



**МИНОБРНАУКИ РОССИИ**  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**"МИРЭА - Российский технологический университет"**  
**РТУ МИРЭА**

---

**Институт Информационных Технологий**  
**Кафедра Вычислительной Техники**

**ПРАКТИЧЕСКАЯ РАБОТА №5**

**по дисциплине**  
**«Системный анализ данных в системах поддержки принятия**  
**решений»**  
**Алгоритм пчелиной колонии**

Студент группы: ИКБО-04-22

Кликушин В.И.  
(Ф. И.О. студента)

Преподаватель

Железняк Л.М.  
(Ф.И.О. преподавателя)

Москва 2024

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 ПЧЕЛИНЫЙ АЛГОРИТМ .....	4
1.1 Описание алгоритма .....	4
1.2 Постановка задачи.....	5
1.3 Математическая модель .....	5
1.4 Ручной расчёт .....	6
1.5 Программная реализация .....	11
ЗАКЛЮЧЕНИЕ .....	12
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ .....	13
ПРИЛОЖЕНИЯ.....	14

# ВВЕДЕНИЕ

Алгоритм пчелиной колонии был разработан в 2005 году турецким ученым Дервишем Карабога. Источником вдохновения для создания алгоритма стало поведение реальных пчелиных колоний при поиске и сборе нектара. Пчелы, благодаря сложным коммуникациям и распределению ролей внутри колонии, эффективно находят и оценивают источники пищи, демонстрируя пример коллективного интеллекта. Этот процесс, перенесенный в математическую модель, лег в основу алгоритма, позволяющего решать сложные задачи оптимизации.

Особенность алгоритма заключается в разделении пчел на группы разведчиков, работников и наблюдателей, каждая из которых выполняет свою роль в поиске оптимального решения. Такой механизм обеспечивает баланс между глобальным исследованием пространства решений и локальным уточнением, что делает алгоритм гибким и эффективным даже для задач с большим количеством переменных.

Алгоритм пчелиной колонии уже доказал свою эффективность в широком спектре задач: от оптимизации маршрутов и распределения ресурсов до анализа данных и проектирования систем управления. В энергетике он используется для оптимизации энергопотребления и размещения генераторов, в логистике — для построения маршрутов доставки, а в биоинформатике — для анализа геномных данных. Кроме того, его успешно применяют в машинном обучении для настройки гиперпараметров моделей и кластеризации данных.

Сегодня, в эпоху роста сложности задач и увеличения объемов данных, алгоритм пчелиной колонии остается актуальным. Его способность адаптироваться к различным типам задач, использовать преимущества коллективного поведения и эффективно находить решения делает его мощным инструментом для исследователей и инженеров.

# 1 ПЧЕЛИНЫЙ АЛГОРИТМ

## 1.1 Описание алгоритма

В алгоритме участвуют искусственные агенты, которые исследуют пространство решений, оценивают их качество и концентрируются на наиболее перспективных областях, улучшая решения итерационно.

Процесс оптимизации начинается с инициализации множества случайных точек, представляющих начальные решения. Затем выполняются итерации, каждая из которых состоит из нескольких фаз: исследование новых решений разведчиками, поиск локальных улучшений собирателями и сосредоточение усилий на наиболее перспективных направлениях. Если в течение заданного числа итераций улучшений не наблюдается, алгоритм завершает работу.

Обобщая выше сказанное, алгоритм состоит из следующих ключевых шагов:

1. Инициализация начальных параметров: количества пчел-разведчиков, максимального расстояния для объединения точек, размера локальной области поиска и других.
2. Случайное размещение пчел-разведчиков в пространстве решений (Формула 1.3.1).
3. Оценка значений целевой функции для каждой точки.
4. Выделение областей, основанных на расстоянии между точками, и определение точек с наибольшим значением функции в каждой области (Формула 1.3.3).
5. Локальный поиск новых решений в выделенных областях с учётом текущего оптимального значения.
6. Сравнение новых решений с текущими лучшими. При обнаружении лучшего значения обновляется глобальное решение.

7. Проверка условия останова. Если улучшений не наблюдается в течение заданного числа итераций или достигнуто максимальное число итераций, алгоритм завершает работу; иначе возвращается к шагу четыре.

## 1.2 Постановка задачи

Цель работы: реализовать преобразование Коши методом пчелиной колонии для нахождения приближённого глобального минимума функции.

Задачи: изучить пчелиный алгоритм, выбрать тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт итерации алгоритма, разработать программную реализацию пчелиного алгоритма для задачи минимизации функции.

Нахождение глобального минимума функции от многих переменных состоит в поиске точки в многомерном пространстве, где значение функции будет минимальным. Сложность этой задачи состоит в том, что функция может содержать множество локальных минимумов, где производная функция равна нулю, но значение функции не является минимальным.

Выбранная функция для оптимизации: функция Гольдшейна-Прайса (Формула 1.2.1).

$$f(x, y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]. \quad (1.2.1)$$

Глобальный минимум функции достигается в точке (0; -1) и равен 3. Функция рассматривается на области  $-2 \leq x, y \leq 2$ .

## 1.3 Математическая модель

Количество пчел-разведчиков равно  $S$ . Случайным образом генерируется  $S$  точек, куда отправляются пчелы (Формула 1.3.1).

$$X_{N,K} \in D, \quad (1.3.1)$$

где  $N$  – номер пчелы-разведчика,  $N \in [1:S]$ ;

$K$  – номер итерации;

$D$  – область поиска.

Каждая точка представлена своими координатами (Формула 1.3.2).

$$X = (x_1, x_2, \dots, x_n), \quad (1.3.2)$$

где  $n$  – номер координаты.

Подобласти, в которые объединяются точки формируются на основе Евклидова расстояния между точками. Для точек  $A = (x_1, x_2, \dots, x_n)$  и  $B = (y_1, y_2, \dots, y_n)$  евклидово расстояние считается по Формуле 1.3.3.

$$d(A, B) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2} \quad (1.3.3)$$

где  $d$  – рассчитанное расстояние.

## 1.4 Ручной расчёт

Количество пчел-разведчиков:  $S = 10$ . Область локального поиска:  $l = 0,5$ .

Допустимое Евклидово расстояние для объединения точек:  $d = 1,5$ .

Случайным образом размещено  $S$  точек в области поиска  $D$ . Координаты точек представлены ниже:

$$X_{1,0} = (1,082, 1,309);$$

$$X_{2,0} = (1,382, -1,130);$$

$$X_{3,0} = (-0,410, -1,284);$$

$$\begin{aligned}
X_{4,0} &= (0,751, 0,690); \\
X_{5,0} &= (0,007, 1,862); \\
X_{6,0} &= (0,006, 1,522); \\
X_{7,0} &= (-1,153, 0,236); \\
X_{8,0} &= (-0,922, -0,675); \\
X_{9,0} &= (0,023, -0,661); \\
X_{10,0} &= (1,067, 1,370).
\end{aligned}$$

Рассчитанные значения целевой функции во всех точках представлены ниже:

$$\begin{aligned}
F(X_{1,0}) &= 6454,259; \\
F(X_{2,0}) &= 32906,002; \\
F(X_{3,0}) &= 78,617; \\
F(X_{4,0}) &= 1216,079; \\
F(X_{5,0}) &= 176789,654; \\
F(X_{6,0}) &= 97662,703; \\
F(X_{7,0}) &= 1088,499; \\
F(X_{8,0}) &= 564,689; \\
F(X_{9,0}) &= 102,551; \\
F(X_{10,0}) &= 8889,103.
\end{aligned}$$

Наименьшее значение целевая функция имеет в точке  $F(X_{3,0})$ . Глобальный минимум функции обновляется и становится равным 78,617. Точка, в которой достигается глобальный минимум на текущей итерации:  $(-0,410, -1,284)$ .

Далее рассчитывается евклидово расстояние между точками для объединения их в области по Формуле 1.3.3:

$$d(X_{1,0}, X_{2,0}) = \sqrt{(1,082 - 1,382)^2 + (1,309 + 1,130)^2} = 2,458 > 1,5$$

Расстояние больше допустимого, поэтому точки не объединяются в одну область.

$$d(X_{1,0}, X_{3,0}) = \sqrt{(1,082 + 0,410)^2 + (1,309 + 1,284)^2} = 2,994 > 1,5$$

$$d(X_{1,0}, X_{4,0}) = \sqrt{(1,082 - 0,751)^2 + (1,309 - 0,690)^2} = 0,702 < 1,5$$

Получено расстояние, меньшее допустимого. Первая и четвёртая точки объединяются в одну область. На текущей итерации четвёртая точка больше не может быть размещена в другую область.

$$d(X_{1,0}, X_{5,0}) = \sqrt{(1,082 - 0,007)^2 + (1,309 - 1,862)^2} = 1,209 < 1,5$$

$$d(X_{1,0}, X_{6,0}) = \sqrt{(1,082 - 0,006)^2 + (1,309 - 1,522)^2} = 1,096 < 1,5$$

$$d(X_{1,0}, X_{7,0}) = \sqrt{(1,082 + 1,153)^2 + (1,309 - 0,236)^2} = 2,480 > 1,5$$

$$d(X_{1,0}, X_{8,0}) = \sqrt{(1,082 + 0,922)^2 + (1,309 + 0,675)^2} = 2,821 > 1,5$$

$$d(X_{1,0}, X_{9,0}) = \sqrt{(1,082 - 0,023)^2 + (1,309 + 0,661)^2} = 2,237 > 1,5$$

$$d(X_{1,0}, X_{10,0}) = \sqrt{(1,082 - 1,067)^2 + (1,309 - 1,370)^2} = 0,062 < 1,5$$

В первую область попадают точки  $(X_{1,0}, X_{4,0}, X_{5,0}, X_{6,0}, X_{10,0})$ .

Точки, распределённые в область, не участвуют в дальнейшем переборе.

Переходим к созданию второй области. Евклидово расстояние для следующих комбинаций точек рассчитано ниже:

$$d(X_{2,0}, X_{3,0}) = \sqrt{(1,382 + 0,410)^2 + (-1,130 + 1,284)^2} = 1,799 > 1,5$$

$$d(X_{2,0}, X_{7,0}) = \sqrt{(1,382 + 1,153)^2 + (-1,130 - 0,236)^2} = 2,880 > 1,5$$

$$d(X_{2,0}, X_{8,0}) = \sqrt{(1,382 + 0,922)^2 + (-1,130 + 0,675)^2} = 2,349 > 1,5$$



$$d(X_{2,0}, X_{9,0}) = \sqrt{(1,382 - 0,023)^2 + (-1,130 + 0,661)^2} = 1,437 < 1,5$$

Во вторую область попадают точки  $(X_{2,0}, X_{9,0})$ . Дальнейший расчёт:

$$d(X_{3,0}, X_{7,0}) = \sqrt{(-0,410 + 1,153)^2 + (-1,284 - 0,236)^2} = 1,692 > 1,5$$

$$d(X_{3,0}, X_{8,0}) = \sqrt{(-0,410 + 0,922)^2 + (-1,284 + 0,675)^2} = 0,796 < 1,5$$

В третью область попадают точки  $(X_{3,0}, X_{8,0})$ . Остаётся единственная точка, не распределённая в область  $X_{7,0}$ . Эта точка представляет четвёртую область и является единственной точкой в своей области.

В каждой области выбрана точка, в которой функция имеет наименьшее значение.

Для первой области центральная точка с минимальным значением целевой функции  $X_{4,0}$ . Определена область поиска в первой области:  $0,251 < x < 1,251$ ;  $0,190 < y < 1,190$ .

Генерируется S-1 точек в подобласти поиска, которые обозначаются в соответствии с Формулой 1.4.1.

$$X_{N,K}^M \in D, \quad (1.4.1)$$

где N – номер пчелы-разведчика,  $N \in [1: S - 1]$ ;

K – номер итерации;

M – номер подобласти;

D – подобласть поиска.

Сгенерированные точки представлены ниже (первая точка – центра подобласти):

$$X_{1,0}^1 = (0,751, 0,690);$$

$$X_{2,0}^1 = (0,714, 1,161);$$

$$X_{3,0}^1 = (0,713, 1,013);$$

$$X_{4,0}^1 = (1,216, 0,739);$$

$$X_{5,0}^1 = (0,931, 0,365);$$

$$X_{6,0}^1 = (0,861, 1,037);$$

$$X_{7,0}^1 = (0,759, 0,749);$$

$$X_{8,0}^1 = (0,863, 1,117);$$

$$X_{9,0}^1 = (1,238, 0,996);$$

$$X_{10,0}^1 = (1,070, 0,636).$$

Посчитаны значения целевой функции в выбранных точках:

$$F(X_{1,0}^1) = 1216,079;$$

$$F(X_{2,0}^1) = 8434,740;$$

$$F(X_{3,0}^1) = 4775,814;$$

$$F(X_{4,0}^1) = 860,193;$$

$$F(X_{5,0}^1) = 1103,519;$$

$$F(X_{6,0}^1) = 3347,560;$$

$$F(X_{7,0}^1) = 1423,953;$$

$$F(X_{8,0}^1) = 4680,991;$$

$$F(X_{9,0}^1) = 1084,610;$$

$$F(X_{10,0}^1) = 897,479.$$

Новая точка для первой подобласти с минимальным значением функции  
 $X_{4,0}^1 = 860,193$ .

Аналогичным образом выбирается лучшая точка в каждой выделенной подобласти.

Лучшее значение функции: 78,61; Лучшая точка:  $X = (-0,410, -1,284)$ .

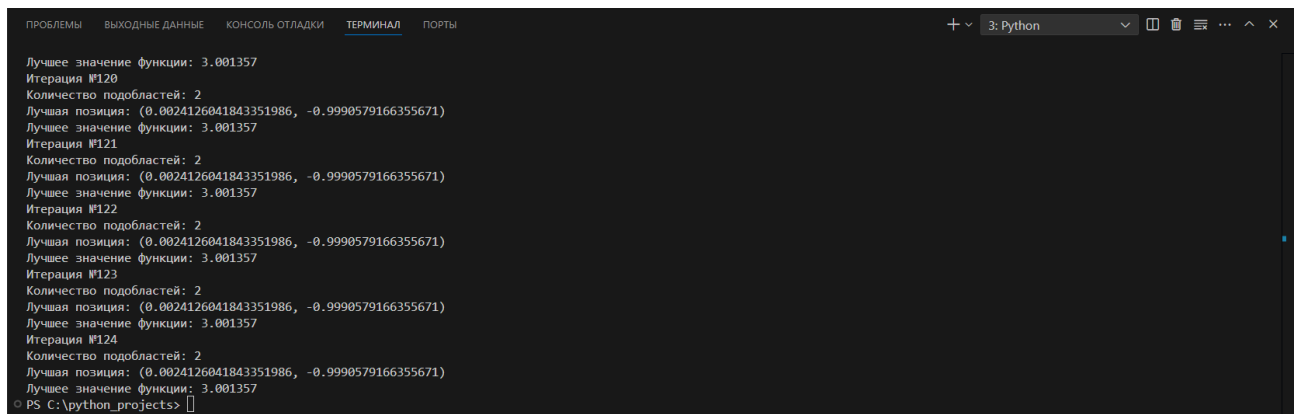
## 1.5 Программная реализация

Разработан класс BeeColony, представляющий колонию пчёл.

Код алгоритма пчелиной колонии для задачи поиска глобального минимума функции представлен в Приложении А.

Количество пчел  $S = 100$ , количество итераций без улучшения до остановки алгоритма  $k = 100$ , максимальное евклидово расстояние  $d = 1,5$ , область локального поиска  $l = 0,5$ .

Результат работы алгоритма пчелиной колонии представлен на Рисунке 1.5.1.



```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ
Лучшее значение функции: 3.001357
Итерация №120
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №121
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №122
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №123
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №124
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
PS C:\python_projects>
```

Рисунок 1.5.1 – Результат работы алгоритма пчелиной колонии для задачи поиска глобального минимума функции

Визуализация процесса поиска представлена на Рисунке 1.5.2.

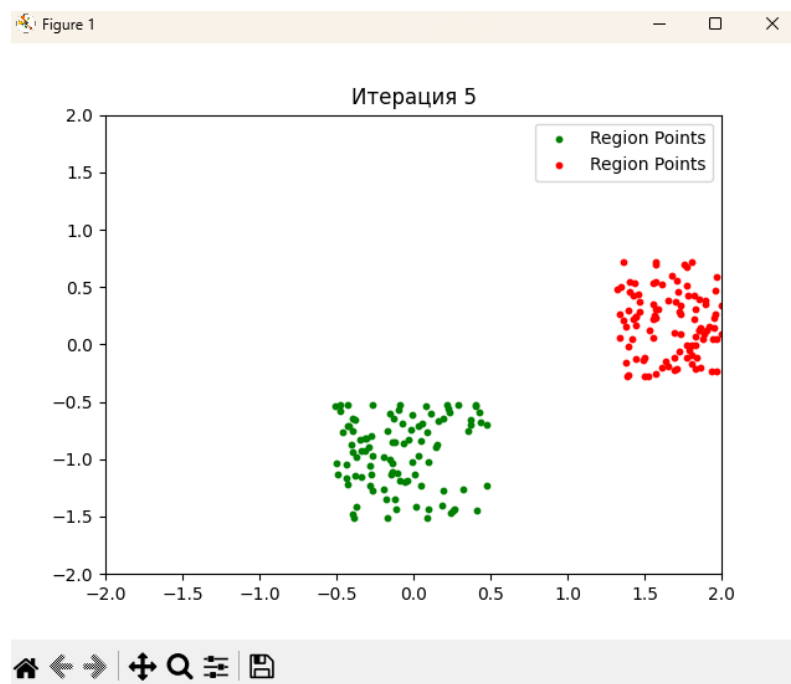


Рисунок 1.5.2 – Визуализация работы алгоритма пчелиной колонии

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы изучен алгоритм пчелиной колонии, проведён его ручной расчёт одной итерации, а также разработана программа на языке Python для оптимизации функции Голдштейна-Прайса.

Основное преимущество алгоритма пчелиной колонии заключается в его способности эффективно находить решения сложных многомерных задач оптимизации благодаря сочетанию глобального и локального поиска. Алгоритм объединяет разведку новых областей решения с их углублённым исследованием, что позволяет адаптироваться к сложным ландшафтам целевой функции и избегать застревания в локальных экстремумах.

Разработанная программа демонстрирует, как параметры алгоритма, такие как количество разведчиков, радиус локального поиска, порог объединения областей и число итераций, существенно влияют на его эффективность и точность. Визуализация процесса оптимизации показала динамику перемещения точек и их группировку в наиболее перспективных областях, что подтверждает способность алгоритма успешно балансировать между исследованием и использованием текущей информации.

Таким образом, алгоритм пчелиной колонии является универсальным инструментом для решения задач оптимизации, требующих поиска глобального экстремума в сложных многомерных пространствах.

## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс] : Учебно-методическое пособие / А. Б. Сорокин – Москва: Московский технологический университет (МИРЭА), 2019.
2. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
3. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
4. Алгоритм пчелиной колонии. [Электронный ресурс]: Википедия. – URL: [https://ru.wikipedia.org/wiki/Алгоритм\\_пчелиной\\_колонии](https://ru.wikipedia.org/wiki/Алгоритм_пчелиной_колонии) (Дата обращения: 25.11.2024).
5. Бахтигозин Т. Э., Пивоваров М. Д., Сафарьян О. А. Анализ алгоритма пчелиной колонии и его применение в криптографии // Молодой исследователь Дона, 2022. URL: <https://cyberleninka.ru/article/n/analiz-algoritma-pchelinoy-kolonii-i-ego-primenenie-v-kriptografii> (Дата обращения: 26.11.2024).

## **ПРИЛОЖЕНИЯ**

Приложение А — Код реализации алгоритма пчелиной колонии.

## Приложение А

### Код реализации алгоритма пчелиной колонии

#### *Листинг А – Реализация алгоритма пчелиной колонии*

```
import random
import math
import itertools
from typing import List, Tuple
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

class Point:
    def __init__(self, *args):
        self.coordinates = args

    def euclidean_distance(self, other_point):
        '''Вычисляет евклидово расстояние между текущей точкой и другой
        точкой.'''
        if not isinstance(other_point, __class__):
            raise ValueError('Евклидово расстояние может быть рассчитано только
            между экземплярами Point.')
        return math.sqrt(sum((x - y) ** 2 for x, y in zip(self, other_point)))

    def __repr__(self):
        return f"Point{self.coordinates}"

    def __str__(self):
        return str(self.coordinates)

    def __getitem__(self, index):
        return self.coordinates[index]

class BeeColony:
    def __init__(self, fitness_function, bounds: List[Tuple[float, float]],
        scouts: int, k: int,
        distance_threshold: float, l: float, max_iterations: int,
        maximize: bool = False):
        '''
        Инициализирует алгоритм пчелиной колонии для оптимизации.
        Параметры:
        fitness_function (Callable[..., float]): Целевая функция для
        оптимизации.
        bounds (List[Tuple[float, float]]): Ограничения для каждой
        координаты в виде [(min1, max1), ..., (minD, maxD)].
        scouts (int): Количество пчел-разведчиков.
        k (int): Количество итераций без улучшения для остановки алгоритма.
        distance_threshold (float): Максимальное евклидово расстояние для
        объединения точек.
        l (float): Размер области локального поиска.
        max_iterations (int): Максимальное количество итераций алгоритма.
        maximize (bool): Определяет, ищется максимум (True) или минимум
        функции (False).
        '''
        self.fitness_function = fitness_function
        self.bounds = bounds
        self.scouts = scouts
        self.k = k
        self.distance_threshold = distance_threshold
        self.l = l
        self.max_iterations = max_iterations
```

### Продолжение Листинга А

```
self.maximize = maximize
self.history = list()

def compare(self, a, b):
    '''Сравнение значений функции с учетом типа оптимизации.'''
    return a > b if self.maximize else a < b

def optimize(self) -> Tuple[Point, float]:
    '''
    Выполняет оптимизацию с использованием алгоритма пчелиной колонии.
    Возвращает:
        Tuple[Point, float]: Лучшая найденная точка и значение целевой
    функции в ней.
    '''
    scouts = [
        Point(*[random.uniform(bounds[0], bounds[1]) for bounds in
self.bounds])
        for _ in range(self.scouts)
    ]
    best_global_value = float("-inf") if self.maximize else float("inf")
    best_global_point = None
    num_iteration = 0
    stagnation_count = 0

    while num_iteration < self.max_iterations:
        print(f"Итерация №{num_iteration}")
        num_iteration += 1
        iteration_data = []

        values = [self.fitness_function(*point) for point in scouts]
        best_local_index = max(range(len(values)), key=lambda i: values[i]
\
        if self.maximize else min(range(len(values)), key=lambda i:
values[i]))
        best_local_point = scouts[best_local_index]
        best_local_value = values[best_local_index]
        if self.compare(best_local_value, best_global_value):
            best_global_value = best_local_value
            best_global_point = best_local_point
            stagnation_count = 0
        else:
            stagnation_count += 1

        combined_regions = []
        used_indices = set()
        for i, point_i in enumerate(scouts):
            if i in used_indices:
                continue
            region = [point_i]
            for j, point_j in enumerate(scouts):
                if j != i and j not in used_indices and \
                    point_i.euclidean_distance(point_j) <=
self.distance_threshold:
                region.append(point_j)
                used_indices.add(j)
            used_indices.add(i)
            combined_regions.append(region)
        print(f"Количество подобластей: {len(combined_regions)}")

        new_scouts = []
        for region in combined_regions:
```



### Продолжение Листинга А

```
        center = max(region, key=lambda p: self.fitness_function(*p)) \
            if self.maximize else min(region, key=lambda p:
self.fitness_function(*p))
        search_area = [
            (max(self.bounds[i][0], center[i] - self.l),
min(self.bounds[i][1], center[i] + self.l))
            for i in range(len(self.bounds))
        ]
        local_scouts = [center] + [
            Point(*[random.uniform(area[0], area[1]) for area in
search_area])
            for _ in range(self.scouts - 1)
        ]
        iteration_data.append(local_scouts)
        local_values = [self.fitness_function(*point) for point in
local_scouts]
        best_local_index = max(range(len(local_values)), key=lambda i:
local_values[i]) \
            if self.maximize else min(range(len(local_values)),
key=lambda i: local_values[i])
        new_scouts.append(local_scouts[best_local_index])

        self.history.append(iteration_data)
        scouts = new_scouts
        if stagnation_count >= self.k:
            break
        print(f"Лучшая позиция: {best_global_point}")
        print(f"Лучшее значение функции: {best_global_value:.6f}")
        return best_global_point, best_global_value

def visualize(self):
    fig, ax = plt.subplots()
    x_min, x_max = self.bounds[0]
    y_min, y_max = self.bounds[1]
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    colors = itertools.cycle(['b', 'g', 'r', 'c', 'm', 'y', 'k'])

    def update(frame):
        ax.clear()
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.set_title(f"Итерация {frame + 1}")
        iteration_data = self.history[frame]
        for region_points in iteration_data:
            x_coords = [p[0] for p in region_points]
            y_coords = [p[1] for p in region_points]
            color = next(colors)
            ax.scatter(x_coords, y_coords, c=color, s=10, label="Region
Points")
        ax.legend(loc='upper right')
        anim = FuncAnimation(fig, update, frames=len(self.history), blit=False,
interval=500, repeat=False)
        plt.show()

def goldstein_price(x: float, y: float) -> float:
    '''Функция Голдштейна-Прайса для оптимизации.'''
    term1 = (1 + (x + y + 1) ** 2 * (19 - 14 * x + 3 * x ** 2 - 14 * y + 6 * x *
y + 3 * y ** 2))
    term2 = (30 + (2 * x - 3 * y) ** 2 * (18 - 32 * x + 12 * x ** 2 + 48 * y -
36 * x * y + 27 * y ** 2))
```

### *Окончание Листинга А*

```
        return term1 * term2

def sphere(x, y):
    '''Функция сферы для оптимизации.'''
    return x ** 2 + y ** 2

colony = BeeColony(
    fitness_function=goldstein_price,
    bounds=[(-2, 2), (-2, 2)],
    scouts=100,
    k=100,
    distance_threshold=1.5,
    l=0.5,
    max_iterations=1000,
    maximize=False
)

best_position, best_value = colony.optimize()
colony.visualize()
print("Лучшая позиция:", best_position)
print(f"Лучшее значение функции: {best_value:.6f}")
```