Init()  {  for(int i = 0; i < max\_cities; i++)  {  cities[i].x = rn(0, max\_distance);  cities[i].y = rn(0, max\_distance);  }  for(int i = 0; i < max\_cities; i++)  {  for(int j = i + 1; j < max\_cities; j++)  {  double dst = cities[i].distance(cities[j]);  distances[i][j] = dst;  distances[j][i] = dst;  pheromones[i][j] = default\_pheromone;  pheromones[j][i] = default\_pheromone;  }  }  // int from\_obstacle = rn(0, max\_cities);  // int to\_obstacle;  // do  // {  // to\_obstacle = rn(0, max\_cities);  // } while (to\_obstacle == from\_obstacle);    // obstacles[from\_obstacle][to\_obstacle] = true;  // obstacles[to\_obstacle][from\_obstacle] = true;  ResetAnts();  }  void UpdateTrails()  {  for(int i = 0; i < max\_cities; i++)  {  for(int j = i + 1; j < max\_cities; j++)  {  pheromones[i][j] \*= (1.0 - rho);  if(pheromones[i][j] <= 0.1)  pheromones[i][j] = default\_pheromone;  pheromones[j][i] = pheromones[i][j];  }  }  for(const auto& ant : ants)  {  for(int i = 0; i < ant.path.size() - 1; i++)  {  int from = ant.path[i];  int to = ant.path[i + 1];  pheromones[from][to] += (qval / ant.tour\_length);  pheromones[to][from] = pheromones[from][to];  }  }  for(int i = 0; i < max\_cities; i++)  for(int j = 0; j < max\_cities; j++)  pheromones[i][j] \*= rho;  }  int main()  {  system("clear");  //srand(time(0));  const int k = 10;  Init();  for(int i = 0; i < k; i++)  {  std::cout << "Итерация №" << i + 1 << std::endl;  while(std::count\_if(std::cbegin(ants), std::cend(ants), [](const Ant& ant) { return ant.path.size() == max\_cities + 1; }) < max\_ants)  for(auto& ant : ants)  if(ant.Simulate())  UpdateTrails();  for(const auto& ant : ants)  std::cout << ant << std::endl;    std::cout << "\n\n";  ResetAnts();  }  return 0;  } |

**Приложение Д**

*Листинг Д - Код метода оптимизации пчелиного алгоритма*

|  |
| --- |
| #include <iostream>  #include <vector>  #include <unistd.h>  #include <ctime>  #include <iomanip>  double f(double x, double y)  {  return pow(x \* x + y - 11, 2) + pow(x + y \* y - 7, 2);  }  struct Vector2D  {  double x;  double y;  void operator=(const Vector2D& other)  {  x = other.x;  y = other.y;  }  friend std::ostream& operator<<(std::ostream& to, const Vector2D& from)  {  return to << "( " << from.x << " ; " << from.y << " )";  }  double Distance(const Vector2D& other) const  {  return sqrt(pow(other.x - x, 2) + pow(other.y - y, 2));  }  };  struct Point : Vector2D  {  double Calculate() const { return f(x, y); };  friend std::ostream& operator<<(std::ostream& to, const Point& from)  {  return to << "f" << Vector2D(from) << " = " << from.Calculate();  }  };  double rn() { return double(rand()) / RAND\_MAX; }  double rn(const double& min, const double& max) {return min + (max - min) \* rn(); }  double rn(const int& min, const int& max) { return min + double(max - min) \* rn(); }  struct Bee : Point  {  using Vector2D::operator=;  Bee() {};  Bee(const Point& coords) : Point(coords) {};  std::vector<Bee> CreateScanners(int, double) const;  static std::vector<Bee> CreateColony(int, double, const Point& offset = {0, 0});  };  void sort(std::vector<Bee>& colony)  {  std::sort(colony.begin(), colony.end(), [](const Bee& a, const Bee& b)  {  return a.Calculate() < b.Calculate();  });  }  void merge(std::vector<Bee>& a, const std::vector<Bee>& b)  {  a.insert(a.end(), b.begin(), b.end());  }  // void merge\_intersections(std::vector<Bee>& colony, double r)  // {  // for(int i = 0; i < colony.size(); i++)  // {  // for(int j = i + 1; j < colony.size(); j++)  // {  // double distance = colony[i].Distance(colony[j]);  // if(distance <= r)  // {  // if(colony[i].Calculate() <= colony[j].Calculate())  // {  // }  // else  // {    // }  // }  // }  // }  // }  std::vector<Bee> Bee::CreateScanners(int n, double r) const  {  return CreateColony(n, r, {x, y});  }  std::vector<Bee> Bee::CreateColony(int n, double r, const Point& offset)  {  std::vector<Bee> colony(n);  for(int i = 0; i < n; i++)  colony[i] = {.x = offset.x + rn(-r, r), .y = offset.y + rn(-r, r)};  return colony;  }  int main()  {  srand(time(0));  system("clear");  const int S = 100;  const double R = 3;  constexpr int N = 10;  constexpr int M = 30;  const int K = 20;  auto colony = Bee::CreateColony(S, 50);  for(int i = 0; i < K; i++)  {  sort(colony);  std::cout << "Итерация №" << i << ":" << std::endl;  for(int j = 0; j < N; j++) std::cout << colony[j] << std::endl;  std::cout << std::endl;  std::vector<Bee> best(N);  std::vector<Bee> possible(M);  std::vector<Bee> new\_colony;  for(int j = 0; j < N; j++)  best[j] = colony[j];  for(int j = 0; j < M; j++)  possible[j] = colony[N + j + 1];  for(const auto& bee : best)  merge(new\_colony, bee.CreateScanners(N, R));  for(const auto& bee : possible)  merge(new\_colony, bee.CreateScanners(M, R));  colony = new\_colony;  }  return 0;  } |