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

Институт Информационных Технологий
Кафедра Вычислительной Техники

Практическая работа №2
«Метод имитации отжига»

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

Метод имитации отжига — это алгоритм оптимизации, основанный на принципах физического процесса отжига, в котором материал медленно охлаждается с целью достижения состояния минимальной энергии.

Алгоритм имитации отжига предложен в 1953 году Метрополисом (N.C. Metropolis). Данный алгоритм можно считать одним из немногих универсальных алгоритмов решения задач глобальной оптимизации.

Метод имитации отжига особенно актуален в современных условиях, когда многие задачи оптимизации стали слишком сложными для традиционных методов. С увеличением объёма данных и усложнением систем возникает необходимость в алгоритмах, способных находить решения в условиях высокой размерности и множества локальных минимумов. Имитация отжига предоставляет эффективный способ решения таких задач за счёт своей способности преодолевать локальные оптимумы и исследовать глобальные решения, что делает метод востребованным в разнообразных отраслях.

Метод имитации отжига находит широкое применение в таких сферах, как логистика и транспорт (например, для решения задач маршрутизации и оптимизации цепей поставок), телекоммуникации (для оптимизации сетевых ресурсов), экономика (для моделирования и оптимизации портфелей инвестиций), проектирование сложных инженерных систем (например, для оптимального размещения элементов на печатных платах), а также в биоинформатике и химии (для решения задач структурной биологии и моделирования молекул).

1 МЕТОД ИМИТАЦИИ ОТЖИГА

1.1 Описание алгоритма

Алгоритм имитации отжига состоит из следующих шагов:

1. Задание начальных параметров — выбор случайного начального решения и заданного значения температуры.
2. Генерация нового решения в окрестности текущего и оценка его качества.
3. Если новое решение лучше текущего, то алгоритм переходит к нему. Если нет, то переход всё равно может быть выполнен с некоторой вероятностью, зависящей от температуры и разности в качестве решений (1.1.1).

$$h(X,T) = \begin{cases} 1\text{, если } f(X') - f(X) < 0 \\ e^{-\frac{\Delta E}{T}}\text{, если } f(X') - f(X) \geq 0 \end{cases} \quad (1.1.1)$$

4. После этого шага температура снижается по заданному закону, и происходит переход к шагу 2.
5. Точкой останова алгоритма является достижение температурой определённого порога.

1.2 Постановка задачи

Цель работы: реализовать метод имитации отжига для решения задачи коммивояжёра и нахождения оптимального значения функции.

Поставлены следующие задачи:

- изучить метод имитации отжига;
- выбрать предметную область для задачи коммивояжёра и функцию для оптимизации;
- расписать ручной расчёт двух итераций в каждой из задач;

- разработать программный код решения задач методом имитации отжига.

Условие задачи коммивояжёра: дан полный граф, т.е. из каждой вершины можно пройти в любую другую вершину. В этом графе нужно найти полный путь минимальной длины, т.е. обойти каждую вершину в графе по одному разу.

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 предметная область для задачи коммивояжёра: в торговом центре расположено n магазинов. Человеку нужно пройтись по всем этим магазинам, при этом ему нужно затратить как можно меньше усилий на это, т.е. общий пройденный путь должен быть минимально возможным. Поэтому нужно определить минимальный путь, позволяющий обойти все магазины.

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 Задача коммивояжёра

1.3.1 Математическая модель

Для ручного расчёта число магазинов в задаче взято равным 6. Для каждого магазина случайным образом сгенерированы координаты в двумерном пространстве (диапазон координат от -10 до 10). Эти данные представлены заданы в Таблице 1.3.1.

Таблица 1.3.1 — Характеристики магазинов

Номер магазина	Координата по x	Координата по y
1	92	-97
2	-67	28
3	-14	-90
4	-67	72
5	19	75
6	77	-97

Важно, что у дома нулевые координаты.

Для расчёта расстояний между вершинами в графе использовалось Евклидово расстояние для точек в двумерном пространстве (1.3.1).

$$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2} \tag{1.3.1}$$

Ниже приведены расчёты длины каждого ребра в графе, т.е. рассчитаны длины путей между каждой парой вершин.

$$\sqrt{(92 + 67)^2 + (-97 - 28)^2} = 202.25$$
$$\sqrt{(92 + 14)^2 + (-97 + 90)^2} = 106.23$$
$$\sqrt{(92 + 67)^2 + (-97 - 72)^2} = 232.04$$
$$\sqrt{(92 - 19)^2 + (-97 - 75)^2} = 186.85$$
$$\sqrt{(92 - 77)^2 + (-97 + 97)^2} = 15.0$$
$$\sqrt{(-67 + 14)^2 + (28 + 90)^2} = 129.36$$
$$\sqrt{(-67 + 67)^2 + (28 - 72)^2} = 44.0$$
$$\sqrt{(-67 - 19)^2 + (28 - 75)^2} = 98.01$$
$$\sqrt{(-67 - 77)^2 + (28 + 97)^2} = 190.69$$
$$\sqrt{(-14 + 67)^2 + (-90 - 72)^2} = 170.45$$
$$\sqrt{(-14 - 19)^2 + (-90 - 75)^2} = 168.27$$
$$\sqrt{(-14 - 77)^2 + (-90 + 97)^2} = 91.27$$
$$\sqrt{(-67 - 19)^2 + (72 - 75)^2} = 86.05$$

$$\sqrt{(-67-77)^2+(72+97)^2}=222.03$$
$$\sqrt{(19-77)^2+(75+97)^2}=181.52$$

Рассчитанные длины рёбер сведены в Таблицу 1.3.2 с указанием вершин, составляющих ребро.

Таблица 1.3.2 — Длины рёбер в графе

Ребро	**Длина ребра**
0 → 1	202.25
0 → 2	106.23
0 → 3	232.04
0 → 4	186.85
0 → 5	15.00
1 → 2	129.36
1 → 3	44.00
1 → 4	98.01
1 → 5	190.69
2 → 3	170.45
2 → 4	168.27
2 → 5	91.27
3 → 4	86.05
3 → 5	222.03
4 → 5	181.52

1.3.2 Ручной расчёт

Сначала нужно составить путь случайным образом: $0 \to 4 \to 5 \to 3 \to 2 \to 6 \to 1 \to 0$. Длина полученного пути: $S_0 = 10.82 + 10.82 + 5.0 + 12.65 + 5.39 + 10.3 + 5.0 + 5.0 + 6.0 + 6.0 = 76.96(\text{м})$.

За изначальную температуру взята $T_0 = 100$.

Для первой итерации вместо четвёртого магазина поставлен пятый магазин, и после этого перестроен путь: $0 \to 5 \to 4 \to 3 \to 2 \to 6 \to 1 \to 0$. После этого проводится расчёт длины текущего пути: $S_1 = 5.0 + 7.81 + 15.81 + 10.3 + 8.06 + 8.06 + 10.82 + 10.82 + 5.0 + 5.0 = 86.68(\text{м})$. Длина текущего пути оказалась больше длины лучшего пути, поэтому проводится расчёт вероятности перехода к текущему решению.

$$H = e^{-\frac{\Delta l}{T_0}} = e^{-\frac{86.68-76.96}{100}} \approx 0.378$$

Вероятность, выданная псевдослучайным генератором чисел от 0 до 1, равна 0.363, что меньше 0.378. Поэтому текущее решение принимается как лучшее.

В конце первой итерации температура уменьшается в два раза по сравнению с изначальной: $T_1 = \frac{T_0}{2} = \frac{100}{2} = 50$.

Для второй итерации на место третьего магазина поставлен шестой, после чего перестроен путь: $0 \to 5 \to 1 \to 2 \to 6 \to 3 \to 4 \to 0$.

Длина получившегося пути равна $S_2 = 5.0 + 7.81 + 15.81 + 10.3 + 5.0 + 7.62 + 8.06 + 10.82 + 10.82 = 81.23(\text{м})$.

Длина получившегося пути меньше длины лучшего пути, поэтому текущее решение сразу принимается.

В конце второй итерации температура уменьшается в два раза по сравнению с температурой на текущей итерации: $T_2 = \frac{T_1}{2} = \frac{50}{2} = 25$.

1.4 Поиск глобального минимума

Выбранная функция: функция Растригина от двух переменных. Её формула представлена формулой 1.2.1.

На Рисунке 1.4.1 представлен график этой функции.

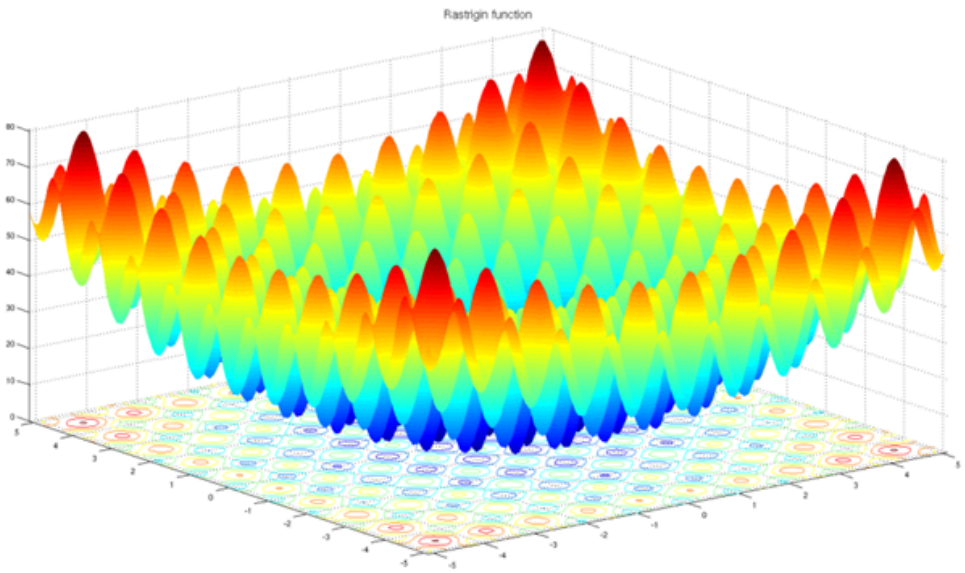

Рисунок 1.4.1 — График функции Растригина

Координаты начальной точки сгенерированы случайным образом и равны (2.75, 3.17). Значение функции в этой точке равно 32.53.

На первой итерации температура равна $T_0 = 100$.

Текущее решение на каждой итерации генерируется с использованием распределения Коши (1.4.1), где $D = 2$, т.к. задача рассматривается в двумерном пространстве.

$$g(x, x', T) = \frac{1}{\pi^D} \prod_{i=1}^{D} \frac{T}{|x' - x|^2 + T^2} \tag{1.4.1}$$

Текущее решение на первой итерации: (4.7, 5). Значение функции в этой точке равно 60.18.

Поскольку текущее решение оказалось хуже лучшего, то проводится расчёт вероятности перехода к этому решению.

$$H = e^{-\frac{\Delta f}{T_0}} = e^{-\frac{60.18-32.53}{100}} \approx 0.758$$

Вероятность, выданная псевдослучайным генератором чисел от 0 до 1, равна 0.923, что больше 0.758. Поэтому текущее решение отбрасывается. Лучшее решение после первой итерации: (2.75, 3.17), значение функции: 32.53.

После выполнения первой итерации температура изменена в соответствии с законом Коши.

$$T_1 = \frac{T_0}{k^{\frac{1}{D}}} = \frac{100}{1^{\frac{1}{2}}} = 100$$

При переходе на вторую итерацию текущим решением выбрано: (1.23, 1.35). Значение функции в этой точке равно 27.96.

Текущее решение оказалось оптимальнее лучшего, поэтому оно автоматически принимается. Лучшее решение после второй итерации: (1.23, 1.35), значение функции: 27.96.

После выполнения второй итерации температура изменена в соответствии с законом Коши.

$$T_2 = \frac{T_0}{k^{\frac{1}{D}}} = \frac{100}{2^{\frac{1}{2}}} \approx 70,711$$

1.5 Программная реализация

Для реализации расчётов метода имитации отжига написан программный код на языке Python.

В программной реализации задачи коммивояжёра зафиксированы следующие параметры:

- количество магазинов: 10;
- диапазон координат магазинов: $[-100; 100]$.

Код, реализующий решение поставленной задачи коммивояжёра методом имитации отжига, представлен в Листинге А.1. Код, отвечающий за генерацию данных для задачи, представлен в Листинге А.2, а сгенерированные данные представлены в Листинге А.3.

На Рисунке 1.5.1 представлен результат выполнения программы, решающей задачу коммивояжёра — построенный граф после завершения алгоритма. Вертикальная и горизонтальная оси являются координатными осями.

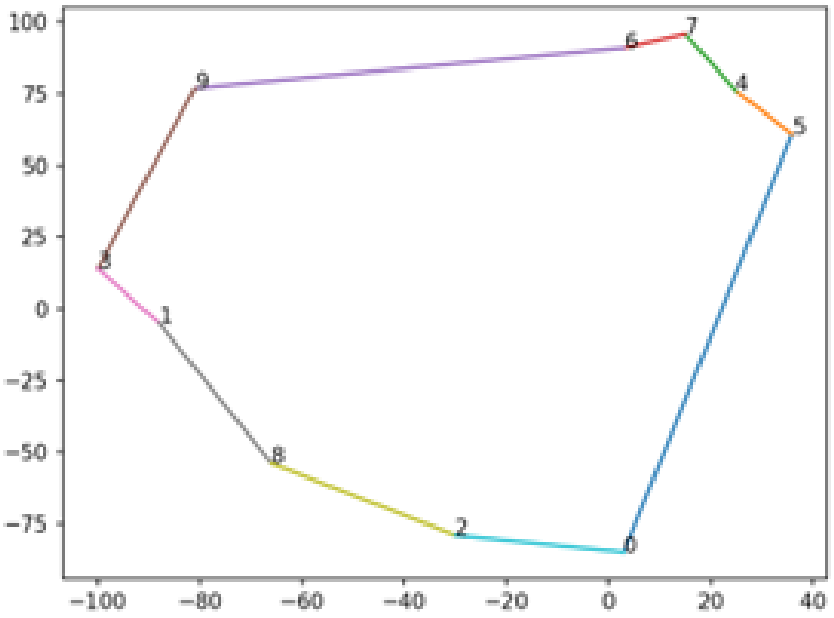

Рисунок 1.5.1 — Построенный минимальный путь в графе

На Рисунке 1.5.2 представлен график, отражающий изменение длины пути в графе в течение выполнения итераций алгоритма.

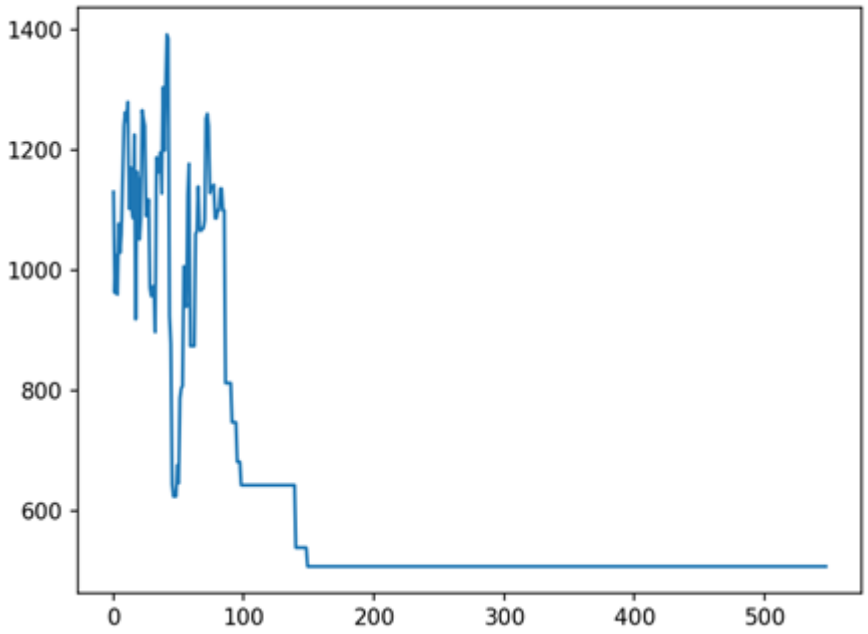

Рисунок 1.5.2 — График изменения длины пути

Код реализации имитации отжига методом Коши для нахождения оптимального значения функции представлен в Листинге Б.1.

На Рисунке 1.5.3 представлен результат выполнения программы для нахождения оптимального значения функции — график зависимости оптимального решения от номера итерации.

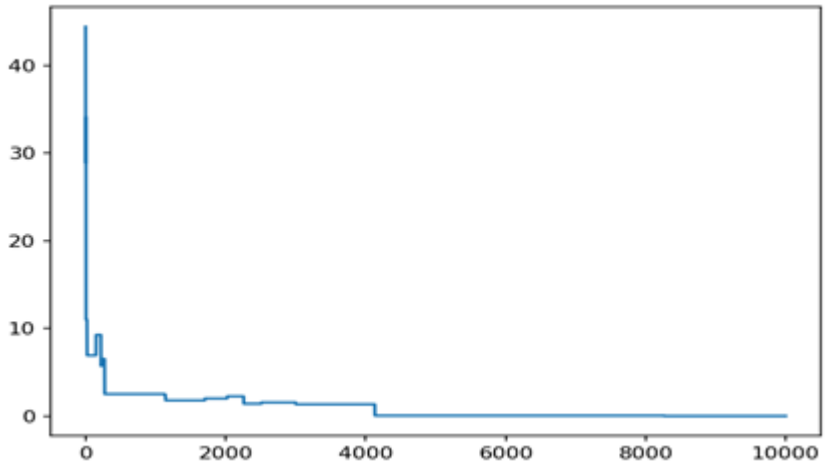

Рисунок 1.5.3 — График зависимости оптимального значения функции от номера итерации

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы выполнены поставленные задачи – изучен метод имитации отжига, произведён его ручной расчёт для решения задачи коммивояжера и задачи поиска глобального минимума функции, а также разработаны программы на языке Python для решения поставленной задачи коммивояжёра – обхода всех магазинов, и для нахождения глобального минимума функции Растригина от двух переменных.

В заключение можно отметить, что метод имитации отжига является мощным инструментом для решения задач оптимизации, в которых стандартные методы недостаточно эффективны из-за наличия множества локальных минимумов. Благодаря своей способности к глобальному поиску и умению избегать застревания в локальных решениях, алгоритм находит широкое применение в различных областях, от логистики и сетевого планирования до биоинформатики и финансов.

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 — Код реализации метода имитации отжига в задаче коммивояжёра на языке Python.

Приложение Б — Код реализации метода имитации отжига в задаче оптимизации функции на языке Python.

Приложение А

Код реализации метода имитации отжига в задаче коммивояжёра на языке Python

Листинг А.1 — Код файла generate.py

```
import pandas as pd
import numpy as np


def generate(min_coord: int, max_coord: int, n_places: int):
    coord_x = np.random.randint(min_coord, max_coord, size=(n_places, ))
    coord_y = np.random.randint(min_coord, max_coord, size=(n_places, ))
    return coord_x, coord_y


def main():
    min_coord, max_coord, n = -100, 100, 10
    x, y = generate(min_coord, max_coord, n)
    df = pd.DataFrame({'x': x, 'y': y})
    df.to_csv('data.csv', index=False)


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

Листинг А.2 — Код файла data.csv

```
x,y
3,-85
-88,-5
-30,-79
-100,14
25,76
36,61
3,91
15,96
-66,-54
-81,77
```

Листинг А.3 — Код файла main.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random


class Path:
    graph: np.ndarray | None = None
    x: np.ndarray | None = None
    y: np.ndarray | None = None

    def __init__(self,
                 path: list[int] | None = None,
                 graph: np.ndarray | None = None,
                 x: np.ndarray | None = None,
                 y: np.ndarray | None = None):
        if path is None:
            self.path = [0]
        else:
            self.path = path

        if Path.graph is None:
            Path.graph = graph
        if Path.x is None:
            Path.x = x
        if Path.y is None:
            Path.y = y
```

Продолжение Листинга А.3

```
        if Path.graph is None:
            raise Exception('Ошибка с данными о графе')

    def create_new_path(self):
        while not all([x in self.path for x in range(Path.graph.shape[0])]):
            possible_places = [
                i for i in range(Path.graph.shape[0])
                if i not in self.path
            ]
            next_place = random.choice(possible_places)
            self.path.append(next_place)
        self.path.append(0)

    def get_replacements(self):
        possible_replacements = [
            self.path[i] for i in range(1, len(self.path))
            if self.path[i] != 0
        ]
        while True:
            first, second = random.choices(possible_replacements, k=2)
            if first != second:
                break
        first_index, second_index = self.path.index(first), self.path.index(
            second)
        if first_index > second_index:
            first_index, second_index = second_index, first_index
            first, second = second, first
        return first_index, second_index

    def recreate_path(self):
        first_index, second_index = self.get_replacements()
        self.path = [elem for elem in self.path]
      self.path[first_index], self.path[second_index] = self.path[second_index],
self.path[first_index]
        print(f'Замены: {self.path[first_index]}, {self.path[second_index]}')

    @property
    def length(self) -> float:
        length = 0
        for i in range(1, len(self.path)):
            length += Path.graph[self.path[i]][self.path[i - 1]]
        return length

    def __str__(self) -> str:
        return ', '.join(map(str, self.path))

    def print_verbose(self):
        result_str = ''
        for i, point in enumerate(self.path):
            result_str += str(point)

            if i != len(self.path) - 1:
                result_str += ' -> '
        return result_str

    def print_length(self):
        cum_length = []
        for i in range(1, len(self.path)):
            edge_len = Path.graph[self.path[i]][self.path[i - 1]]
            if not cum_length:
                cum_length.append(edge_len)
            else:
                cum_length.append(edge_len)
        return " + ".join(map(lambda x: str(round(x, 2)), cum_length))

    def draw_path(self):
        for i in range(len(Path.graph)):
            plt.text(Path.x[i], Path.y[i], f'{i}')
        for v0, v1 in zip(self.path, self.path[1:]):
            x = (Path.x[v0], Path.x[v1])
```

Продолжение Листинга А.3

```
            y = (Path.y[v0], Path.y[v1])
            plt.plot(x, y)
        plt.show()


class Solution:

    def __init__(self, file_path):
        self.file_path = file_path
        self.current_path: Path
        self.best_path: Path | None = None

    def create_graph(self):
        df = pd.read_csv(self.file_path)
        # df = pd.concat([pd.DataFrame([[0, 0, 0]], columns=df.columns), df],
        #                ignore_index=True)
        x = df['x'].to_numpy()
        y = df['y'].to_numpy()
        graph = np.ndarray(shape=(len(df), len(df)))
        for i in range(len(df)):
            for j in range(len(df)):
                graph[i][j] = np.sqrt((x[i] - x[j])**2 + (y[i] - y[j])**2)
                if i < j:
                    sign_1 = '-' if x[j] >= 0 else '+'
                    sign_2 = '-' if y[j] >= 0 else '+'
                        print(f'sqrt(({x[i]} {sign_1} {abs(x[j])})^2 + ({y[i]}
{sign_2} {abs(y[j])})^2) = {round(graph[i][j], 2)} \\')

        for i in range(len(df)):
            for j in range(len(df)):
                if i < j:
                    print(f'[{i} #math.arrow {j}], [{graph[i][j]: .2f}],')

        self.current_path = Path(graph=graph, x=x, y=y)
        self.current_path.create_new_path()

    def solve(self):
        temperature = 1000
        history = []
        self.best_path = self.current_path
        history.append(self.best_path.length)
        print('Начальный путь: ' + self.best_path.print_verbose())
            print(f'Длина начального пути: {self.best_path.print_length()} =
{self.best_path.length:.2f} (м)')
        print('============\n')
        i = 0
        while temperature > 1:
            i += 1
            print(f'Температура: {temperature}')
            print('=============')
            self.current_path = Path(self.best_path.path)
            self.current_path.recreate_path()
            print('Лучший путь: ' + self.best_path.print_verbose())
            print(f'Длина лучшего пути: {self.best_path.length:.2f} м')
            print('Текущий путь: ' + self.current_path.print_verbose())
             print(f'Длина текущего пути: {self.current_path.print_length()} =
{self.current_path.length:.2f} (м)')
            if self.current_path.length < self.best_path.length:
                self.best_path = self.current_path
            else:
                prob_lim = np.exp(-(self.current_path.length -
                                    self.best_path.length) /
                                  (temperature))
                probability = np.random.random()
                print(f'H = {prob_lim}; p = {probability}')
                if probability < prob_lim:
                    self.best_path = self.current_path
            temperature *= 0.9
            print('==============\n')
            history.append(self.best_path.length)
```

Окончание Листинга А.3

```
        print('Найденное лучшее решение: ' + self.best_path.print_verbose())
        print(f'Длина найденного решения: {self.best_path.length:.2f} m')
        plt.plot(history)
        plt.show()
        self.best_path.draw_path()


def main():
    solution = Solution('backup_6.csv')
    solution.create_graph()
    solution.solve()


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

Приложение Б

Код реализации метода имитации отжига в задаче оптимизации функции на языке Python

Листинг Б.1 — Реализация метода имитации отжига

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


class Solution:

    def __init__(self):
        self.current_solution: np.ndarray
        self.best_solution: np.ndarray | None = None
        self._max = 5.12
        self._min = -self._max
        self.n = 2

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

    def init_solution(self):
      self.best_solution = self.current_solution = np.random.random(size=self.n)
* (self._max - self._min) + self._min

    @staticmethod
    def cauchy_distribution(x, main_x, temperature):
         return ((1 / np.pi) * temperature / ((x - main_x) ** 2 + temperature
** 2))

    def generate_solution(self, temperature: float):
        while True:
              new_x = np.random.random(size=self.n) * (self._max - self._min)
+ self._min
               p_distribute = Solution.cauchy_distribution(self.best_solution,
new_x, temperature)
            p = np.random.random(size=p_distribute.shape)
            if np.all(p <= p_distribute):
                return new_x

    def solve(self):
        temperature = 10
        t0 = temperature
        history = []
        history.append(self.rastrigin(self.best_solution))
        print(
              f'Исходное решение: ({", ".join(map(lambda x: str(round(x, 2)),
self.current_solution))})'
        )
        print(
          f'Исходное значение функции: {self.rastrigin(self.current_solution)}')
        print('============\n')
        k = 2
        while t0 > 0.1:
            print(f'Температура: {t0}')
            print('=============')
            self.current_solution = self.generate_solution(temperature)
            current_rastrigin, best_rastrigin = self.rastrigin(
                self.current_solution), self.rastrigin(self.best_solution)
            print(
                    f'Лучшее решение: ({", ".join(map(lambda x: str(round(x,
2)), self.best_solution))})')
            print(f'Лучшее значение функции: {best_rastrigin}')
            print(
                  f'Текущее решение: ({", ".join(map(lambda x: str(round(x,
2)), self.current_solution))})'
```

Окончание Листинга Б.1

```
            )
            print(f'Текущее значение функции: {current_rastrigin}')
            if current_rastrigin < best_rastrigin:
                self.best_solution = self.current_solution
            else:
                prob_lim = np.exp(-(current_rastrigin - best_rastrigin) / t0)
                probability = np.random.random()
                print(f'H = {prob_lim}; p = {probability}')
                if probability < prob_lim:
                    self.best_solution = self.current_solution
            t0 = temperature / (k ** (1 / self.n))
            k += 1
            print('===============\n')
            history.append(self.rastrigin(self.best_solution))

        best_rastrigin = self.rastrigin(self.best_solution)
          print(f'Лучшее решение: ({", ".join(map(lambda x: str(round(x, 2)),
self.best_solution))})')
        print(f'Лучшее значение функции: {best_rastrigin}')
        plt.plot(history)
        plt.show()


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


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