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

Институт Информационных Технологий
Кафедра Вычислительной Техники

Практическая работа №6
«Электромагнитный алгоритм»

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

Электромагнитный алгоритм предложен Бирбилом (I. Birbil) и Фангом (S.C. Fang) в 2003 году [2]. Этот алгоритм вдохновлен фундаментальными принципами электромагнетизма, а именно взаимодействием между электрическими зарядами и магнитными полями.

В алгоритме пространство поиска представляется как заполненное частицами, каждая из которых обладает определенным электрическим зарядом. Эти частицы взаимодействуют друг с другом посредством электромагнитных сил. Сила притяжения или отталкивания между двумя частицами зависит от их зарядов и расстояния между ними.

Каждого из агентов популяции в электромагнитном алгоритме интерпретируют как заряженную частицу, заряд которой пропорционален значению фитнесс-функции в той точке области поиска, в которой на данной итерации находится агент. Текущий заряд частиц популяции определяет суммарную силу, действующую на данную частицу со стороны других частиц, а также направление и величину её перемещений на текущей итерации. В соответствии с законами электростатики эта сила вычисляется путём векторного суммирования сил притяжения и отталкивания со стороны всех частиц популяции [2].

Электромагнитный алгоритм широко используется для решения различных задач оптимизации, таких как:

- задача коммивояжера;
- распределение ресурсов;
- планирование;
- сетевое проектирование.

1 АЛГОРИТМ РОЯ ЧАСТИЦ

1.1 Описание алгоритма

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

$$X_j = \left(x_1 j, x_2 j, \ldots, x_{nj}\right) \tag{1.1.1}$$

где $j \in [1; S]$ — номер частицы;

n — размерность векторов в задаче.

Затем начинается локальный поиск – для каждого заряда выполняется линейный стохастический поиск 1.1.2 в целях сбора локальной информации об окружении частиц [2, с.22-23].

$$x'_{ij} = x_{ij} + u^{\text{sign}} * U(0; 1) * \alpha(x^+ - x^-) \tag{1.1.2}$$

где u^{sign} — случайное целое число, равное -1 или 1;

$U(0; 1)$ — случайное число от 0 до 1 из равномерного распределения;

$\alpha \in (0; 1)$ — свободный параметр алгоритма;

x^+, x^- — границы рассматриваемой области поиска.

При этом, изменённая координата принимается только в том случае, если её изменение дало улучшение значения целевой функции, иначе происходит переход к следующей итерации локального поиска.

Как правило, свободный параметр α задаётся близким к нулю (порядка $\sim 10^{-2} : 10^{-4}$), поэтому локальный поиск служит только для незначительного улучшения положения точек в пространстве.

После осуществления локального поиска вычисляется лучшая точка, т.е. точка, где достигается минимальное среди всех значение целевой функции.

Затем для каждой частицы вычисляется её заряд по Формуле 1.1.3.

$$q_i = \exp\left(-n * \frac{(\varphi_i - \varphi_{\text{best}})}{\sum_{j,j\neq i}(\varphi_j - \varphi_{\text{best}})}\right) \tag{1.1.3}$$

где φ_i — значение целевой функции в текущей точке;

φ_{best} — значение целевой функции в лучшей точке.

На основе посчитанных зарядов происходит вычисление силы отталкивания и притяжения между частицами (1.1.4). Частица i отталкивается от частицы j, если значение функции у частицы i лучше, чем значение функции у частицы j, и наоборот, притягивается, если значение функции хуже.

$$F_i = \sum_{j=1,j\neq i}^{S} F_{i,j} = \sum_{j=1,j\neq i}^{S} \begin{cases} (X_j - X_i)\frac{q_i q_j}{\|X_j - X_i\|^2}, \text{если } \varphi_j < \varphi_i \\ (X_i - X_j)\frac{q_i q_j}{\|X_j - X_i\|^2}, \text{если } \varphi_i < \varphi_j \end{cases} \tag{1.1.4}$$

Таким образом, лучшая частица на данной итерации притянет к себе все остальные. Далее выполняется перемещение частиц, высчитываемое на основе электромагнитных сил (1.1.5).

$$X_i(t+1) = X_i(t) + U(0;1)\frac{F_i}{\|F_i\|}V_i \tag{1.1.5}$$

где $U(0;1)$ — случайное число от 0 до 1 из равномерного распределения;

$\frac{F_i}{\| F_i \|}$ — нормированный вектор силы;

V_i — вектор скорости, компоненты которого рассчитываются по Формуле 1.1.6.

$$v_{ij} = \begin{cases} (x^+ - x_{ij}), \ F_{ij} > 0 \\ (x_{ij} - x^-), \ F_{ij} \leq 0 \end{cases}, \ j \in [1:S], \ j \neq i \qquad (1.1.6)$$

где x^+, x^- — границы области поиска.

Важно отметить, что лучшая частица должна остаться на своём месте, поэтому она в данной итерации не передвигается, а только притягивает к себе другие частицы.

После осуществления перемещения частиц происходит переход к следующей итерации. Точкой останова алгоритма является достижение максимального числа итераций.

1.2 Постановка задачи

Цель работы: реализовать электромагнитный алгоритм для нахождения оптимального значения функции.

Поставлены следующие задачи:

- изучить электромагнитный алгоритм;
- выбрать тестовую функцию для оптимизации (нахождение глобального минимума);
- произвести ручной расчёт одной итерации алгоритма;
- разработать программную реализацию электромагнитного алгоритма 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.

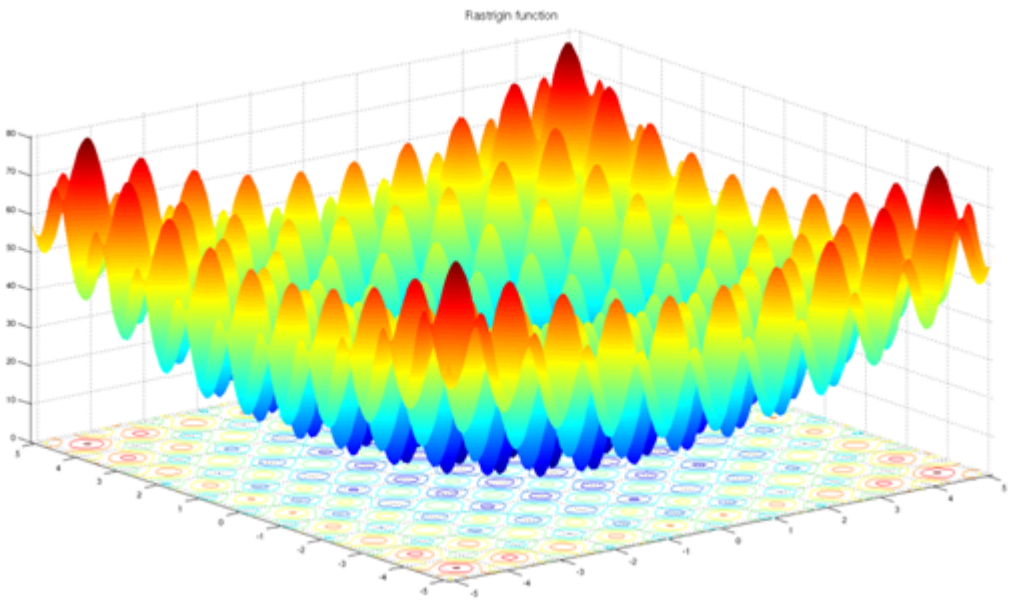

Рисунок 1.3.1 — График функции Растригина

Количество частиц, используемых для ручного расчёта: 4.

Создано 4 частицы со следующими координатами:

$$X_1 = (-1.911, 2.375); f(X_0) = 27.905$$
$$X_2 = (3.325, 4.537); f(X_1) = 65.893$$
$$X_3 = (3.839, -2.436); f(X_2) = 44.592$$
$$X_4 = (0.141, -1.939); f(X_3) = 8.157$$

После выполнения локального поиска (1.1.2) точки имеют следующие координаты:

$$X_1 = (-1.911, 2.368); f(X_0) = 27.550$$
$$X_2 = (3.291, 4.537); f(X_1) = 63.706$$
$$X_3 = (3.873, -2.436); f(X_2) = 43.152$$
$$X_4 = (0.141, -1.961); f(X_3) = 7.813$$

При этом, значение φ_{best} равно 7.813 в точке $(0.141, -1.961)$. Далее приведены расчёты значения заряда для каждой из частиц по Формуле 1.1.3:

$$q_1 = \exp\left(-2 * \frac{27.550 - 7.813}{110.971}\right) = 0.701$$
$$q_2 = \exp\left(-2 * \frac{63.706 - 7.813}{110.971}\right) = 0.365$$
$$q_3 = \exp\left(-2 * \frac{43.152 - 7.813}{110.971}\right) = 0.529$$

При этом, заряд у лучшей частицы будет равен единице, поскольку аргумент в экспоненте будет равен нулю. Затем вычисляются значение силы для каждой частицы по Формуле 1.1.4:

$$F_1 = 0.701 * 0.365 * \frac{(-1.911, 2.368) - (3.291, 4.537)}{31.766} +$$
$$+0.701 * 0.529 * \frac{(-1.911, 2.368) - (3.873, -2.436)}{56.543} +$$
$$+0.701 * 1.000 * \frac{(0.141, -1.961) - (-1.911, 2.368)}{22.953} = (-0.017, -0.118)$$
$$F_2 = 0.365 * 0.701 * \frac{(-1.911, 2.368) - (3.291, 4.537)}{31.766} +$$
$$+0.365 * 0.529 * \frac{(3.873, -2.436) - (3.291, 4.537)}{48.965} +$$
$$+0.365 * 1.000 * \frac{(0.141, -1.961) - (3.291, 4.537)}{52.152} = (-0.062, -0.09)$$
$$F_3 = 0.529 * 0.701 * \frac{(-1.911, 2.368) - (3.873, -2.436)}{56.543} +$$
$$+0.529 * 0.365 * \frac{(3.873, -2.436) - (3.291, 4.537)}{48.965} +$$
$$+0.529 * 1.000 * \frac{(0.141, -1.961) - (3.873, -2.436)}{14.160} = (-0.175, 0.022)$$
$$F_4 = 1.000 * 0.701 * \frac{(0.141, -1.961) - (-1.911, 2.368)}{22.953} +$$

$$+1.000 * 0.365 * \frac{(0.141, -1.961) - (3.291, 4.537)}{52.152} +$$
$$+1.000 * 0.529 * \frac{(0.141, -1.961) - (3.873, -2.436)}{14.160} = (-0.099, -0.16)$$

И наконец, позиции частиц изменяются по Формулам 1.1.5 - 1.1.6. Частица X_4 не передвигается, т.к. она имеет лучшее значение.

Рассчитаем смещение для 1-й частицы. Сгенерировано случайное число $U(0;1) = 0.845$.

Нормированный вектор силы у 1-й частицы: $F_1 = (-0.144, -0.99)$.

Рассчитаны компоненты вектора скорости для 1-й частицы:

$$v_{11} = -1.911 + 5.12 = 3.209$$
$$v_{12} = 2.368 + 5.12 = 7.488$$

Тогда частица сместится на следующую позицию:

$$X_1 = (-1.911, 2.368) + 0.845 * (-0.144, -0.99) * (3.209, 7.488)$$
$$= (-2.301, -3.891)$$
$$F(X_1) = 35.864$$

Рассчитаем смещение для 2-й частицы. Сгенерировано случайное число $U(0;1) = 0.403$ Нормированный вектор силы у 2-й частицы: $F_2 = (-0.563, -0.826)$ Рассчитаны компоненты вектора скорости для 2-й частицы:

$$v_{21} = 3.291 + 5.12 = 8.411$$
$$v_{22} = 4.537 + 5.12 = 9.657$$

Тогда частица сместится на следующую позицию:

$$X_2 = (3.291, 4.537) + 0.403 * (-0.563, -0.826) * (8.411, 9.657)$$

$$= (1.38, 1.317)$$
$$F(X_2) = 35.025$$

Рассчитаем смещение для 3-й частицы. Сгенерировано случайное число $U(0;1) = 0.767$ Нормированный вектор силы у 3-й частицы: $F_3 = (-0.992, 0.123)$ Рассчитаны компоненты вектора скорости для 3-й частицы:

$$v_{31} = 3.873 + 5.12 = 8.993$$
$$v_{32} = 5.12 + 2.436 = 7.556$$

Тогда частица сместится на следующую позицию:

$$X_3 = (3.873, -2.436) + 0.767 * (-0.992, 0.123) * (8.993, 7.556)$$
$$= (-2.973, -1.722)$$
$$F(X_3) = 23.689$$

1.4 Программная реализация

Для реализации расчётов электромагнитного алгоритма написан программный код на языке Python.

В программной реализации зафиксированы следующие параметры:

- количество частиц: 20;
- количество итераций локального поиска: 10;
- количество общих итераций: 100;
- $\alpha = 0.005$.

Код реализации электромагнитного 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 минимального оптимального решения от номера итерации.

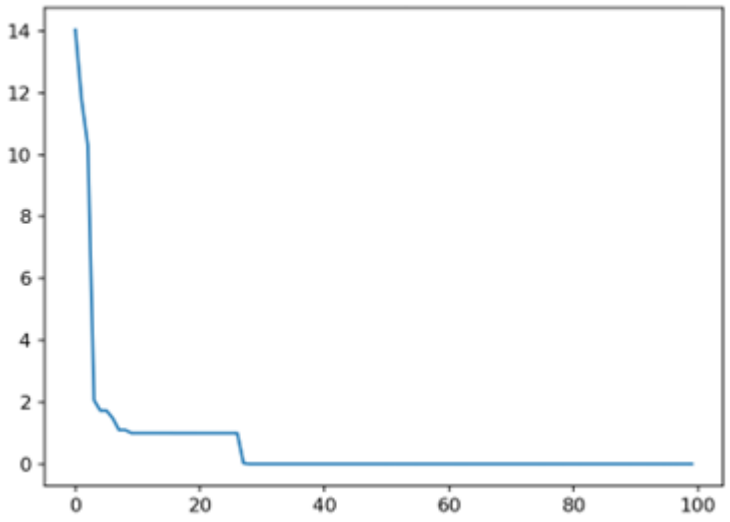

Рисунок 1.4.1 — Результат выполнения программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы выполнены поставленные задачи – изучен электромагнитный алгоритм, 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 электромагнитный алгоритм является мощным инструментом для решения задач оптимизации (например, для нахождения глобального минимума функции от нескольких переменных), в которых стандартные методы недостаточно эффективны из-за наличия множества локальных минимумов. Алгоритм имеет меньшую зависимость от свободных параметров, чем пчелиный алгоритм, но большую, чем роевой алгоритм. В то же время, алгоритм показал быструю сходимость, однако имеет тенденцию сходится к локальным минимумам.

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 электромагнитного алгоритма на языке Python.

Приложение А

Реализация электромагнитного алгоритма на языке Python

Листинг А.1 — Реализация электромагнитного алгоритма

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


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


class EMA:
    def __init__(self, n: int):
        self.n = n
        self.population_size = 10 * n
        self.max_iter = 50 * n
        self.local_iter = 10
        self.scale = 0.005
        self._min = -5.12
        self._max = -self._min

    def calculate_best(self):
        self.values = np.array([rastrigin(x) for x in self.x])
        self.best_value = np.min(self.values)
           self.best_x = self.x[np.where(abs(self.values - self.best_value) <
1e-3)].flatten()

    def create_population(self):
         self.x = np.vstack([self._min + np.random.uniform(0, 1, size=self.n) *
(self._max - self._min)
                             for _ in range(self.population_size)])
        self.calculate_best()

    def local_search(self):
        search_field = self.scale * (self._max - self._min)
        for k, particle in enumerate(self.x):
            cnt = 0
            while cnt < self.local_iter:
                for i in range(self.n):
                    sign = np.random.randint(0, 2) * 2 - 1
                    y = particle.copy()
                    velocity = np.random.uniform()
                    y[i] += sign * velocity * search_field
                    if rastrigin(y) < rastrigin(particle):
                        self.x[k] = y.copy()
                        cnt = self.local_iter
                        break
                    cnt += 1
        self.calculate_best()

    def calculate_force(self):
              self.q = np.exp(-self.n * (self.values - self.best_value) /
(np.sum(self.values - self.best_value)))
        self.force = np.zeros_like(self.x)
        for i in range(self.population_size):
            for j in range(self.population_size):
                if i != j:
                    if self.values[j] < self.values[i]:
                                    self.force[i] += ((self.x[j] - self.x[i]) /
np.linalg.norm(self.x[j] - self.x[i]) ** 2) \
                                             * self.q[i] * self.q[j]
                    else:
                                    self.force[i] += ((self.x[i] - self.x[j]) /
np.linalg.norm(self.x[j] - self.x[i]) ** 2) \
                                             * self.q[i] * self.q[j]

    def move_particles(self):
        for i in range(self.population_size):
```

Окончание Листинга А.1

```
            if abs(self.values[i] - self.best_value) > 1e-3:
                alpha = np.random.uniform()
                velocity = np.ones_like(self.x[i])
               normalized_force = self.force[i] / np.linalg.norm(self.force[i])
                for j in range(self.n):
                    if self.force[i][j] > 0:
                        velocity[j] = self._max - self.x[i][j]
                    else:
                        velocity[j] = self.x[i][j] - self._min
                self.x[i] += alpha * np.multiply(normalized_force, velocity)

    def solve(self):
        self.create_population()
        history = []
        for i in range(self.max_iter):
            history.append(self.best_value)
           print(f'Текущее лучшее значение: {round(self.best_value, 4)} в точке
{list(map(lambda x: round(x, 4), self.best_x))}')
            print(f'Итерация: {i + 1}')
            self.local_search()
            self.calculate_best()
            self.calculate_force()
            self.move_particles()
        plt.plot(history)
        plt.show()



def main():
    ema = EMA(2)
    ema.solve()


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