МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники

Практическая работа №5
«Алгоритм пчелиной колонии»

по дисциплине
«Системный анализ данных СППР»

Студент группы: ИКБО-04-22 Егоров Л.А.
(Ф.И.О. студента)

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

Москва 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Алгоритм пчелиной колонии — это эвристический метод оптимизации, разработанный Марко Дориго и Дино Д’Агостино в 2005 году. Этот алгоритм вдохновлен поведением медоносных пчел, которые демонстрируют удивительную способность находить наилучшие источники нектара для сбора меда.

Основной целью работы пчелиной колонии в природе является разведка пространства вокруг улья с целью поиска нектара с последующим его сбором. Для этого в составе колонии существуют различные типы пчел: пчелы-разведчики и рабочие пчелы-фуражиры (кроме них, в колонии существуют трутни и матка, не участвующие в процессе сбора нектара). Разведчики ведут исследование окружающего улей пространства и сообщают информацию о перспективных местах, в которых было обнаружено наибольшее количество нектара (для обмена информацией в улье существует специальный механизм, именуемый танцем пчелы).

Алгоритм пчелиной колонии моделирует это поведение. Вместо реальных пчел и танцев, алгоритм использует «искусственных пчел» и «искусственные танцы». Искусственные пчелы перемещаются по пространству поиска, представленному в виде графа или сетки, и оценивают качество каждой позиции. Затем они возвращаются в «улей» и передают информацию о найденных позициях другим пчелам. Вероятность выбора пчелой определенной позиции зависит от ее качества и количества информации, полученной от других пчел. Со временем, пчелы концентрируют свои усилия на наиболее перспективных позициях.

Алгоритм пчелиной колонии широко используется для решения различных задач оптимизации, таких как:

- задача календарного планирования;
- задача коммивояжёра;
- транспортная задача.

1 АЛГОРИТМ РОЯ ЧАСТИЦ

1.1 Описание алгоритма

Сначала происходит инициализация начальных параметров и пчёл – генерация точек в области поиска (количество точек задано и равно S), а также свободных параметров алгоритма. Каждая точка имеет координаты (1.1.1).

$$X_j = \left(x_{1j}, x_{2j}, \ldots, x_{nj}\right) \ , \tag{1.1.1}$$

где $j \in [1; S]$ — номер частицы;

n — размерность векторов в задаче.

Формирование подобластей происходит на основе Евклидова расстояния между пчёлами (1.1.2).

$$d = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2} \tag{1.1.2}$$

Сначала выбирается точка с наименьшим значением функции – она становится центром новой подобласти. Вокруг неё собираются все пчёлы, расстояние до которых от центральной пчелы меньше заданного числа ε. После проверки всех пчёл убираются те пчёлы, которые вошли в подобласть, и данные действия повторяются для оставшихся пчёл.

После формирования подобластей начинается поиск оптимального значения в каждой из них. В каждой области выбирается точка с наилучшим значение функции, вокруг неё в квадрате со стороной 2Δ генерируются случайным образом $S-1$ пчёл, а затем среди сгенерированных пчёл и центральной пчелы выбирается та, которая имеет наименьшее значение функции. Теперь эта точка становится центром новой области, и процесс повторяется до тех пор, пока не наилучшая точка не останется статичной в течение заданного числа итераций.

Такой поиск проводится в каждой из полученных подобластей, и точкой останова алгоритма является окончание поиска в последней области.

Точкой останова алгоритма является выполнение заданного числа итераций.

1.2 Постановка задачи

Цель работы: реализовать глобальный алгоритм роя частиц для нахождения оптимального значения функции.

Поставлены следующие задачи:

- изучить алгоритм пчелиной колонии;
- выбрать тестовую функцию для оптимизации (нахождение глобального минимума);
- произвести ручной расчёт одной итерации алгоритма;
- разработать программную реализацию алгоритма пчелиной колонии для задачи минимизации функции.

Выбранная функция для оптимизации: функция Растригина (1.2.1). Она примечательна тем, что имеет большое количество локальных минимумов. Глобальный минимум функции достигается в точке $(0;0)$ и равен 0, при этом, в остальных локальных минимумах значение функции больше нуля. Функция рассматривается на области $x_i \in [-5.12, 5.12]$.

$$f(x,y) = 20 + x^2 - 10\cos(2\pi x) + y^2 - 10\cos(2\pi y) \qquad (1.2.1)$$

1.3 Ручной расчёт алгоритма

Выбранная функция: функция Растригина от двух переменных. Её формула представлена Формулой 1.2.1. На Рисунке 1.3.1 представлен график этой функции.

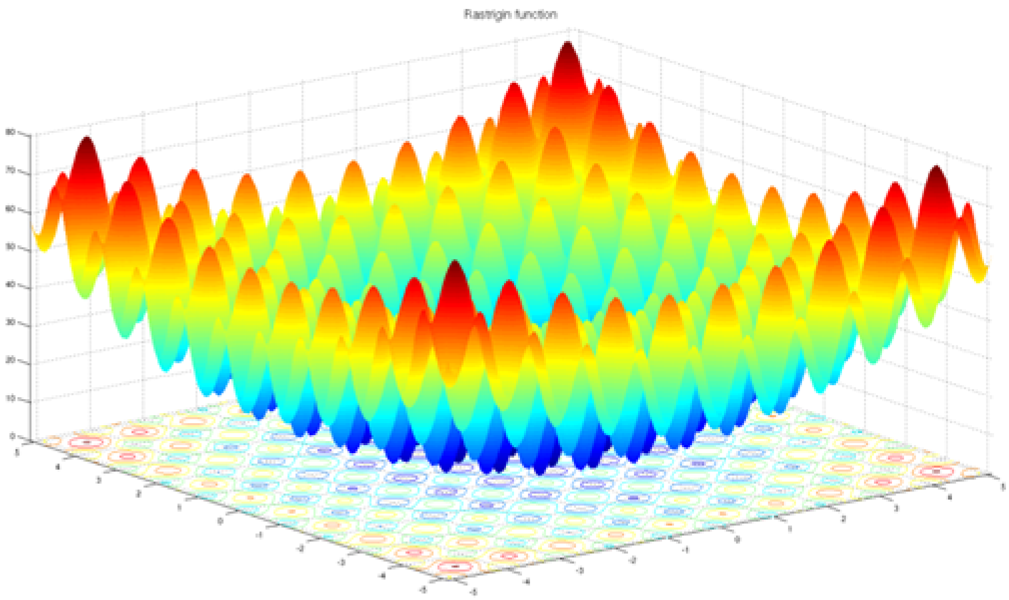

Рисунок 1.3.1 — График функции Растригина

Инициализированы свободные параметры алгоритма:

- $\varepsilon = 2$;
- $\Delta = 1$;
- количество пчёл-разведчиков (S): 8.

Создано 8 пчёл со следующими координатами:

$$X_1 = (-4.078, -5.091); f(X_0) = 45.310$$
$$X_2 = (4.723, -2.923); f(X_1) = 43.654$$
$$X_3 = (3.593, 3.706); f(X_2) = 57.682$$
$$X_4 = (-0.960, 3.632); f(X_3) = 31.198$$
$$X_5 = (3.945, -2.769); f(X_4) = 32.614$$
$$X_6 = (1.065, 4.941); f(X_5) = 27.056$$
$$X_7 = (4.918, 4.094); f(X_6) = 43.920$$
$$X_8 = (-1.068, 1.670); f(X_7) = 19.633$$

Среди оставшихся точек лучшее значение имеется у пчелы X_8: значение функции у неё равно 19.633. Далее рассчитывается Евклидово расстояние между точкой X_8 и оставшимися точками по Формуле 1.2.1.

$$d_{81} = \sqrt{(-1.068 + 4.078)^2 + (1.670 + 5.091)^2} = 7.401 \geq 2$$
$$d_{82} = \sqrt{(-1.068 - 4.723)^2 + (1.670 + 2.923)^2} = 7.392 \geq 2$$
$$d_{83} = \sqrt{(-1.068 - 3.593)^2 + (1.670 - 3.706)^2} = 5.087 \geq 2$$
$$d_{84} = \sqrt{(-1.068 + 0.960)^2 + (1.670 - 3.632)^2} = 1.964 < 2$$
$$d_{85} = \sqrt{(-1.068 - 3.945)^2 + (1.670 + 2.769)^2} = 6.696 \geq 2$$
$$d_{86} = \sqrt{(-1.068 - 1.065)^2 + (1.670 - 4.941)^2} = 3.904 \geq 2$$
$$d_{87} = \sqrt{(-1.068 - 4.918)^2 + (1.670 - 4.094)^2} = 6.458 \geq 2$$

Следовательно, в область точки X_8 вошла точка X_4.

Среди оставшихся точек лучшее значение имеется у пчелы X_6: значение функции у неё равно 27.056.

Далее рассчитывается Евклидово расстояние между точкой X_6 и оставшимися точками по Формуле 1.2.1.

$$d_{61} = \sqrt{(1.065 + 4.078)^2 + (4.941 + 5.091)^2} = 11.273 \geq 2$$
$$d_{62} = \sqrt{(1.065 - 4.723)^2 + (4.941 + 2.923)^2} = 8.673 \geq 2$$
$$d_{63} = \sqrt{(1.065 - 3.593)^2 + (4.941 - 3.706)^2} = 2.813 \geq 2$$
$$d_{65} = \sqrt{(1.065 - 3.945)^2 + (4.941 + 2.769)^2} = 8.230 \geq 2$$
$$d_{67} = \sqrt{(1.065 - 4.918)^2 + (4.941 - 4.094)^2} = 3.945 \geq 2$$

Следовательно, точка X_6 образует область сама с собой.

Среди оставшихся точек лучшее значение имеется у пчелы X_5: значение функции у неё равно 32.614.

Далее рассчитывается Евклидово расстояние между точкой X_5 и оставшимися точками по Формуле 1.2.1.

$$d_{51} = \sqrt{(3.945 + 4.078)^2 + (-2.769 + 5.091)^2} = 8.352 \geq 2$$
$$d_{52} = \sqrt{(3.945 - 4.723)^2 + (-2.769 + 2.923)^2} = 0.794 < 2$$
$$d_{53} = \sqrt{(3.945 - 3.593)^2 + (-2.769 - 3.706)^2} = 6.485 \geq 2$$
$$d_{57} = \sqrt{(3.945 - 4.918)^2 + (-2.769 - 4.094)^2} = 6.932 \geq 2$$

Следовательно, в область точки X_5 вошла точка X_2.

Среди оставшихся точек лучшее значение имеется у пчелы X_7: значение функции у неё равно 43.920.

Далее рассчитывается Евклидово расстояние между точкой X_7 и оставшимися точками по Формуле 1.2.1.

$$d_{71} = \sqrt{(4.918 + 4.078)^2 + (4.094 + 5.091)^2} = 12.856 \geq 2$$
$$d_{73} = \sqrt{(4.918 - 3.593)^2 + (4.094 - 3.706)^2} = 1.380 < 2$$

Следовательно, в область точки X_7 вошла точка X_3.

Среди оставшихся точек лучшее значение имеется у пчелы X_1: значение функции у неё равно 45.310.

Поскольку точек больше не осталось, то точка X_1 образует область сама с собой.

Рассмотрим поиск в первой подобласти. Лучшая точка: $(-1.068, 1.670)$ со значением функции 19.633. Новые сгенерированные точки имеют следующие координаты (точка X_8 является текущим центром области):

$$X_1 = (-0.822, 0.917); f(X_0) = 8.453$$
$$X_2 = (-1.212, 2.105); f(X_1) = 15.630$$
$$X_3 = (-0.196, 1.449); f(X_2) = 28.284$$
$$X_4 = (-0.845, 0.943); f(X_3) = 6.618$$
$$X_5 = (-0.775, 0.699); f(X_4) = 22.633$$
$$X_6 = (-0.826, 2.319); f(X_5) = 25.707$$
$$X_7 = (-1.464, 2.197); f(X_6) = 33.421$$

$$X_8 = (-1.068, 1.670); f(X_7) = 19.633$$

Минимальное значение среди достигнуто точкой X_4 (значение функции равно 6.618). Следовательно, эта точка становится центром области, и происходит переход к новой итерации.

1.4 Программная реализация

Для реализации расчётов алгоритма пчелиной колонии написан программный код на языке Python.

В программной реализации зафиксированы следующие параметры:

- количество пчёл: 100;
- количество итераций: 30;
- $\varepsilon = 0.8$;
- $\Delta = 1$.

Код реализации алгоритма пчелиной колонии для нахождения оптимального значения функции представлен в Листинге А.1.

На Рисунке 1.4.1 представлен результат выполнения программы для нахождения оптимального значения функции – консольный вывод результатов поиска в нескольких областях.

```
User@Huawei MINGW64 /d/grander-materials/САД/Практики/prac5 (main)
$ python new_bee_algorithm.py
Количество областей: 47
Найденное значение в 1-й области: 0.006 в точке (0.002, -0.005)
Найденное значение в 2-й области: 0.005 в точке (-0.004, 0.002)
Найденное значение в 3-й области: 0.024 в точке (-0.009, -0.005)
Найденное значение в 4-й области: 0.003 в точке (0.002, 0.003)
Найденное значение в 5-й области: 0.185 в точке (0.007, 0.03)
Найденное значение в 6-й области: 0.015 в точке (0.009, 0.002)
Найденное значение в 7-й области: 0.072 в точке (-0.014, 0.013)
Найденное значение в 8-й области: 0.004 в точке (-0.002, -0.004)
Найденное значение в 9-й области: 0.001 в точке (0.002, 0.002)
Найденное значение в 10-й области: 0.007 в точке (0.003, 0.005)
Найденное значение в 11-й области: 0.096 в точке (-0.012, -0.018)
Найденное значение в 12-й области: 0.018 в точке (-0.006, 0.007)
```

Рисунок 1.4.1 — Результаты поиска в первых 12 областях

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы выполнены поставленные задачи — изучен алгоритм пчелиной колонии, произведён его ручной расчёт для решения задачи поиска глобального минимума функции, а также разработана программа на языке Python для нахождения глобального минимума функции Растригина от двух переменных.

В заключение можно отметить, что алгоритм пчелиной колонии является мощным инструментом для решения задач оптимизации (в том числе, задач нахождения глобального минимума функции), в которых стандартные методы недостаточно эффективны из-за наличия множества локальных минимумов. Алгоритм имеет высокую сходимость, однако его результативность сильно зависит от настройки большого количества свободных параметров.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпенко, А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко — 3-е изд. — Москва: Издательство МГТУ им. Н. Э. Баумана, 2021. — 446 с.

2. Пряжников, В. Алгоритм имитации отжига [Электронный ресурс]. URL: https://pryazhnikov.com/notes/simulated-annealing/ (Дата обращения: 12.11.2024).

3. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие / А. Б. Сорокин — М.: Московский технологический университет (МИРЭА), 2019.

4. Rastrigin function [Электронный ресурс]: Википедия. URL: https://en.wikipedia.org/wiki/Rastrigin_function (Дата обращения: 01.11.2024).

5. Wang, Q., Zeng, J., Song, W. A New Electromagnetism-like Algorithm with Chaos Optimization 2010. С. 535–538.

ПРИЛОЖЕНИЯ

Приложение А — Реализация алгоритма пчелиной колонии на языке Python.

Приложение А

Реализация алгоритма пчелиной колонии на языке Python

Листинг А.1 — Реализация алгоритма пчелиной колонии

```
import numpy as np
import matplotlib.pyplot as plt


def rastrigin(x: np.ndarray):
    return 10 * len(x) + np.sum(x**2 - 10 * np.cos(2 * np.pi * x))


class BeeColony:

    def __init__(self,
                 scout_bee_count: int = 10,
                 optimal_bee_count: int = 5,
                 suboptimal_bee_count: int = 2,
                 optimal_solution_count: int = 2,
                 suboptimal_solution_count: int = 3,
                 field_size: float = 1.5):
        self.scout_bee_count = scout_bee_count
        self.optimal_bee_count = optimal_bee_count
        self.suboptimal_bee_count = suboptimal_bee_count
        self.optimal_solution_count = optimal_solution_count
        self.suboptimal_solution_count = suboptimal_solution_count
        self.field_size = field_size

        self._max = 5.12
        self._min = -self._max
        self.n = 2

    def solution_step(self, previous_result: np.ndarray | None = None):
        if previous_result is None:
            bee_area = np.random.random(size=(self.scout_bee_count, self.n)
                                        ) * (self._max - self._min) + self._min
        else:
            bee_area = previous_result
        function_values = np.array([rastrigin(x) for x in bee_area])
        bee_area = bee_area[function_values.argsort()]

        optimal_solution = bee_area[:self.optimal_solution_count]
        suboptimal_solution = bee_area[self.optimal_solution_count:self.
                                       optimal_solution_count +
                                       self.suboptimal_solution_count]

        new_bee_area = []
        for solution in optimal_solution:
            min_search_field = solution - self.field_size
            max_search_field = solution + self.field_size
            new_bee_area.append(solution)
            for _ in range(self.optimal_bee_count - 1):
                bee = np.random.random(self.n) * (
                    max_search_field - min_search_field) + min_search_field
                new_bee_area.append(bee)

        for solution in suboptimal_solution:
            min_search_field = solution - self.field_size
            max_search_field = solution + self.field_size
            new_bee_area.append(solution)
            for _ in range(self.suboptimal_bee_count - 1):
                bee = np.random.random(self.n) * (
                    max_search_field - min_search_field) + min_search_field
                new_bee_area.append(bee)
        return np.array(new_bee_area)


class Solution:
```

Окончание Листинга А.1

```
    def __init__(self):
        self.bee_colony = BeeColony(scout_bee_count=100,
                                    optimal_bee_count=30,
                                    suboptimal_bee_count=10,
                                    optimal_solution_count=10,
                                    suboptimal_solution_count=5,
                                    field_size=0.5)

    def solve(self):
        history = []
        result: np.ndarray | None = None
        best_result_value = float("inf")
        best_result_repeat = 0
        while result is None or best_result_repeat < 1000:
            result = self.bee_colony.solution_step(result)
            function_values = np.array([rastrigin(x) for x in result])
            if np.min(function_values) < best_result_value:
                best_result_value = np.min(function_values)
                best_result = result[function_values.argmin()]
                best_result_repeat = 0
                history.append(best_result_value)
            else:
                best_result_repeat += 1
            print(
                f'Лучший результат {best_result_value} в точке {best_result}.'
                f' Количество повторений: {best_result_repeat}')
        plt.plot(history)
        plt.show()


def main():
    solution = Solution()
    solution.solve()


if __name__ == '__main__':
    main()
```