

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
1 ОНТОЛОГИЯ.....	5
1.1 Постановка задачи	5
1.2 Теоретический раздел	5
1.3 Создание онтологии	5
1.4 Программная реализация	8
2 МЕТОД ИМИТАЦИИ ОТЖИГА	10
2.1 Постановка задачи	10
2.2 Теоретический раздел	10
2.3 Ручной просчет алгоритма.....	11
2.4 Программная реализация	15
3 МЕТОД ОПТИМИЗАЦИИ РОЯ ЧАСТИЦ	16
3.1 Постановка задачи	16
3.2 Теоретический раздел	16
3.3 Ручной просчет алгоритма.....	18
3.4 Программная реализация	20
4 МЕТОД ОПТИМИЗАЦИИ МУРАВЬИНОЙ КОЛОНИИ	22
4.1 Постановка задачи	22
4.2 Теоретический раздел	22
4.3 Ручной просчет алгоритма.....	24
4.4 Программная реализация	37
5 АЛГОРИТМ ПЧЕЛИНОЙ КОЛОНИИ	39
5.1 Постановка задачи	39

5.2 Теоретический раздел	39
5.3 Ручной просчет алгоритма.....	41
5.4 Программная реализация	44
ЗАКЛЮЧЕНИЕ.....	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	48
ПРИЛОЖЕНИЯ	50

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):

$$O = (X, R, F), \text{ где} \quad (1.1)$$

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.

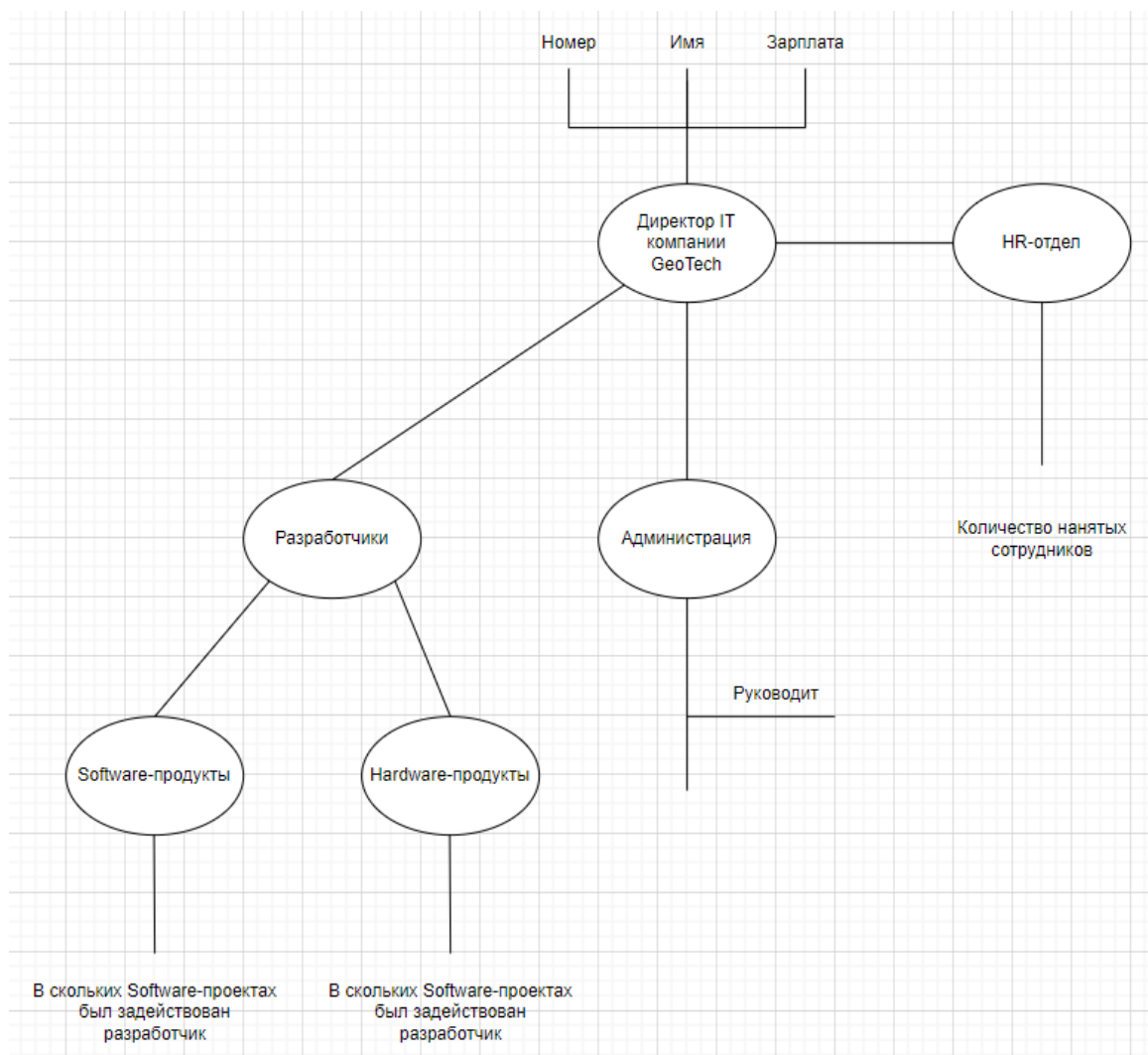


Рисунок 1.1 – Изображение, описывающее онтологию

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.

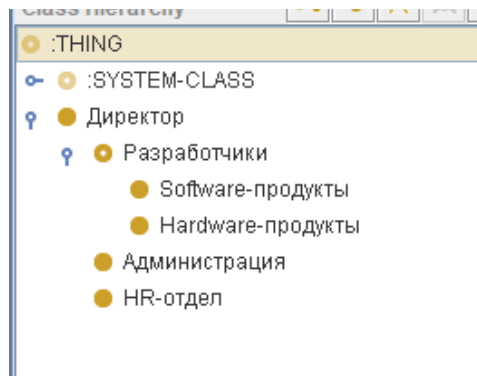


Рисунок 1.2 – Созданные классы

Созданные слоты представлены на рисунке 1.3.

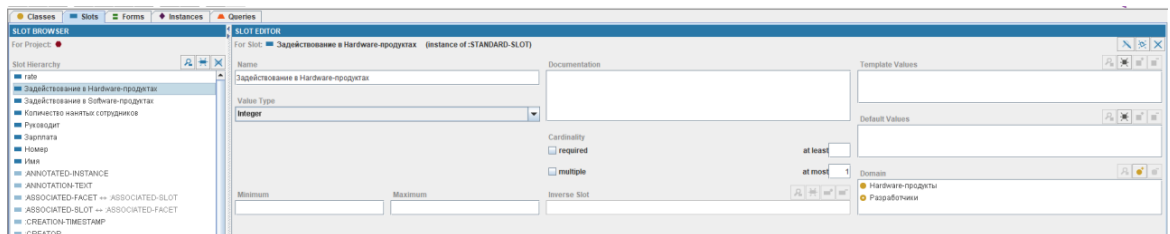


Рисунок 1.3 – Созданные классы

Созданные сущности представлены на рисунке 1.4

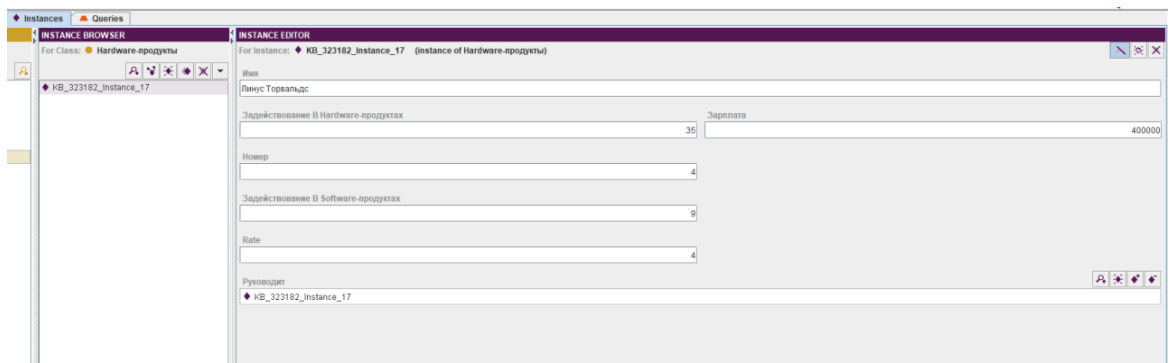


Рисунок 1.4 – Созданные классы

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.

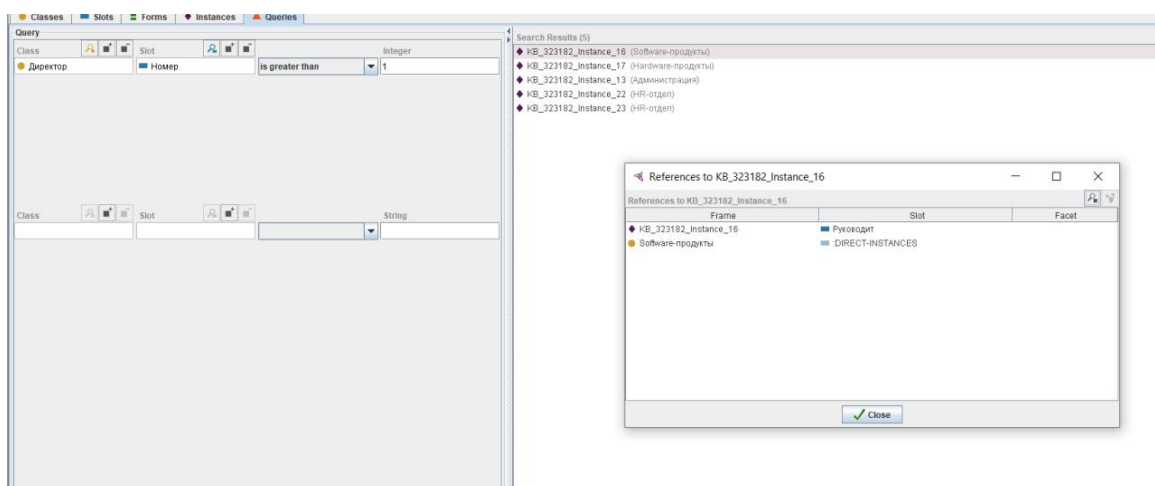


Рисунок 1.5 – Первый запрос

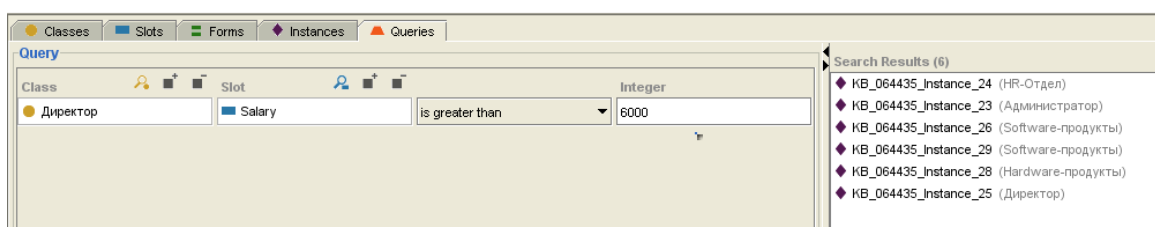


Рисунок 1.6 – Второй запрос

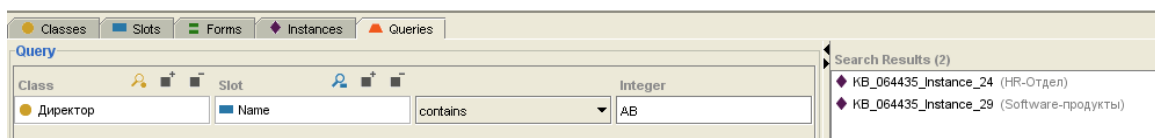
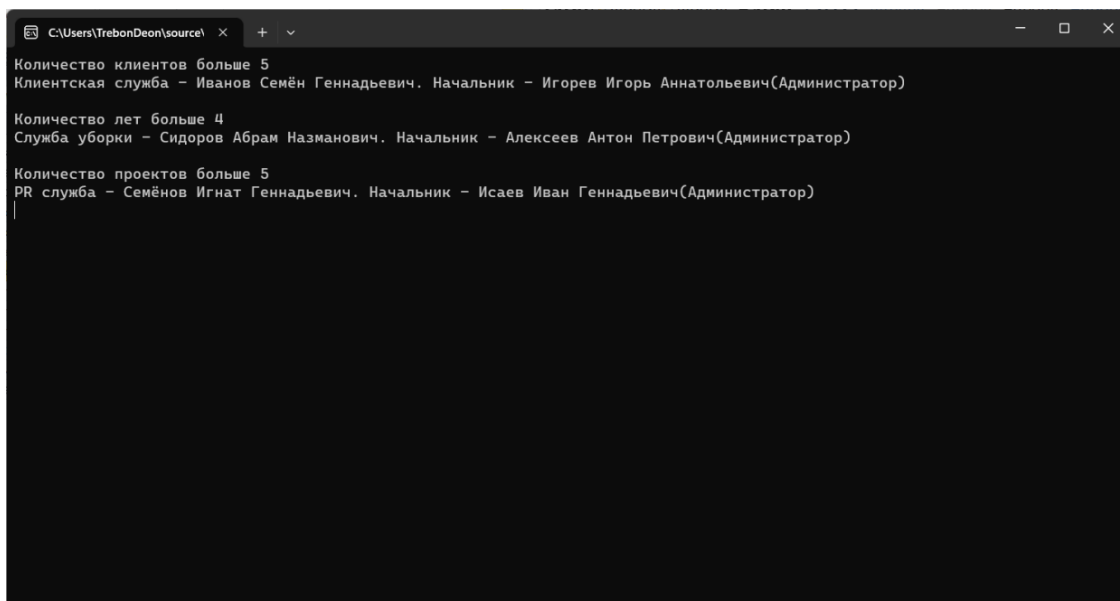


Рисунок 1.7 – Третий запрос

1.4 Программная реализация

Была реализована онтология на языке C#, используя принципы ООП.

Вывод работы программы представлен на рисунке 1.8.



The screenshot shows a Windows command prompt window with a dark background. The title bar at the top indicates the file path 'C:\Users\TrebonDeon\source\'. The window contains the following text output from a program:

```
Количество клиентов больше 5
Клиентская служба – Иванов Семён Геннадьевич. Начальник – Игорь Игорь Аннатович(Администратор)

Количество лет больше 4
Служба уборки – Сидоров Абрам Назманович. Начальник – Алексеев Антон Петрович(Администратор)

Количество проектов больше 5
PR служба – Семёнов Игнат Геннадьевич. Начальник – Исаев Иван Геннадьевич(Администратор)
|
```

Рисунок 1.8 – 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 $\mathcal{G}(x,T)$, 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 $h(\Delta E,T)$ переходит к следующему состоянию x' , в противном случае процесс генерации x' повторяется. Здесь ΔE обозначает приращение функции $f(x')-f(x)$. Величина $h(\Delta E,T)$ называется вероятностью принятия нового состояния.

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

$$h(\Delta E, T) = 1 / (1 + \exp(\Delta E / T)), \quad (2.1)$$

либо приближенное значение, представленное в формуле (2.2).

$$h(\Delta E, T) = \exp(-\Delta E / T). \quad (2.2)$$

Вторая формула используется наиболее часто. При ее использовании $h(\Delta E, T)$ оказывается больше единицы в случае $\Delta E < 0$, 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).

$$g(x, T) = \begin{cases} 1, & f(x') - f(x) < 0 \\ \exp\left(\frac{-\Delta E}{T}\right), & f(x') - f(x) \geq 0 \end{cases} \quad (2.3)$$

Конкретная схема метода отжига задается следующими параметрами:

- Выбором закона изменения температуры $T(k)$, где k – номер шага.
- Выбором порождающего семейства распределений $\mathcal{G}(x, T)$.
- Выбором функции вероятности принятия $h(\Delta E, T)$.

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:

$$AB = |AB| = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}, \quad (2.4)$$

- 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 ($T_{n+1} = T_n * 0.5$), 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 ($P = T_0 * e^{\frac{\Delta S}{T_n}}$). 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.

```
Итерация 16

Т = 0.00305176
Замена = 4 <> 5
Вероятность Р = 0 <> 91
Новый Маршрут: { 2 7 5 6 1 8 4 3 0 9 2 } S = 309
Маршрут не принят.

Итерация 17

Т = 0.00152588
Замена = 4 <> 8
Вероятность Р = inf <> 60
Новый Маршрут: { 2 7 5 3 6 8 4 1 0 9 2 } S = 262
Маршрут принят т.к. dS = -10 < 0.

Итерация 18

Т = 0.000762939
Замена = 2 <> 4
Вероятность Р = 100 <> 41
Новый Маршрут: { 2 3 5 7 6 8 4 1 0 9 2 } S = 262
Маршрут принят т.к. Р = 100 > 41.

Итерация 19

Т = 0.00038147
Замена = 9 <> 6
Вероятность Р = 0 <> 76
Новый Маршрут: { 2 3 5 7 6 0 4 1 8 9 2 } S = 336
Маршрут не принят.

Итерация 20

Т = 0.000190735
Замена = 9 <> 3
Вероятность Р = inf <> 31
Новый Маршрут: { 2 3 0 7 6 8 4 1 5 9 2 } S = 196
Маршрут принят т.к. dS = -66 < 0.

Лучший маршрут { 2 3 0 7 6 8 4 1 5 9 2 }, где S = 196.
○ (base) magomeddaurbekov@MacBook-Pro-Magomed-2 СППР %
```

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)

$$f(x, y) = (x^2 + y - 11)^2 + (y^2 + x - 7)^2 \quad (3.1)$$

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 u_i , которая отражает когнитивную компоненту. Аналогично значение глобального оптимума, достигнутого частицами роя, будем обозначать \hat{u} , 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 $x_i(t)$ в гиперпространстве и скоростью движения $v_i(t)$. Позиция частицы изменяется в соответствии с формулой (3.2):

$$x_i(t + 1) = x_i(t) + v_i(t + 1), \quad (3.2)$$

где $x_i(0) \sim (x_{min}, x_{max})$

Вектор скорости $v_i(t + 1)$ управляет процессом поиска решения и его компоненты определяются с учетом когнитивной и социальной составляющей по формуле (3.3):

$$v_{ij}(t + 1) = v_i(t) + c1r1j(t)[y_{ij}(t) - x_{ij}(t)] + c2r2j(t)[\hat{y}_j(t) - x_{ij}(t)] \quad (3.3)$$

Здесь $v_i(t)$ - j -ая компонента скорости ($j=1, \dots, nx$) i -ой частицы в момент времени t , $x_{ij}(t)$ - j -я координата позиции i -й частицы, $c1$ и $c2$ – положительные коэффициенты ускорения (часто полагаемые 2), регулирующие вклад когнитивной и социальной компонент, $r1j(t)$ и $r2j(t) \sim (0,1)$ - случайные числа из диапазона $[0,1]$, которые генерируются в соответствии с нормальным распределением и вносят элемент случайности в процесс поиска. Кроме этого $y_{ij}(t)$ - персональная лучшая позиция по j -й координате i -ой частицы, а $\hat{y}_j(t)$ – 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):

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_j(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_j(t)) \end{cases} \quad (3.4)$$

где $f: R^{n_\infty} \rightarrow R$ фитнес-функция. Как и в эволюционных алгоритмах, фитнес-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) $\widehat{y_j}(t)$ в момент t определяется в соответствии с формулой (3.5)

$$\widehat{y_j}(t) \in \{y_0(t), \dots, y_{n_s}(t)\} | f(y_i(t)) = \min \{y_0(t), \dots, y_{n_s}(t)\} \quad (3.5)$$

где n_s – 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), v_x (изменение скорости координаты x), v_y (изменение скорости координаты y):

- $x = [3.74, -2.25]$;
- $y = [1.75, -3.34]$;
- $v_x = [4.76, 7.56]$;
- $v_y = [2.34, -4.32]$.

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

$$\begin{aligned} & (3.74*3.74+1.75-11)*(3.74*3.74+1.75-11)+(3.74+1.75*1.75-7) \\ & \quad *(3.74+1.75*1.75-7)=22.48 \\ & (-2.25*(-2.25)+(-3.34)-11)*(-2.25*(-2.25)+(-3.34)-11)+(-2.25+(-3.34) \\ & \quad *(-3.34)-7)*(-2.25+(-3.34)*(-3.34)-7)=89.70 \end{aligned}$$

По формуле (3.6) рассчитывается изменение скорости:

$$(w * v + (a1 * r1 * (pbest - coord)) + (a2 * r2 * (gbest - coord))), \quad (3.6)$$

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.

```

Итерация: #999
Частица: ( 3 ; 2 )
Скорость: ( 2.37192e-07 ; -5.92981e-08 )
Значение функции: 1.0962e-40
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 7.24984e-07 ; -1.69522e-07 )
Значение функции: 9.19393e-28
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( -6.15745e-07 ; 1.53936e-07 )
Значение функции: 1.02373e-37
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 2.35041e-07 ; -5.87602e-08 )
Значение функции: 1.74768e-39
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 8.83864e-07 ; -2.20966e-07 )
Значение функции: 8.37125e-39
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( -1.89596e-07 ; 4.73991e-08 )
Значение функции: 1.02489e-37
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 5.25554e-07 ; -1.31389e-07 )
Значение функции: 0
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 1.15002e-06 ; -2.87505e-07 )
Значение функции: 2.67373e-39
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 1.33485e-06 ; -3.33712e-07 )
Значение функции: 1.12804e-40
Локальный минимум: ( 3 ; 2 )

Частица: ( 3 ; 2 )
Скорость: ( 1.88772e-06 ; -4.7193e-07 )
Значение функции: 1.27034e-39
Локальный минимум: ( 3 ; 2 )

Глобальный минимум: f( 3 ; 2 ) = 0

```

```

○ (base) magomeddaurbekov@MacBook-Pro-Magomed-2 СППР %

```

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 $n_g = |V|$ - число узлов в графе.

Обозначим L^k - 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 τ_{ij} . Строго говоря, в начальный момент времени концентрация феромона для каждой дуги графа нулевая, но мы для удобства каждой дуге присвоим небольшое случайное число $\tau_{ij}(0)$. Муравей выбирает следующую дугу пути случайным образом[1]. Множество муравьев $k=\{1, \dots, n_k\}$ 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 i -ой вершине, то он выбирает следующую вершину $j \in N_i^k$ на основе вероятностей перехода (4.1):

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t)}{\sum_{j \in N_i^k} \tau_{ij}^\alpha(t)}, & \text{если } j \in N_i^k \\ 0, & \text{если } j \notin N_i^k \end{cases} \quad (4.1)$$

Здесь N_i^k представляет множество возможных вершин, связанных с j -й вершиной, для i -го муравья. Если для любого i -го узла и k -го муравья $N_i^k = \emptyset$, тогда предшественник узла i включается в N_i^k . 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 α 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):

$$\Delta\tau_{ij}^k(t) = \frac{1}{L^k(t)} \quad (4.2)$$

Здесь $L^k(t)$ – длина пути, построенного k -м муравьем в момент времени t .

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

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t) \quad (4.3)$$

где n_k - 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 $x^k(t)$

обозначает решение в момент t , и некоторая функция $f(x^k(t))$ выражает качество решения.

Если $\Delta\tau^k$ не пропорционально качеству решения и все муравьи откладывают одинаковое количество феромона $\Delta\tau_{ij}^1 = \Delta\tau_{ij}^2 = \dots = \Delta\tau_{ij}^k$, 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.

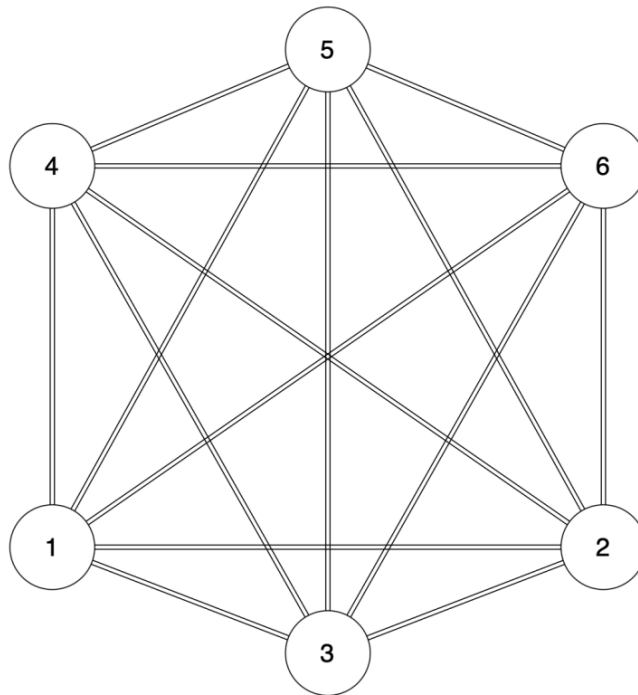


Рисунок 4.1 – Граф алгоритма

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:

$$0 \rightarrow 1: \frac{1^2}{1^2 + 1^2 + 1^2} = \frac{1}{3}$$

$$0 \rightarrow 5: \frac{1}{3}$$

$$0 \rightarrow 6: \frac{1}{3}$$

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:

$$6 \rightarrow 1: \frac{1^2}{1^2+1^2+1^2+1^2} = \frac{1}{4}$$

$$6 \rightarrow 2: \frac{1}{4}$$

$$6 \rightarrow 4: \frac{1}{4}$$

$$6 \rightarrow 5: \frac{1}{4}$$

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:

$$4 \rightarrow 5: \frac{1^2}{1^2+1^2+1^2} = \frac{1}{3}$$

$$4 \rightarrow 3: \frac{1}{3}$$

$$4 \rightarrow 7: \frac{1}{3}$$

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:

$$7 \rightarrow 2: \frac{1^2}{1^2+1^2} = \frac{1}{2}$$

$$7 \rightarrow 3: \frac{1}{2}$$

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 : $0 \rightarrow 6 \rightarrow 4 \rightarrow 7 \rightarrow 3$

Таким же образом считаются переходы для 2 муравья.

Считаются переходы для 2 муравья. Из текущей нулевой вершины:

$$0 \rightarrow 1: \frac{1^2}{1^2+1^2+1^2} = \frac{1}{3}$$

$$0 \rightarrow 5: \frac{1}{3}$$

$$0 \rightarrow 6: \frac{1}{3}$$

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:

$$5 \rightarrow 6: \frac{1^2}{1^2+1^2} = \frac{1}{2}$$

$$5 \rightarrow 4: \frac{1}{2}$$

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:

$$4 \rightarrow 6: \frac{1^2}{1^2+1^2+1^2} = \frac{1}{3}$$

$$4 \rightarrow 3: \frac{1}{3}$$

$$4 \rightarrow 7: \frac{1}{3}$$

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:

$$7 \rightarrow 2: \frac{1^2}{1^2 + 1^2} = \frac{1}{2}$$

$$7 \rightarrow 3: \frac{1}{2}$$

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 : $0 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$

Все муравьи добрались до конечной точки. Необходимо выветрить феромон на всех дугах графа: $(1 - 0.6) * 1 = 0.4$. 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 этом 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:

$$\text{Для дуг 1 муравья: } 0.4 + \frac{1}{3+11+5+3} = 0.4 + \frac{1}{22} \approx 0.44$$

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]].

$$\text{Для дуги 0->5 2 муравья: } 0.4 + \frac{1}{16+28+5+3} = 0.4 + \frac{1}{52} \approx 0.42$$

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]].

$$\text{Для дуги 5->4 2 муравья: } 0.4 + \frac{1}{16+28+5+3} = 0.4 + \frac{1}{52} \approx 0.42$$

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->2 муравья: $0.44 + \frac{1}{16+28+5+3} = 0.44 + \frac{1}{52} \approx 0.46$

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:

$$0 \rightarrow 1: \frac{0.4^2}{0.4^2 + 0.42^2 + 0.44^2} \approx 0.30$$

$$0 \rightarrow 5: \frac{0.42^2}{0.4^2 + 0.42^2 + 0.44^2} \approx 0.33$$

$$0 \rightarrow 6: \frac{0.44^2}{0.4^2 + 0.42^2 + 0.44^2} \approx 0.36$$

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:

$$6 \rightarrow 1: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.44^2 + 0.4^2} \approx 0.24$$

$$6 \rightarrow 2: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.44^2 + 0.4^2} \approx 0.24$$

$$6 \rightarrow 4: \frac{0.44^2}{0.4^2 + 0.4^2 + 0.44^2 + 0.4^2} \approx 0.29$$

$$6 \rightarrow 5: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.44^2 + 0.4^2} \approx 0.24$$

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:

$$4 \rightarrow 5: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.44^2} \approx 0.31$$

$$4 \rightarrow 3: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.44^2} \approx 0.31$$

$$4 \rightarrow 7: \frac{0.44^2}{0.4^2 + 0.4^2 + 0.44^2} \approx 0.38$$

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:

$$7 \rightarrow 2: \frac{0.4^2}{0.4^2 + 0.46^2} \approx 0.43$$

$$7 \rightarrow 3: \frac{0.46^2}{0.4^2 + 0.46^2} \approx 0.57$$

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:

$$0 \rightarrow 1: \frac{0.4^2}{0.4^2 + 0.42^2 + 0.44^2} \approx 0.30$$

$$0 \rightarrow 5: \frac{0.42^2}{0.4^2 + 0.42^2 + 0.44^2} \approx 0.33$$

$$0 \rightarrow 6: \frac{0.44^2}{0.4^2 + 0.42^2 + 0.44^2} \approx 0.36$$

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:

$$5 \rightarrow 6: \frac{0.4^2}{0.42^2 + 0.4^2} \approx 0.48$$

$$5 \rightarrow 4: \frac{0.42^2}{0.42^2 + 0.4^2} \approx 0.52$$

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:

$$4 \rightarrow 3: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.46^2} \approx 0.30$$

$$4 \rightarrow 6: \frac{0.4^2}{0.4^2 + 0.4^2 + 0.46^2} \approx 0.30$$

$$4 \rightarrow 7: \frac{0.46^2}{0.4^2 + 0.4^2 + 0.46^2} \approx 0.40$$

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.

$$7 \rightarrow 2: \frac{0.4^2}{0.4^2 + 0.46^2} \approx 0.43$$

$$7 \rightarrow 3: \frac{0.46^2}{0.4^2 + 0.46^2} \approx 0.57$$

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: $(1 - 0.6) * 0.4 = 0.16$.

Для дуг равных 0.42: $(1 - 0.6) * 0.42 = 0.168$.

Для дуг равных 0.44: $(1 - 0.6) * 0.44 = 0.176$.

Для дуг равных 0.46: $(1 - 0.6) * 0.46 = 0.184$.

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 этом часть феромона на той дуге, по которой он прошел.

Для дуги 0->6 1 муравья: $0.176 + \frac{1}{3+11+5+3} = 0.176 + \frac{1}{22} \approx 0.221$

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: $0.176 + \frac{1}{3+11+5+3} = 0.176 + \frac{1}{22} \approx 0.221$

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: $0.184 + \frac{1}{3+11+5+3} = 0.184 + \frac{1}{22} \approx 0.230$

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: $0.184 + \frac{1}{3+11+5+3} = 0.184 + \frac{1}{22} \approx 0.230$

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: $0.168 + \frac{1}{16+28+5+3} = 0.4 + \frac{1}{52} \approx 0.187$

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: $0.168 + \frac{1}{16+28+5+3} = 0.168 + \frac{1}{52} \approx 0.187$

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: $0.230 + \frac{1}{16+28+5+3} = 0.230 + \frac{1}{52} \approx 0.25$

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: $0.230 + \frac{1}{16+28+5+3} = 0.230 + \frac{1}{52} \approx 0.25$

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.

```
Итерация №1
0 -> 2 -> 9 -> 5 -> 8 -> 1 -> 12 -> 10 -> 14 -> 3 -> 13 -> 7 -> 11 -> 4 -> 6 -> 0 = 589.662; dS = 589.662
1 -> 12 -> 11 -> 4 -> 7 -> 14 -> 10 -> 5 -> 3 -> 13 -> 0 -> 8 -> 9 -> 2 -> 6 -> 1 = 493.895; dS = 493.895
2 -> 8 -> 6 -> 5 -> 14 -> 10 -> 1 -> 12 -> 4 -> 11 -> 13 -> 3 -> 7 -> 0 -> 9 -> 2 = 409.901; dS = 409.901
3 -> 13 -> 7 -> 14 -> 10 -> 5 -> 2 -> 12 -> 1 -> 4 -> 11 -> 0 -> 9 -> 8 -> 6 -> 3 = 508.047; dS = 508.047
4 -> 6 -> 8 -> 2 -> 1 -> 12 -> 3 -> 13 -> 5 -> 9 -> 7 -> 14 -> 10 -> 11 -> 0 -> 4 = 673.294; dS = 673.294
5 -> 2 -> 3 -> 13 -> 7 -> 6 -> 1 -> 12 -> 10 -> 14 -> 11 -> 4 -> 8 -> 9 -> 0 -> 5 = 565.641; dS = 565.641
6 -> 8 -> 2 -> 12 -> 1 -> 10 -> 14 -> 11 -> 4 -> 13 -> 3 -> 7 -> 5 -> 0 -> 9 -> 6 = 433.226; dS = 433.226
7 -> 3 -> 4 -> 11 -> 10 -> 14 -> 12 -> 1 -> 8 -> 9 -> 2 -> 5 -> 13 -> 6 -> 0 -> 7 = 563.66; dS = 563.66
8 -> 6 -> 1 -> 5 -> 7 -> 13 -> 3 -> 14 -> 10 -> 11 -> 4 -> 12 -> 2 -> 9 -> 0 -> 8 = 441.841; dS = 441.841
9 -> 2 -> 8 -> 11 -> 4 -> 10 -> 14 -> 5 -> 3 -> 7 -> 13 -> 6 -> 1 -> 12 -> 0 -> 9 = 552.51; dS = 552.51

Итерация №2
0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 3 -> 13 -> 7 -> 0 = 354.437; dS = -235.225
1 -> 12 -> 2 -> 8 -> 6 -> 0 -> 9 -> 5 -> 14 -> 10 -> 11 -> 4 -> 3 -> 13 -> 7 -> 1 = 480.216; dS = -13.6792
2 -> 9 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 14 -> 10 -> 5 -> 7 -> 3 -> 13 -> 0 -> 2 = 381.777; dS = -28.1246
3 -> 13 -> 7 -> 5 -> 10 -> 14 -> 11 -> 4 -> 6 -> 8 -> 2 -> 9 -> 0 -> 12 -> 1 -> 3 = 479.332; dS = -28.7157
4 -> 3 -> 13 -> 7 -> 0 -> 9 -> 2 -> 5 -> 10 -> 14 -> 11 -> 12 -> 1 -> 6 -> 8 -> 4 = 478.69; dS = -194.604
5 -> 14 -> 10 -> 11 -> 4 -> 3 -> 13 -> 7 -> 0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 5 = 387.623; dS = -178.019
6 -> 8 -> 2 -> 9 -> 0 -> 7 -> 3 -> 13 -> 5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 = 351.236; dS = -81.9902
7 -> 3 -> 13 -> 0 -> 4 -> 11 -> 10 -> 14 -> 5 -> 2 -> 9 -> 8 -> 6 -> 1 -> 12 -> 7 = 471.186; dS = -92.4738
8 -> 6 -> 1 -> 12 -> 2 -> 9 -> 0 -> 7 -> 3 -> 13 -> 5 -> 10 -> 14 -> 11 -> 4 -> 8 = 416.099; dS = -25.7423
9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 3 -> 13 -> 0 -> 9 = 365.017; dS = -187.493

Итерация №3
0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 3 -> 13 -> 0 = 365.017; dS = 10.5806
1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 3 -> 13 -> 0 -> 9 -> 2 -> 8 -> 6 -> 1 = 365.017; dS = -115.198
2 -> 9 -> 0 -> 7 -> 3 -> 13 -> 5 -> 10 -> 14 -> 11 -> 4 -> 12 -> 1 -> 6 -> 8 -> 2 = 348.77; dS = -33.0068
3 -> 13 -> 7 -> 0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 3 = 354.437; dS = -124.895
4 -> 11 -> 14 -> 10 -> 5 -> 7 -> 3 -> 13 -> 0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 = 362.551; dS = -116.139
5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 -> 8 -> 2 -> 9 -> 0 -> 7 -> 13 -> 3 -> 5 = 354.437; dS = -33.186
6 -> 8 -> 2 -> 9 -> 0 -> 7 -> 3 -> 13 -> 5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 = 351.236; dS = 0
7 -> 3 -> 13 -> 5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 -> 8 -> 2 -> 9 -> 0 -> 7 = 351.236; dS = -119.95
8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 13 -> 3 -> 9 -> 2 -> 0 -> 8 = 435.519; dS = 19.4206
9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 13 -> 3 -> 7 -> 0 -> 9 = 351.236; dS = -13.7811

Итерация №4
0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 3 -> 13 -> 0 = 365.017; dS = 0
1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 3 -> 13 -> 0 -> 9 -> 2 -> 8 -> 6 -> 1 = 365.017; dS = 0
2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 13 -> 3 -> 7 -> 0 -> 9 -> 2 = 351.236; dS = 2.46616
3 -> 13 -> 7 -> 5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 -> 8 -> 2 -> 9 -> 0 -> 3 = 364.985; dS = 10.5483
4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 3 -> 13 -> 0 -> 9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 = 365.017; dS = 2.46616
5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 -> 8 -> 2 -> 9 -> 0 -> 7 -> 3 -> 13 -> 5 = 351.236; dS = -3.20054
6 -> 8 -> 2 -> 9 -> 0 -> 7 -> 3 -> 13 -> 5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 = 351.236; dS = 0
7 -> 3 -> 13 -> 5 -> 14 -> 10 -> 11 -> 4 -> 12 -> 1 -> 6 -> 8 -> 2 -> 9 -> 0 -> 7 = 351.236; dS = 0
8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 13 -> 3 -> 7 -> 0 -> 9 -> 2 -> 8 = 351.236; dS = -84.283
9 -> 2 -> 8 -> 6 -> 1 -> 12 -> 4 -> 11 -> 10 -> 14 -> 5 -> 7 -> 13 -> 3 -> 0 -> 9 = 364.985; dS = 13.7489
```

Рисунок 4.2 – Вывод работы программы

На рисунке 4.2 показаны первые 4 итерации из 10, в которых виден процесс поиска муравьиной колонией 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)

$$f(x, y) = (x^2 + y - 11)^2 + (y^2 + x - 7)^2 \quad (5.1)$$

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 $X_{\beta,0} \in D$, отправляются пчелы-разведчики, где β – номер пчелы разведчика, $\beta \in [1:S]$, 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) d_{β} .

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 d_β является подобластью локального поиска, представляющая собой гиперкуб в пространстве R^k с центром в точке $X_{\beta,0}$. Длина его сторон равна 2Δ , где Δ – параметр, называемый размером области локального поиска.

3 шаг: Сравнивается евклидово расстояние $\|X_{\beta,0} - X_{\gamma,0}\|$ между двумя агентами-разведчиками. Для точек $A = (x_1, x_2, \dots, x_n)$ и $B = (y_1, y_2, \dots, y_n)$ евклидово расстояние считается по формуле (5.2)

$$d(A, B) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2} \quad (5.2)$$

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

- поставить в соответствие этим агентам два различных пересекающихся участка d_β, d_γ (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 посылаются 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 τ раз, где τ — 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.

```

Итерация №1:
f( 5.86542 ; -1.82954 ) = 470.318
f( 5.89196 ; 0.0884022 ) = 567.819
f( -3.36282 ; 7.15453 ) = 1722.34
f( 7.92568 ; 4.88295 ) = 3828.31
f( 8.4057 ; 2.60458 ) = 3943.42
f( 8.68435 ; -0.123561 ) = 4136.66
f( -0.826281 ; 8.70059 ) = 4609.49
f( 8.21274 ; 5.46267 ) = 4797.38
f( -2.24032 ; 8.86642 ) = 4820.94
f( 9.07456 ; 0.0630772 ) = 5103.8

Итерация №2:
f( 3.24954 ; 1.00599 ) = 7.81887
f( 3.7937 ; 2.6923 ) = 53.3596
f( 4.3415 ; -0.409762 ) = 61.5397
f( 4.54877 ; -2.79646 ) = 76.3649
f( 5.02608 ; -2.70178 ) = 161.989
f( 4.93876 ; -0.209452 ) = 177.833
f( 5.57882 ; -0.69944 ) = 378.154
f( 5.45717 ; 0.721776 ) = 381.393
f( 5.57784 ; 0.732293 ) = 435.281
f( 6.01538 ; -1.49385 ) = 562.816

Итерация №3:
f( 3.6185 ; -1.83907 ) = 0.0647651
f( 3.49997 ; -1.95275 ) = 0.592226
f( 3.44751 ; -1.70398 ) = 1.09134
f( 3.43178 ; -2.00665 ) = 1.72184
f( 3.70881 ; -2.10817 ) = 1.74853
f( 3.13855 ; 1.62011 ) = 1.7509
f( 3.28316 ; 2.04923 ) = 3.5757
f( 3.30101 ; 1.994 ) = 3.65125
f( 3.12074 ; 2.34735 ) = 3.83983
f( 3.33276 ; 1.61584 ) = 4.08483

Итерация №4:
f( 2.90011 ; 2.16197 ) = 0.512391
f( 3.67097 ; -1.97005 ) = 0.560795
f( 3.68025 ; -1.97622 ) = 0.665711
f( 3.0679 ; 2.13242 ) = 0.674729
f( 3.01611 ; 1.7132 ) = 1.13615
f( 3.77343 ; -2.01285 ) = 2.18343
f( 3.67524 ; -2.19775 ) = 2.36195
f( 2.74195 ; 1.93454 ) = 2.65961
f( 2.79552 ; 2.38091 ) = 2.79067
f( 3.35612 ; -1.63355 ) = 2.82824

```

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-samy-e-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 A.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> SetupProbabilities(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 = SetupProbabilities(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 << "\tСкорость: " << 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;
}

```