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

Институт Информационных Технологий
Кафедра Вычислительной Техники

Практическая работа №3
«Алгоритм роя частиц»

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

В основу алгоритма оптимизации роем частиц положена социально-психологическая поведенческая модель толпы. Развитие алгоритма инспирировали такие задачи, как моделирование поведения птиц в стае и рыб в косяке. Целью было обнаружить базовые принципы, благодаря которым, например, птицы в стае ведут себя удивительно синхронно, меняя как по команде направления своего движения, так что стая движется как единое целое. К современному времени концепция алгоритма роя частиц развилась в высокоэффективный алгоритм оптимизации, часто составляющий конкуренцию лучшим модификациям генетического алгоритма.

В настоящее время роевой алгоритм применяются при решении задач численной и комбинаторной оптимизации, обучении искусственных нейронных сетей, построении нечетких контроллеров и т.д. в различных областях науки техники:

- управление энергетическими системами;
- решение NP-трудных комбинаторных проблем;
- задачи календарного планирования;
- оптимизация в мобильной связи;
- оптимизация процессов пакетной обработки;
- оптимизация многокритериальных задач;
- обработка изображений; распознавание образов;
- кластеризация данных;
- биоинформатика;
- проектирование сложных технических систем и т.д.

1 АЛГОРИТМ РОЯ ЧАСТИЦ

1.1 Описание алгоритма

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

$$x_k = \left(x_1^k, x_2^k, \dots, x_n^k\right); v_k = \left(v_1^k, v_2^k, \dots, v_n^k\right), \quad (1.1.1)$$

где
```
(
  equation(block: false, body: [k]),
  [номер частицы],
)
```
;
```
(
  equation(block: false, body: [n]),
  [размерность векторов в задаче],
)
```
.

Далее происходит поиск лучшего решения для каждой частицы, после которого обновляется лучшее решение для всего роя, если какой-то частицей найдено решение, которое лучше текущего. Затем выполняется коррекция скорости для каждой частицы по Формуле 1.1.2.

$$v_i(t+1) = v_i(t) + c_1 r_1(t)(y_i(t) - x_i(t)) + c_2 r_2(t)(\hat{y}(t) - x_i(t)), \quad (1.1.2)$$

где
```
(
  equation(
    block: false,
    body: sequence(
      attach(base: [c], b: [1]),
      [,],
      attach(base: [c], b: [2]),
    ),
  ),
  [положительные коэффициенты ускорения],
)
```
;

```
(
  equation(
    block: false,
    body: sequence(
      attach(base: [r], b: [1]),
      [ ],
      lr(body: sequence([(], [t], [)])),
      [,],
      attach(base: [r], b: [2]),
      [ ],
      lr(body: sequence([(], [t], [)])),
    ),
  ),
  sequence(
    [вектора размерности],
    [ ],
    equation(block: false, body: [n]),
    [, состоящие из случайных чисел из диапазона],
    [ ],
    equation(
      block: false,
      body: lr(body: sequence([(], [0], [;], [ ], [1], [)])),
    ),
    [; при этом,],
    [ ],
    equation(
      block: false,
      body: sequence(
        attach(base: [r], b: [2]),
        [ ],
        lr(body: sequence([(], [t], [)])),
        [=],
        [1],
        [-],
        attach(base: [r], b: [1]),
        [ ],
        lr(body: sequence([(], [t], [)])),
      ),
    ),
  ),
)

;

(
  equation(
    block: false,
    body: sequence(
      attach(base: [y], b: [i]),
      [ ],
      lr(body: sequence([(], [t], [)])),
    ),
  ),
  [позиция i-й частицы, где достигалось лучшее решение],
)

;

(
  equation(
    block: false,
    body: sequence([y], [ ], lr(body: sequence([(], [t], [)]))),
  ),
  [координаты частицы с лучшим решением всего роя],
)

.
```

После этого выполняется коррекция позиции каждой частицы по Формуле 1.1.3.

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (1.1.3)$$

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 роя частиц;
- выбрать тестовую функцию для оптимизации (нахождение глобального минимума);
- произвести ручной расчёт двух итераций алгоритма для трёх частиц;
- разработать программную реализацию алгоритма роя частиц 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 $x_i \in [-5.12, 5.12]$.

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

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.

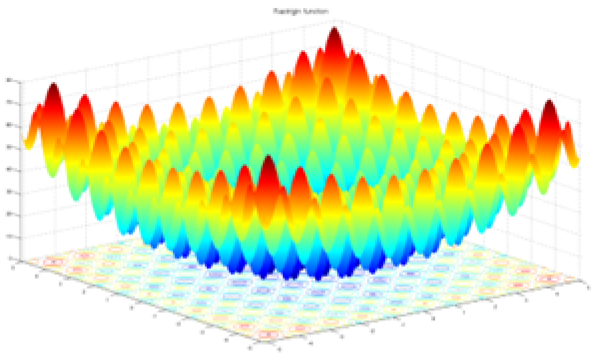

Рисунок 1.3.1 — График функции Растригина

Инициализированы свободные параметры алгоритма:

- $c_1 = c_2 = 2$;
- количество частиц: 3.

Далее созданы три частицы со следующими характеристиками:

$$x_1 = (0.2, 3.5); v_1 = (0, 0)$$
$$x_2 = (1.3, 0.98); v_2 = (0, 0)$$
$$x_3 = (4.87, -3.1); v_3 = (0, 0)$$

Значение целевой функции в первой точке равно 39.2; значение целевой функции у второй частицы равно 15.82; значение целевой функции у третьей частицы равно 38.39.

Лучшая позиция у каждой частицы пока что считается равной текущей позиции каждой частицы, а лучшая позиция всего роя — у второй частицы.

Затем выполняется коррекция скорости по Формуле 1.1.2. Для коррекции скорости первой частицы сгенерирован двумерный вектор из случайных чисел $r_1 = (0.234, 0.567)$.

$$v_{11}(1) = 0 + 2 * 0.234 * (0.2 - 0.2) + 2 * (1 - 0.234) * (1.3 - 0.2) = 1.685$$
$$v_{12}(1) = 0 + 2 * 0.567 * (3.5 - 3.5) + 2 * (1 - 0.567) * (0.98 - 3.5) = -2.182$$

Для коррекции скорости второй частицы сгенерирован двумерный вектор из случайных чисел $r_2 = (0.123, 0.987)$.

$$v_{21}(1) = 0 + 2 * 0.123 * (1.3 - 1.3) + 2 * (1 - 0.123) * (1.3 - 1.3) = 0$$
$$v_{22}(1) = 0 + 2 * 0.987 * (0.98 - 0.98) + 2 * (1 - 0.987) * (0.98 - 0.98) = 0$$

Для коррекции скорости третьей частицы сгенерирован двумерный вектор из случайных чисел $r_2 = (0.555, 0.002)$.

$$v_{31}(1) = 0 + 2 * 0.555 * (4.87 - 4.87) + 2 * (1 - 0.555)(1.3 - 4.87) = -3.177$$
$$v_{32}(1) = 0 + 2 * 0.002 * (-3.1 + 3.1) + 2 * (1 - 0.002) * (0.98 + 3.1) = 8.144$$

После коррекции скоростей выполняется коррекция координат каждой из частиц:

$$x_{11}(1) = 0.2 + 1.685 = 1.885$$
$$x_{12}(1) = 3.5 - 2.182 = 1.318$$
$$x_{21}(1) = 1.3 + 0 = 1.3$$
$$x_{22}(1) = 0.98 + 0 = 0.98$$
$$x_{31}(1) = 4.87 - 3.177 = 1.693$$
$$x_{32}(1) = -3.1 + 8.144 = 5.044$$

После этого происходит переход ко второй итерации. Заново рассчитаны значения функции у каждой частицы:

$$f_1(1) = 21.93$$
$$f_2(1) = 15.82$$
$$f_3(1) = 42.19$$

Значение целевой функции у первой частицы улучшилось, поэтому лучшая позиция теперь $(1.885, 1.318)$. Значение целевой функции у второй частицы не изменилось, поэтому её лучшая позиция осталась $(1.3, 0.98)$. Значение целевой функции у третьей частицы ухудшилось, поэтому лучшая позиция у третьей частицы остаётся как была изначально: $(4.87, -3.1)$. Лучшая позиция всего роя остаётся у второй частицы.

Затем выполняется коррекция скорости по Формуле 1.1.2. Для коррекции скорости первой частицы сгенерирован двумерный вектор из случайных чисел $r_1 = (0.124, 0.5)$.

$$v_{11}(2) = 1.685 + 2 * 0.124 * (1.885 - 1.885) +$$
$$+2 * (1 - 0.124) * (1.3 - 1.885) = 0.66$$
$$v_{12}(2) = 0 + 2 * 0.5 * (1.318 - 1.318) +$$
$$+2 * (1 - 0.5) * (0.98 - 1.318) = -0.33$$

Для коррекции скорости второй частицы сгенерирован двумерный вектор из случайных чисел $r_2 = (0.01, 0.8)$.

$$v_{21}(2) = 0 + 2 * 0.01 * (1.3 - 1.3) + 2 * (1 - 0.01) * (1.3 - 1.3) = 0$$
$$v_{22}(2) = 0 + 2 * 0.8 * (0.98 - 0.98) + 2 * (1 - 0.8) * (0.98 - 0.98) = 0$$

Для коррекции скорости третьей частицы сгенерирован двумерный вектор из случайных чисел $r_3 = (0.4, 0.8)$.

$$v_{31}(2) = 0 + 2 * 0.4 * (4.87 - 1.693) + 2 * (1 - 0.4) * (1.3 - 1.693) = 2.07$$
$$v_{32}(2) = 0 + 2 * 0.8 * (-3.1 - 5.044) + 2 * (1 - 0.8) * (0.98 - 5.044) = -14.656$$

После коррекции скоростей выполняется коррекция координат каждой из частиц:

$$x_{11}(1) = 1.885 + 0.66 = 2.545$$
$$x_{12}(1) = 1.318 - 0.33 = 0.988$$
$$x_{21}(1) = 1.3 + 0 = 1.3$$
$$x_{22}(1) = 0.98 + 0 = 0.98$$
$$x_{31}(1) = 1.693 + 2.07 = 3.763$$
$$x_{32}(1) = 5.044 - 14.656 = -9.612$$

Но координата x_{32} получилась вне области поиска, поэтому она принимается равной -5.12, т.е. позиция третьей частицы: $(3.763, -5.12)$. Заново рассчитаны значения функции у каждой частицы:

$$f_1(2) = 27.08$$
$$f_2(2) = 15.82$$
$$f_3(2) = 52.27$$

1.4 Программная реализация

Для реализации расчётов алгоритма роя частиц написан программный код на языке Python.

В программной реализации зафиксированы следующие параметры:

- количество частиц: 20;
- количество итераций: 30;
- c_1 и c_2: 2.

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

На Рисунке 1.4.1 представлен результат выполнения программы для нахождения оптимального значения функции — график зависимости оптимального решения от номера итерации.

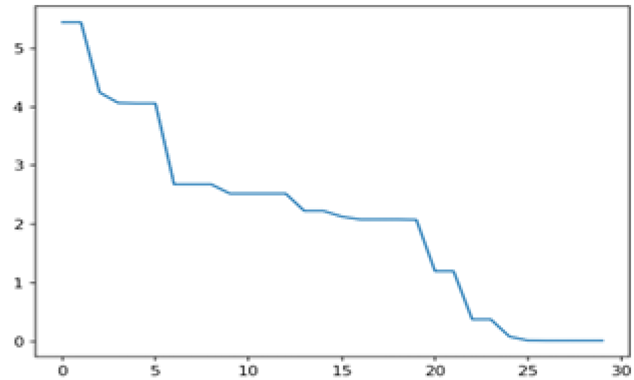

Рисунок 1.4.1 — График зависимости оптимального значения функции от номера итерации

ЗАКЛЮЧЕНИЕ

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

В заключение можно отметить, что роевой алгоритм является мощным инструментом для решения задач оптимизации, в которых стандартные методы недостаточно эффективны из-за наличия множества локальных минимумов. При этом, алгоритм является простым в реализации и имеет мало свободных параметров, из-за чего алгоритм не нуждается в длительной метаоптимизации.

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 роевого алгоритма в задаче оптимизации на языке Python.

Приложение А

Реализация роевого алгоритма в задаче оптимизации на языке Python

Листинг А.1 — Реализация роевого алгоритма

```
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 Particle:
    def __init__(self):
        self._max = 5.12
        self._min = -self._max
        self.n = 2
            self.x = np.random.random(size=self.n) * (self._max - self._min)
+ self._min
        self.best_x = self.x.copy()
        self.speed = np.zeros(shape=(self.n,))

    def correct_speed(self, global_best: np.ndarray):
        c1 = 2
        c2 = 2
        alpha = np.random.random()
        self.speed += c1 * alpha * (self.best_x - self.x) + c2 * (1 - alpha) *
(global_best - self.x)

    def correct_position(self):
        self.x += self.speed
        for i in range(len(self.x)):
            if self.x[i] > self._max:
                self.x[i] = self._max
            elif self.x[i] < self._min:
                self.x[i] = self._min

    def __str__(self):
        return f"Текущая позиция: {self.x} -- лучшая позиция: {self.best_x}"


class Swarm:
    def __init__(self, particle_count: int = 10):
        self.particles = [Particle() for _ in range(particle_count)]
        self.best_solution: np.ndarray | None = None

    def solution_step(self):
        if self.best_solution is None:
            self.best_solution = self.particles[0].best_x.copy()

        for particle in self.particles:
            if rastrigin(particle.x) < rastrigin(particle.best_x):
                particle.best_x = particle.x.copy()
            if rastrigin(particle.best_x) < rastrigin(self.best_solution):
                self.best_solution = particle.best_x.copy()

        for particle in self.particles:
            particle.correct_speed(self.best_solution)
            particle.correct_position()

        return rastrigin(self.best_solution), self.best_solution

    def draw_swarm(self):
        particle_x = [particle.x[0] for particle in self.particles]
        particle_y = [particle.x[1] for particle in self.particles]
        ax = plt.gca()
        ax.set_xlim(-5.12, 5.12)
        ax.set_ylim(-5.12, 5.12)
        plt.scatter(particle_x, particle_y, s=[1.5 for _ in self.particles])
```

Окончание Листинга А.1

```
        plt.show()


class Solution:

    def __init__(self):
        self.swarm = Swarm(particle_count=1000)

    def solve(self):
        steps = 50
        history = []
        for step in range(1, steps + 1):
            value, x = self.swarm.solution_step()
            print(f'Итерация: {step}. Значение: {value} в точке {x}')
            history.append(value)
        plt.plot(history)
        plt.show()


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


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