

|  |
| --- |
| 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)*

Преподаватель \_\_Железняк Л.М.\_\_ *(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

# СОДЕРЖАНИЕ

[Введение 3](#_Toc182400705)

[1 Метод имитации отжига 4](#_Toc182400706)

[1.1 Постановка задачи 4](#_Toc182400707)

[1.2 Задача коммивояжёра 5](#_Toc182400708)

[1.2.1 Математическая модель 5](#_Toc182400709)

[1.2.2 Ручной расчёт 8](#_Toc182400710)

[1.3 Поиск глобального минимума 11](#_Toc182400711)

[1.4 Программная реализация 12](#_Toc182400712)

[Заключение 15](#_Toc182400713)

[Список информационных источников 16](#_Toc182400714)

[Приложения 17](#_Toc182400715)

# ВВЕДЕНИЕ

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

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

Метод имитации отжига — это метод поиска оптимального решения, который позволяет временные ухудшения решения, чтобы избежать застревания в локальных минимумах и достичь глобального оптимума.

Алгоритм метода:

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

## 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: обойти все магазинов, пройдя минимально возможное расстояние.

Выбранная функция для оптимизации: функция Растригина (1.1). 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 .

(1.1)

## 1.2 Задача коммивояжёра

### 1.2.1 Математическая модель

Рядом с домом человека находится n магазинов. Каждый день ему нужно обходить каждый из них, но это нельзя сделать за один заход, поскольку все пакеты нести одновременно тяжело. Поэтому человеку нужно относить пакеты домой, если их стало тяжело нести. В поставленных условиях нужно определить минимальный путь, позволяющий обойти все магазины.

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), а также массы пакетов, которые там закупаются (диапазон масс от 1 до 5). Эти данные представлены заданы в Таблице 1.2.1.

Таблица 1.2.1 – Характеристики магазинов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер магазина | Координата по x | Координата по y | Масса (кг) |
| 1 | 6 | 0 | 2 |
| 2 | -9 | -5 | 1 |
| 3 | -4 | -7 | 1 |
| 4 | 6 | 9 | 2 |
| 5 | 0 | 5 | 1 |
| 6 | 3 | -4 | 3 |

У дома нулевые координаты, а также при его посещении человек оставляет все свои пакеты.

Человек не может нести с собой пакеты с суммарной массой больше 4 кг, поэтому при наборе этой массы он обязательно зайдёт домой. Но также человеку не запрещено посещать дом после любого магазина.

Для расчёта расстояний между вершинами в графе использовалось Евклидово расстояние для точек в двумерном пространстве (1.2).

|  |  |
| --- | --- |
|  | (1.2) |

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

Рассчитанные длины рёбер сведены в Таблицу 1.2.2 с указанием вершин, составляющих ребро.

Таблица 1.2.2 – Длины рёбер в графе

|  |  |
| --- | --- |
| Ребро | Длина ребра |
| 0 - 1 | 6.0 |
| 0 - 2 | 10.3 |
| 0 - 3 | 8.06 |
| 0 - 4 | 10.82 |
| 0 - 5 | 5.0 |
| 0 - 6 | 5.0 |
| 1 - 2 | 15.81 |
| 1 - 3 | 12.21 |
| 1 - 4 | 9.0 |
| 1 - 5 | 7.81 |
| 1 - 6 | 5.0 |
| 2 - 3 | 5.39 |
| 2 - 4 | 20.52 |
| 2 - 5 | 13.45 |
| 2 - 6 | 12.04 |

Продолжение Таблицы 1.2.2

|  |  |
| --- | --- |
| 3 - 4 | 18.87 |
| 3 - 5 | 12.65 |
| 3 - 6 | 7.62 |
| 4 - 5 | 7.21 |
| 4 - 6 | 13.34 |
| 5 - 6 | 9.49 |

### 1.2.2 Ручной расчёт

Сначала нужно составить путь случайным образом (Листинг 1.2.1):

Листинг 1.2.1 – Первоначальный путь

0 (0) -> 4 (2) -> 0 (0) -> 5 (1) -> 3 (2) -> 2 (3) -> 0 (0) -> 6 (3) -> 0 (0) -> 1 (2) -> 0 (0)

Перед скобками стоит номер посещённого магазина или 0, если это дом, а в скобках стоит суммарная масса пакетов после посещения магазина.

Длина полученного пути: S0 = 10.82 + 10.82 + 5.0 + 12.65 + 5.39 + 10.3 + 5.0 + 5.0 + 6.0 + 6.0 = 76.96 (м)

Построенный путь представлен на Рисунке 1.2.1. Каждая вершина подписана своим номером из Таблицы 1.2.1

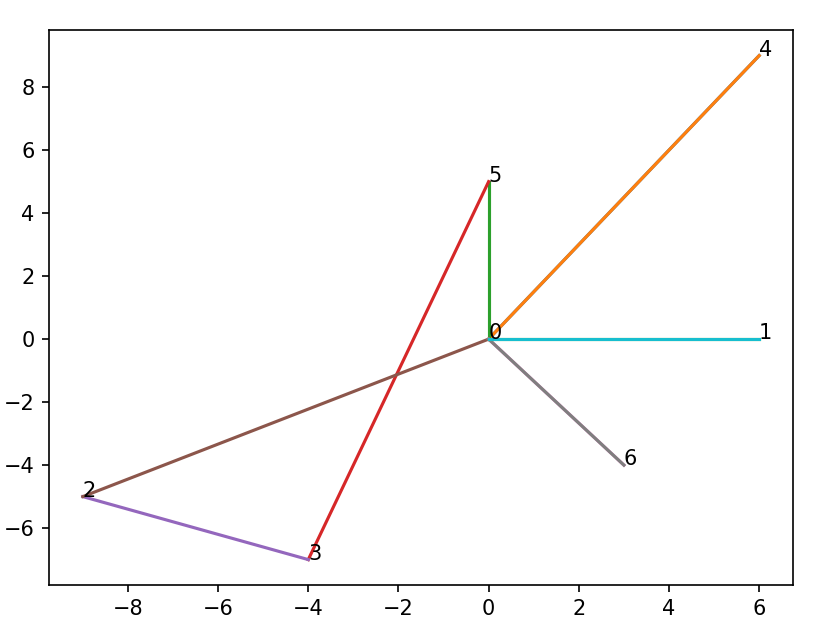


Рисунок 1.2.1 – Изначальный путь

За изначальную температуру взята T0 = 100.

Для первой итерации вместо четвёртого магазина поставлен пятый магазин, и после этого перестроен путь (Листинг 1.2.2).

Листинг 1.2.2 – Текущий путь на первой итерации

0 (0) -> 5 (1) -> 1 (3) -> 2 (4) -> 0 (0) -> 3 (1) -> 0 (0) -> 4 (2) -> 0 (0) -> 6 (3) -> 0 (0)

После этого проводится расчёт длины текущего пути: S1 = 5.0 + 7.81 + 15.81 + 10.3 + 8.06 + 8.06 + 10.82 + 10.82 + 5.0 + 5.0 = 86.68 (м).

Длина текущего пути оказалась больше длины лучшего пути, поэтому проводится расчёт вероятности перехода к текущему решению (1.3).

|  |  |
| --- | --- |
|  | (1.3) |

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

На Рисунке 1.2.2 представлен график лучшего пути после первой итерации.

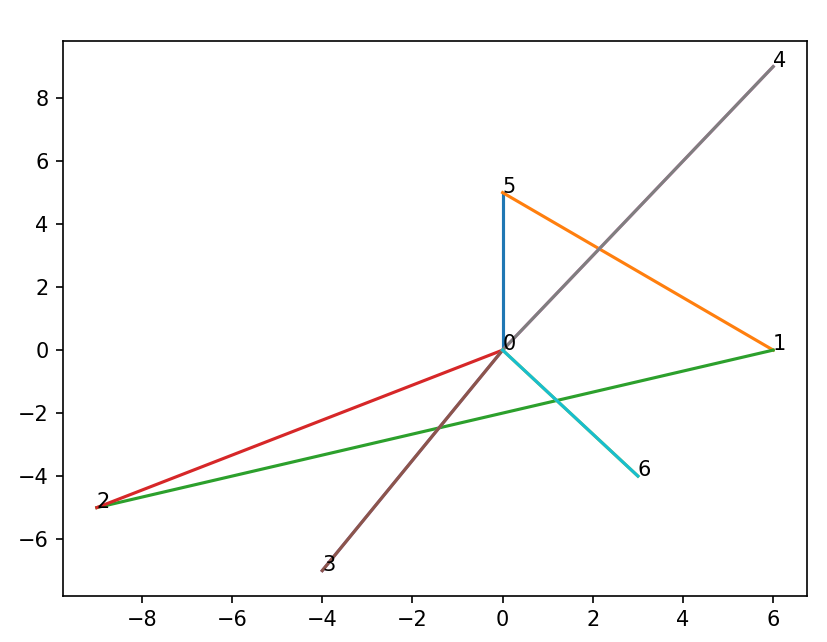


Рисунок 1.2.2 – Лучший путь после первой итерации

В конце первой итерации температура уменьшается в два раза по сравнению с изначальной: T1 = T0 / 2 = 100 / 2 = 50

Для второй итерации на место третьего магазина поставлен шестой, после чего перестроен путь (Листинг 1.2.3).

Листинг 1.2.3 – Текущий путь на второй итерации

0 (0) -> 5 (1) -> 1 (3) -> 2 (4) -> 0 (0) -> 6 (3) -> 3 (4) -> 0 (0) -> 4 (2) -> 0 (0)

Длина получившегося пути равна S2 = 5.0 + 7.81 + 15.81 + 10.3 + 5.0 + 7.62 + 8.06 + 10.82 + 10.82 = 81.23 (м).

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

На Рисунке 1.2.3 представлен график лучшего пути после второй итерации.

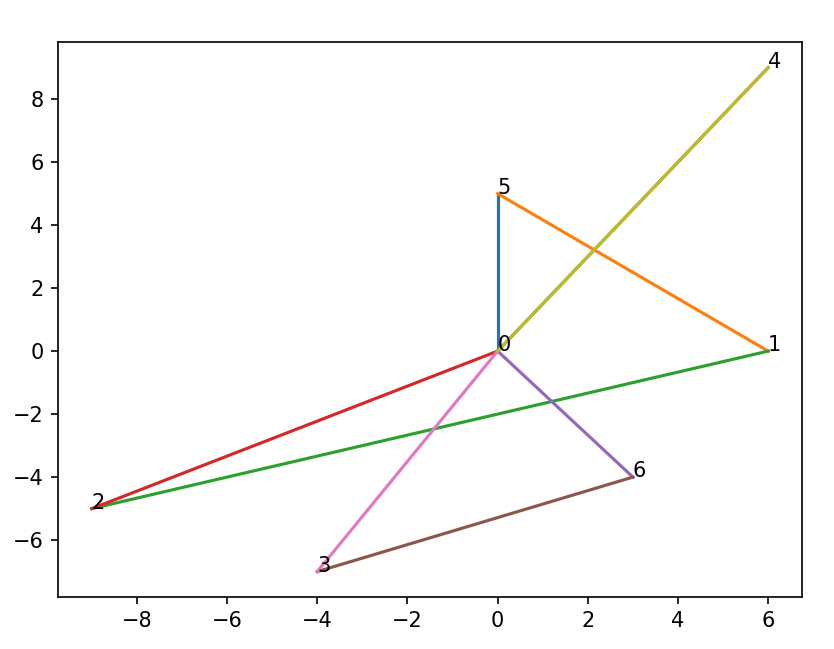


Рисунок 1.2.3 – Лучший путь после второй итерации

В конце второй итерации температура уменьшается в два раза по сравнению с температурой на текущей итерации: T2 = T1 / 2 = 50 / 2 = 25.

## 1.3 Поиск глобального минимума

Выбранная функция: функция Растригина от двух переменных. Её формула представлена формулой (1.1).

На Рисунке 1.3.1 представлен график этой функции.



Рисунок 1.3.1 – 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 T0 = 100.

Текущее решение на каждой итерации генерируется с использованием распределения Коши (1.4), где D = 2, т.к. задача рассматривается в двумерном пространстве.

|  |  |
| --- | --- |
|  | (1.4) |

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 (1.5).

|  |  |
| --- | --- |
|  | (1.5) |

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 (1.6).

|  |  |
| --- | --- |
|  | (1.6) |

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 (1.6).

|  |  |
| --- | --- |
|  |  |

## 1.4 Программная реализация

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

В программной реализации задачи коммивояжёра зафиксированы следующие параметры:

* количество магазинов: 20;
* максимальный суммарный вес пакетов: 50 кг;
* диапазон координат магазинов: [-50; 50];
* диапазон весов пакетов: [1; 7].

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 1.4.1 представлен результат выполнения программы, решающей задачу коммивояжёра – построенный граф после завершения алгоритма. Вертикальная и горизонтальная оси являются координатными осями.

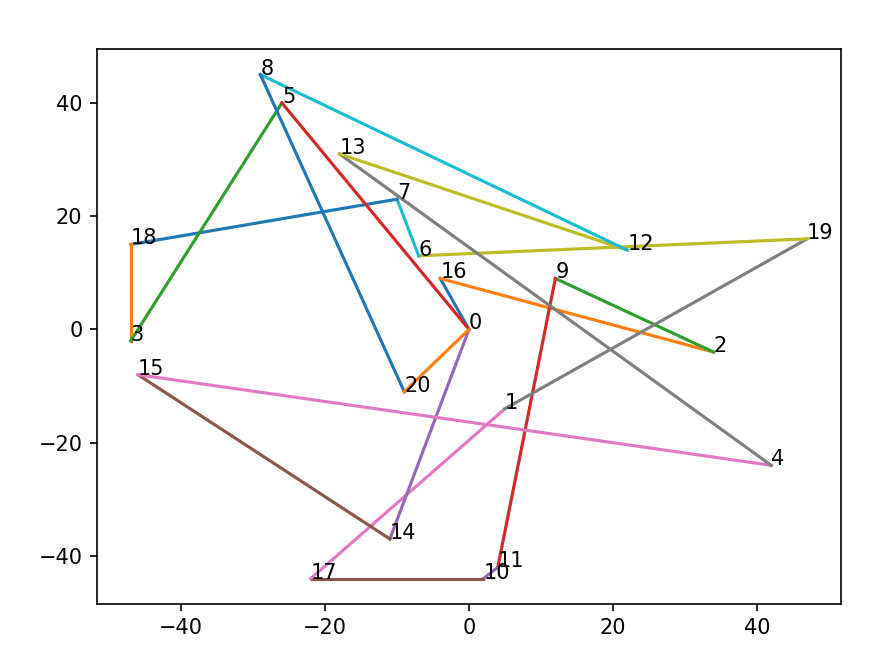


Рисунок 1.4.1 – Построенный граф

На Рисунке 1.4.2 представлен график, отражающий изменение длины пути в графе в течение выполнения итераций алгоритма. На вертикальной оси отражена длина пути в метрах, а на горизонтальной – номер итерации. Оптимальное решение найдено приблизительно на 15 итерации.

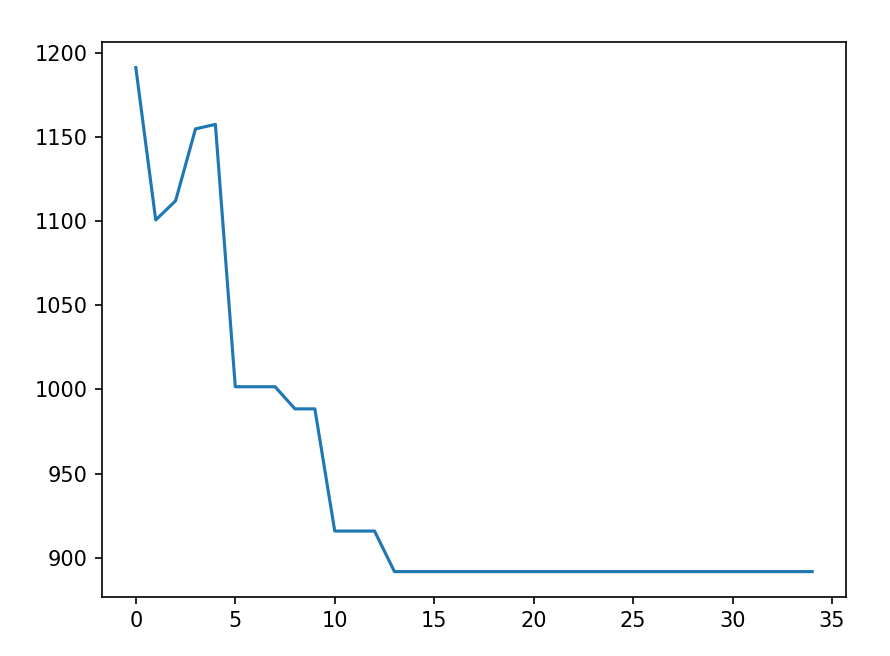


Рисунок 1.4.2 – График изменения длины пути

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

На Рисунке 1.4.3 представлен результат выполнения программы для нахождения оптимального значения функции – график зависимости оптимального решения от номера итерации.

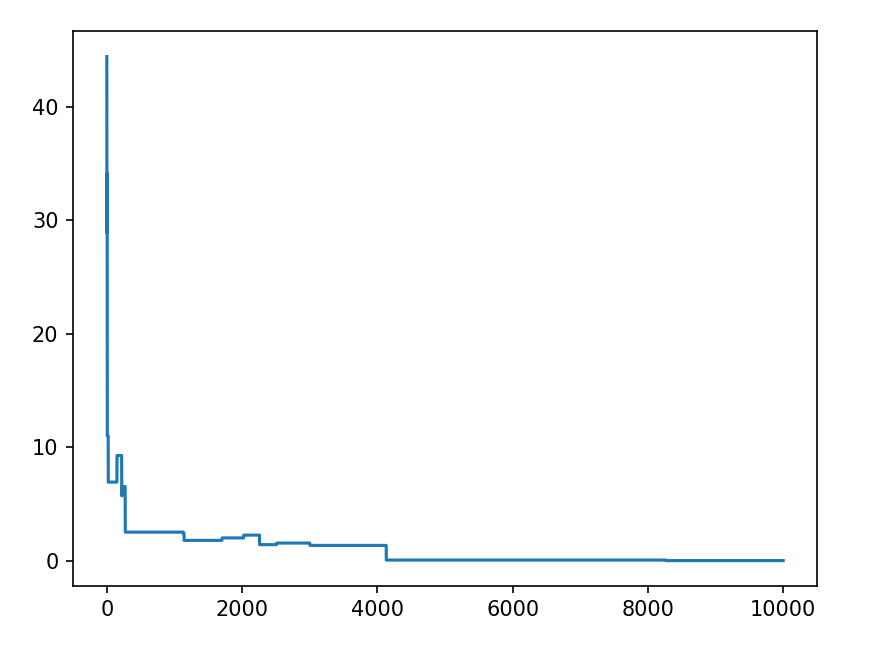


Рисунок 1.4.3 – График зависимости оптимального значения функции от номера итерации

# 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 информационных источников

1. 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.
2. Карпенко, А.П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко. – Москва: Издательство МГТУ им. Н. Э. Баумана, 2020. – 448 c.
3. Пряжников, А. А. Имитация отжига: простое объяснение метода и его применение [Электронный ресурс]. URL: <https://pryazhnikov-com.turbopages.org/pryazhnikov.com/s/notes/simulated-annealing/> (Дата обращения: 12.11.2024).
4. Сорокин, А. Б. Безусловная оптимизация : учебно-методическое пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк ; — Москва : МИРЭА - Российский технологический университет, 2020.

# приложения

Приложение А — Реализация метода имитации отжига в задаче коммивояжёра на языке Python.

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

### Приложение А

Реализация метода имитации отжига в задаче коммивояжёра на языке Python.

Листинг А.1 – Реализация метода имитации отжига

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import random

class Path:

    weights: np.ndarray | None = None

    graph: np.ndarray | None = None

    x: np.ndarray | None = None

    y: np.ndarray | None = None

    max\_weight = 50

    def \_\_init\_\_(self,

                 path: list[int] | None = None,

                 weights: np.ndarray | 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.weights is None:

            Path.weights = weights

        if Path.graph is None:

            Path.graph = graph

        if Path.x is None:

            Path.x = x

        if Path.y is None:

            Path.y = y

        if Path.weights is None or Path.graph is None:

            raise Exception('Ошибка с данными о графе')

    def create\_new\_path(self, current\_weight: int = 0):

        while not all([x in self.path for x in range(Path.graph.shape[0])]):

            possible\_weights = [

                i for i in range(len(Path.weights))

                if Path.weights[i] <= (Path.max\_weight - current\_weight) and (

                    i not in self.path)

            ]

            if len(possible\_weights) == 0:

                possible\_weights = [0]

            while True:

                next\_place = random.choice(possible\_weights)

                if next\_place == 0:

                    if self.path[-1] == 0:

                        continue

                    current\_weight = 0

                else:

                    next\_weight = Path.weights[next\_place]

                    current\_weight += next\_weight

                break

            self.path.append(next\_place)

        self.path.append(0)

Продолжение Листинга А.1

    def get\_replacements(self):

        possible\_replacements = [

            self.path[i] for i in range(1, len(self.path))

            if self.path[i - 1] == 0

        ]

        if len(possible\_replacements) == 1:

            possible\_replacements = self.path[1:-1]

        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()

        first, second = self.path[first\_index], self.path[second\_index]

        print(f'Замены: {first}, {second}')

        self.path = self.path[:first\_index + 1]

        self.path[-1] = second

        current\_weight = Path.weights[second]

        self.create\_new\_path(current\_weight=current\_weight)

    @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 = ''

        cum\_weight = 0

        for i, point in enumerate(self.path):

            result\_str += str(point)

            if point == 0:

                cum\_weight = 0

                result\_str += " (0)"

            else:

                cum\_weight += Path.weights[point]

                result\_str += f" ({cum\_weight})"

            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)

Продолжение Листинга А.1

            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])

            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('data.csv')

        df = pd.concat([pd.DataFrame([[0, 0, 0]], columns=df.columns), df],

                       ignore\_index=True)

        weights = df['weight'].to\_numpy()

        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)}')

        self.current\_path = Path(weights=weights, 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 > 0.0000001:

            if i == 0:

                self.best\_path.draw\_path()

            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())

Окончание Листинга А.1

            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 /= 2

            print('===============\n')

            history.append(self.best\_path.length)

        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('data.csv')

    solution.create\_graph()

    solution.solve()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Листинг А.2 – Генерация данных для задачи

import pandas as pd

import numpy as np

def generate(min\_coord: int, max\_coord: int, min\_weight: int, max\_weight: 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, ))

    weight = np.random.randint(min\_weight, max\_weight, size=(n\_places, ))

    return coord\_x, coord\_y, weight

def main():

    min\_coord, max\_coord, min\_weight, max\_weight, n = -50, 50, 1, 7, 20

    x, y, weight = generate(min\_coord, max\_coord, min\_weight, max\_weight, n)

    df = pd.DataFrame({'x': x, 'y': y, 'weight': weight})

    df.to\_csv('data.csv', index=False)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Листинг А.3 – Сгенерированные данные

x,y,weight

5,-14,2

34,-4,5

-47,-2,3

42,-24,3

-26,40,1

-7,13,4

-10,23,2

-29,45,6

12,9,6

2,-44,5

4,-42,4

22,14,6

-18,31,6

-11,-37,6

-46,-8,4

-4,9,5

-22,-44,6

-47,15,2

47,16,5

-9,-11,6

### Приложение Б

Код реализации метода имитации отжига в задаче оптимизации 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(

Окончание Листинга Б.1

                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))})'

            )

            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()