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

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА №2

по дисциплине
«Системный анализ данных в системах поддержки принятия решений»
Метод имитации отжига

Студент группы:ИКБО-04-22 __Кликушин В.И._
(Ф. И.О. студента)

Преподаватель __Железняк Л.М.__
(Ф.И.О. преподавателя)

Москва 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 МЕТОД ИМИТАЦИИ ОТЖИГА 4
1.1 Описание алгоритма 4
1.2 Постановка задачи 4
1.3 Задача коммивояжёра 6
1.3.1 Математическая модель 6
1.3.2 Ручной расчёт 9
1.4 Поиск глобального минимума 11
1.5 Программная реализация 13
ЗАКЛЮЧЕНИЕ 17
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 18
ПРИЛОЖЕНИЯ 19

ВВЕДЕНИЕ

Метод имитации отжига находит применение в широком спектре областей. В логистике и управлении транспортом он используется для расчета оптимальных маршрутов, планирования поставок и распределения транспортных средств. В производственных процессах и проектировании электроники метод помогает оптимизировать работу на производственных линиях и проектировать интегральные схемы, требующие высокой точности.

В энергетике имитация отжига применяется для решения задач оптимального распределения энергии, снижения эксплуатационных затрат и управления энергосистемами. Финансовые и инвестиционные компании также используют этот метод для оптимизации портфелей активов, распределения рисков и поиска эффективных инвестиционных стратегий. В сфере машинного обучения метод помогает в настройке параметров сложных моделей, таких как нейронные сети, позволяя избегать локальных минимумов в нелинейных пространствах.

Разработанный на основе физических процессов, этот метод моделирует принципы отжига металлов — процесса, при котором материал нагревают до высоких температур и затем медленно охлаждают, чтобы достичь состояния с минимальной энергией и более устойчивой структурой. В случае оптимизационной задачи метод имитации отжига стремится найти глобальный минимум целевой функции, исследуя различные состояния и постепенно «охлаждая» поисковое пространство, сужая диапазон возможных решений. Подобная стратегия позволяет методу преодолевать локальные минимумы и находить оптимальные решения там, где другие методы могли бы застрять.

Широкая применимость метода делает его мощным инструментом для решений, где требуется найти баланс между точностью и скоростью вычислений. Благодаря способности избегать локальных минимумов и более гибко исследовать пространство решений, метод имитации отжига успешно решает задачи, которые менее эффективны для традиционных алгоритмов.

1 МЕТОД ИМИТАЦИИ ОТЖИГА

Метод имитации отжига — это алгоритм оптимизации, вдохновленный процессом отжига в металлургии, при котором материал медленно охлаждается для нахождения минимальной энергии в более стабильной конфигурации.

1.1 Описание алгоритма

Принцип работы алгоритма:

1. Генерируется случайное решение, которое становится текущим.
2. В каждом шаге генерируется новое (рабочее) решение, полученное путём модификации текущего решения.
3. Если рабочее решение лучше, оно принимается и становится текущим. Если хуже, решение может быть принято с вероятностью, зависящей от разницы в качестве решения и температуры.
4. Температура постепенно снижается, уменьшая вероятность принятия худших решений, что приводит к сужению области поиска и стабилизации алгоритма.
5. Алгоритм завершает работу, когда температура становится ниже заданного порогового значения.

1.2 Постановка задачи

Цель работы: реализовать задачу коммивояжера и преобразование Коши методом имитации отжига для нахождения приближённого решения.

Изучить метод имитации отжига, выбрать предметную область для задачи Коммивояжёра и тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт двух итераций алгоритма для каждой задачи, разработать программные реализации классического алгоритма

имитации отжига для задачи коммивояжёра и отжига Коши для задачи минимизации функции.

Условие оригинальной задачи коммивояжёра: «Представим себе коммивояжёра, который должен посетить ряд городов, находящихся в разных местах. Его цель — начать и завершить путь в одном и том же городе, посетив каждый из остальных городов ровно один раз и минимизировав при этом общий пройденный путь (или затраты на поездку). Задача заключается в нахождении такого маршрута, который будет иметь минимальную длину среди всех возможных маршрутов, удовлетворяющих этим условиям».

Выбранная предметная область для задачи коммивояжёра: оптимизация маршрута подачи документов в высшие учебные заведения города Москвы.

Условие задачи коммивояжёра для выбранной предметной области: Абитуриент приезжает в Москву для подачи документов в несколько высших учебных заведений. Его цель — начать и завершить путь в отеле, где он остановился, посетить приёмные комиссии всех университетов ровно один раз и минимизировать при этом общее время и затраты на перемещение по городу. Необходимо найти оптимальный маршрут, который позволит посетить все выбранные университеты, минимизируя суммарное время, затраченное на дорогу, учитывая, что каждый университет можно посетить только один раз.

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 функция для оптимизации: функция Гольдшейна-Прайса (Формула 1.2.1).

$$f(x,y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + \\ +(2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] . \quad (1.2.1)$$

Глобальный минимум функции достигается в точке (0; -1) и равен 3. Функция рассматривается на области $-2 \leq x, y \leq 2$.

1.3 Задача коммивояжёра

1.3.1 Математическая модель

Особенность задачи заключается в том, что её модель представлена в виде ориентированного взвешенного полного графа. Между каждой парой вершин (университетов и отелем) существует направленное ребро с уникальным весом для каждого направления. Это означает, что путь от одной вершины к другой может иметь одну стоимость (время достижения), а обратный путь — совершенно другую. В отличие от неориентированного графа, где вес ребра между двумя вершинами одинаков независимо от направления, в задаче учитываются асимметрии, такие как односторонние улицы, разная интенсивность движения или различия в транспортных затратах по направлению туда и обратно.

Для ручного расчёта число университетов в задаче взято равным шести. С начальной стартовой точкой в отеле граф содержит семь вершин. Для семи вершин число возможных маршрутов может быть рассчитано по Формуле 1.3.1.1.

$$(n-1)! = (7-1)! = 6! = 1 * 2 * 3 * 4 * 5 * 6 = 720 \qquad (1.3.1.1)$$

Для полного взвешенного ориентированного графа, состоящего из семи вершин, число рёбер E, вес которых необходимо заполнить, рассчитано по Формуле 1.3.1.2.

$$E = n * (n-1) = 7 * 6 = 42 \qquad (1.3.1.2)$$

Местоположение приёмных комиссий университетов и отеля представлено реальными адресами объектов. Адрес объекта представлен названием города, наименованием улицы, номером дома (строения). Допустимо указание почтового индекса. В качестве стартовой точки выбран отель «Звёзды Арбата». Информация о пунктах маршрута отображена в Таблице 1.3.1.1.

Таблица 1.3.1.1 – Характеристики пунктов маршрута

Наименование пункта	**Сокращённое название**	**Адрес**
Отель «Звёзды Арбата»	Отель	Москва, Новый Арбат, 32
Московский государственный технический университет им. Н.Э. Баумана	МГТУ	Москва, 2-я Бауманская ул., д. 5, стр. 1
Московский физико-технический институт	МФТИ	Московская область, г. Долгопрудный, Институтский переулок, д. 9.
Национальный исследовательский ядерный университет «МИФИ»	МИФИ	Москва, Каширское шоссе, 31
Российская академия народного хозяйства и государственной службы при Президенте РФ	РАНХиГС	проспект Вернадского, 84с1, Москва, 119606
Университет науки и технологий МИСИС	МИСИС	Ленинский проспект, 2/4, Москва, 119049
МИРЭА – Российский технологический университет	МИРЭА	проспект Вернадского, 86с2, Москва

Для расчёта расстояний между вершинами в графе использовался сервис «Google Maps». Под расстоянием понимается время, необходимое на то, чтобы добраться из одной точки в другую по лучшему маршруту (преимущественно автомобиль). Данные получены 02.11.2024 в 14:15 часов. С учётом времени суток, времени года, погодных условий, пробок и аварий на дорогах, а также прочих факторов полученные сведения о времени на маршрут могут отличаться.

Рассчитанные длины рёбер сведены в Таблицу 1.3.1.2 с указанием сокращённых названий вершин, составляющих ребро.

Таблица 1.3.1.2 – Длины рёбер в графе

Ребро	Длина ребра
Отель - МГТУ	22
Отель - МФТИ	48
Отель - МИФИ	41
Отель - РАНХиГС	26
Отель - МИСИС	9
Отель - МИРЭА	26
МГТУ - Отель	18
МГТУ - МФТИ	46
МГТУ - МИФИ	39
МГТУ - РАНХиГС	42
МГТУ - МИСИС	17
МГТУ - МИРЭА	41
МФТИ - Отель	45
МФТИ - МГТУ	45
МФТИ - МИФИ	79
МФТИ - РАНХиГС	56
МФТИ - МИСИС	50
МФТИ - МИРЭА	56
МИФИ - Отель	34
МИФИ - МГТУ	30
МИФИ - МФТИ	67
МИФИ - РАНХиГС	43
МИФИ - МИСИС	28
МИФИ - МИРЭА	42
РАНХиГС - Отель	33
РАНХиГС - МГТУ	42
РАНХиГС - МФТИ	63
РАНХиГС - МИФИ	44
РАНХиГС - МИСИС	27
РАНХиГС - МИРЭА	9
МИСИС - Отель	14
МИСИС - МГТУ	18
МИСИС - МФТИ	50
МИСИС - МИФИ	33
МИСИС - РАНХиГС	30
МИСИС - МИРЭА	28
МИРЭА - Отель	31
МИРЭА - МГТУ	41
МИРЭА - МФТИ	67
МИРЭА - МИФИ	42
МИРЭА - РАНХиГС	10
МИРЭА - МИСИС	25

Единицей измерения для длины ребра является число минут на дорогу между пунктами. Граф, представляющий модель задачи с заданными рёбрами и вершинами представлен на Рисунке 1.3.1.1.

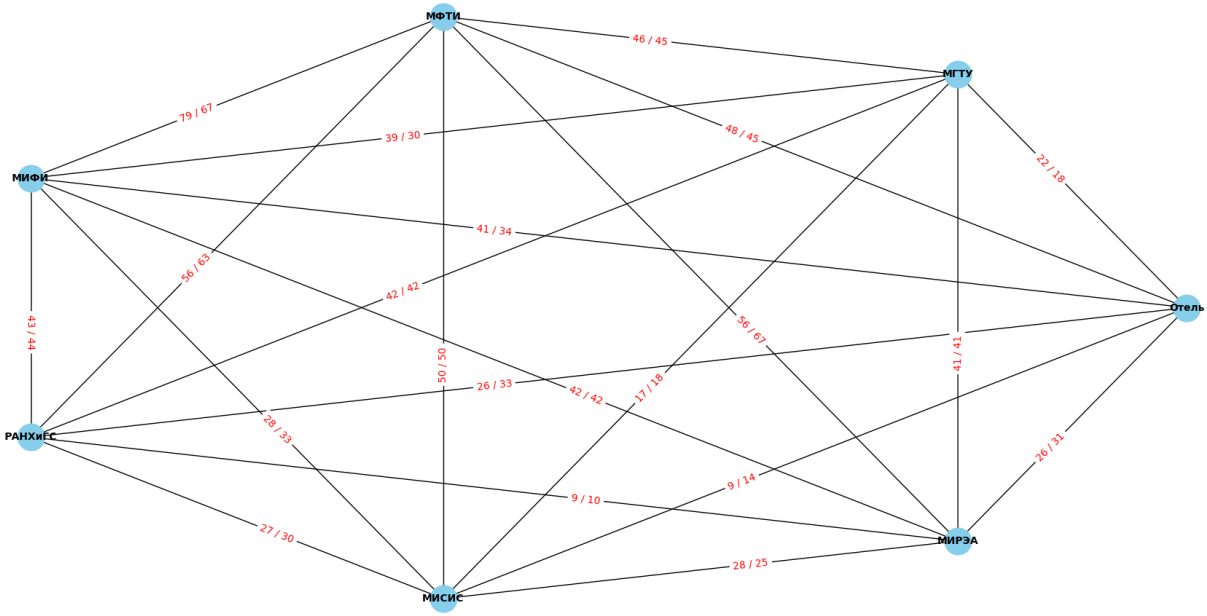

Рисунок 1.3.1.1 – Граф модели задачи

Веса рёбер на графе представлены двумя числами, записанными через косую черту.

1.3.2 Ручной расчёт

Случайным образом составлен первоначальный маршрут (Листинг 1.3.2.1):

Листинг 1.3.2.1 – Первоначальный маршрут

```
Отель -> РАНХиГС -> МИСИС -> МИРЭА -> МГТУ -> МФТИ -> МИФИ -> Отель
```

Построенный маршрут становится текущим. Длина полученного пути: S_0 = 26 + 27 + 28 + 41 + 46 + 79 + 34 = 281.

Перейдём к первой итерации алгоритма. За начальную температуру взята T_0 = 100 °C. Снижение температуры происходит по закону, представленном в Формуле 1.3.2.1.

$$T_{k+1} = 0.5 * T_k \quad (1.3.2.1)$$

Верхняя граница перехода на худшее решение рассчитывается по Формуле 1.3.2.2.

$$h(\Delta E, T) = \exp\left(\frac{-\Delta E}{T}\right) \quad (1.3.2.2)$$

Путём перестановки двух вершин в текущем маршруте получено рабочее решение (Листинг 1.3.2.2).

Листинг 1.3.2.2 – Рабочее решение на первой итерации

```
Отель -> РАНХиГС -> МГТУ -> МИРЭА -> МИСИС -> МФТИ -> МИФИ -> Отель
```

Произведён расчёт длины рабочего пути: $S_1 = 26 + 42 + 41 + 25 + 50 + 79 + 34 = 297$.

Длина рабочего пути оказалась больше длины текущего пути, поэтому проводится расчёт вероятности перехода к худшему решению (1.3.2.3).

$$h(\Delta E, T) = \exp\left(\frac{-\Delta E}{T}\right) = \exp\left(\frac{281-297}{100}\right) = 0.852143788966 \quad (1.3.2.3)$$

Вероятность, выданная псевдослучайным генератором чисел от 0 до 1, равна 0.768616527027, что меньше 0.852143788966. Поэтому рабочее решение принимается как лучшее и становится текущим.

В конце итерации температура уменьшается в два раза по сравнению с изначальной: $T_1 = T_0 / 2 = 100 / 2 = 50$.

На второй итерации текущее решение модифицировано и получено новое рабочее решение (Листинг 1.3.2.3).

Листинг 1.3.2.3 – Рабочее решение на второй итерации

```
Отель -> РАНХиГС -> МИФИ -> МИРЭА -> МИСИС -> МФТИ -> МГТУ -> Отель
```

Произведён расчёт длины рабочего пути: $S_2 = 26 + 44 + 42 + 25 + 50 + 45 + 18 = 250$.

Длина рабочего пути меньше длины текущего пути, поэтому рабочее решение сразу принимается и становится текущим.

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

1.4 Поиск глобального минимума

В качестве функции для поиска глобального минимума выбрана функция Гольдшейна-Прайса (Формула 1.2.1).

График функции представлен на Рисунке 1.4.1.

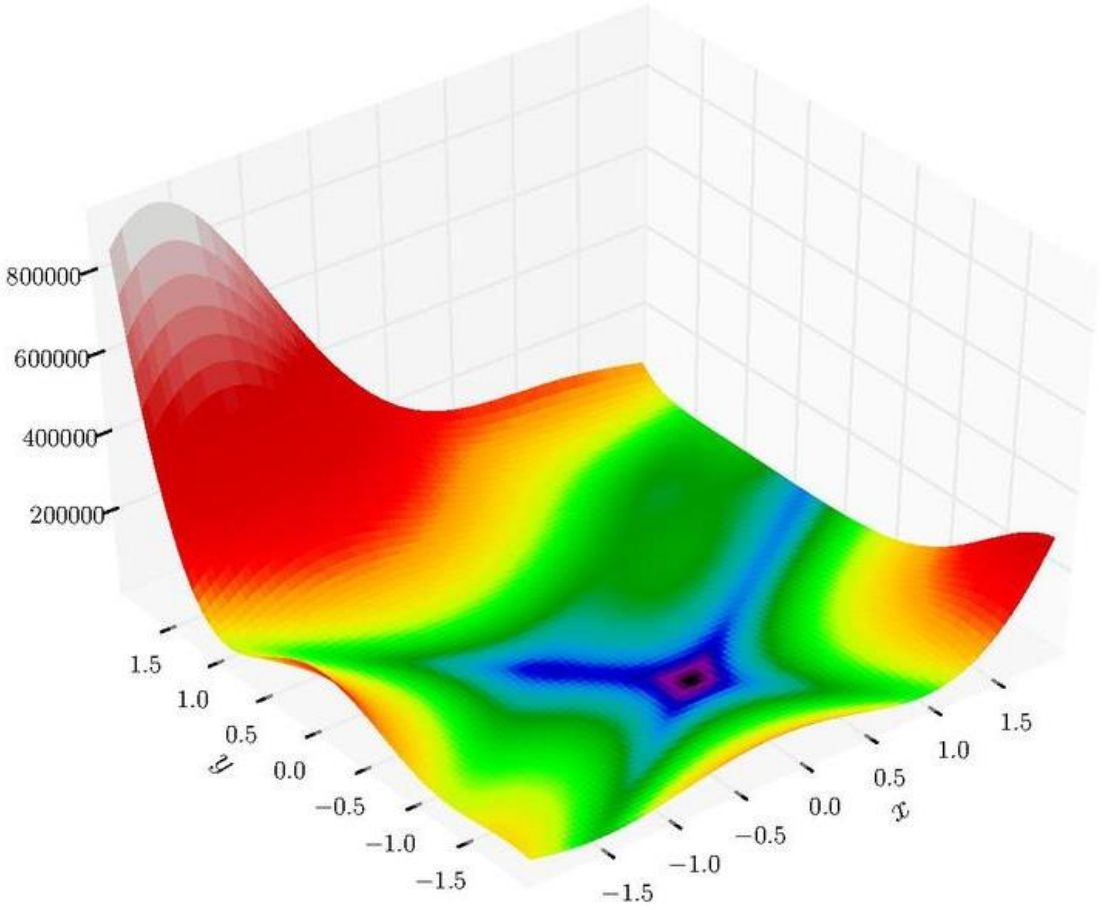

Рисунок 1.4.1 – График функции Гольдшейна-Прайса

Глобальный минимум функции достигается в точке (0; -1) и равен 3. Функция рассматривается на области $-2 \leq x, y \leq 2$.

Снижение температуры происходит по закону, представленному в Формуле 1.4.1.

$$T_k = \frac{T_0}{k^{\frac{1}{D}}} \tag{1.4.1}$$

Начальная температура T_0 = 200 °C.

Текущее решение на каждой итерации генерируется с использованием распределения Коши (Формула 1.4.2), где D = 2, так как задача рассматривается в двумерном пространстве.

$$g(x', x, T) = \frac{1}{\pi^D} \prod_{i=1}^{D} \frac{T}{|x' - x|^2 + T^2} \quad (1.4.2)$$

В начале итерации уменьшается температура (Формула 1.4.3):

$$T_1 = \frac{T_0}{k^{\frac{1}{D}}} = \frac{200}{1^{\frac{1}{2}}} = 200 \quad (1.4.3)$$

Случайное решение, которое становится текущим: (1.47, -0.06). Значение функции в этой точке равно 123.72.

Рабочее решение на первой итерации: (1.57, 1.19). Значение функции в этой точке равно 1618.40.

Поскольку рабочее решение оказалось хуже текущего, то проводится расчёт вероятности перехода к этому решению (Формула 1.4.4).

$$h(\Delta E, T_0) = e^{-\frac{\Delta f}{T_0}} = e^{-\frac{1618.40 - 123.72}{200}} \approx 0.000567 \quad (1.4.4)$$

Вероятность, выданная псевдослучайным генератором чисел от 0 до 1, равна 0.08837, что больше 0.000567. Поэтому рабочее решение отбрасывается.

Перед второй итераций температура вновь уменьшается в соответствии с законом охлаждения Коши (Формула 1.4.5).

$$T_2 = \frac{T_1}{k^{\frac{1}{D}}} = \frac{200}{2^{\frac{1}{2}}} = 141.42 \quad (1.4.5)$$

Рабочее решение на второй итерации: (1.36, -0.22). Значение функции в этой точке: 344.59.

Рабочее решение оказалось хуже текущего, поэтому проводится расчёт вероятности перехода к этому решению (Формула 1.4.6).

$$h(\Delta E, T_0) = e^{-\frac{\Delta f}{T_0}} = e^{-\frac{344.59-123.72}{141.42}} \approx 0.209758 \quad (1.4.6)$$

Вероятность, выданная псевдослучайным генератором чисел от 0 до 1, равна 0.03872, что меньше 0.209758. Поэтому рабочее решение принимается и становится текущим.

По завершению второй итерации текущим решением является точка (1.36, -0.22). Значение функции в этой точке: 344.59.

1.5 Программная реализация

Разработан класс Vertex, представляющий вершину графа. Экземпляр класса содержит атрибуты name — наименование вершины, short_name — сокращённое название вершины, address — физический адрес объекта.

Реализован класс Graph, который используется для представления модели задачи. Экземпляр класса Graph хранит атрибуты vertices — список экземпляров класса Vertex и adjacency_matrix — двумерный список, представляющий матрицу весов. Класс содержит минимальный набор операций для работы с графом: добавление ребра, добавление вершины, удаление ребра, удаление вершины, отрисовка графа, заполнение матрицы весов из файла. В качестве дополнительной возможности автоматизирован процесс сбора информации о «стоимости» маршрутов средствами библиотеки selenium. Для модуля программы, реализующий метод имитации отжига написан отдельный класс.

Код алгоритма имитации отжига для задачи коммивояжера представлен в Приложении А.

Процесс автоматического заполнения матрицы весов представлен на Рисунке 1.5.1. Функция может не отработать и выбросить исключение из-за плохого интернет-соединения.

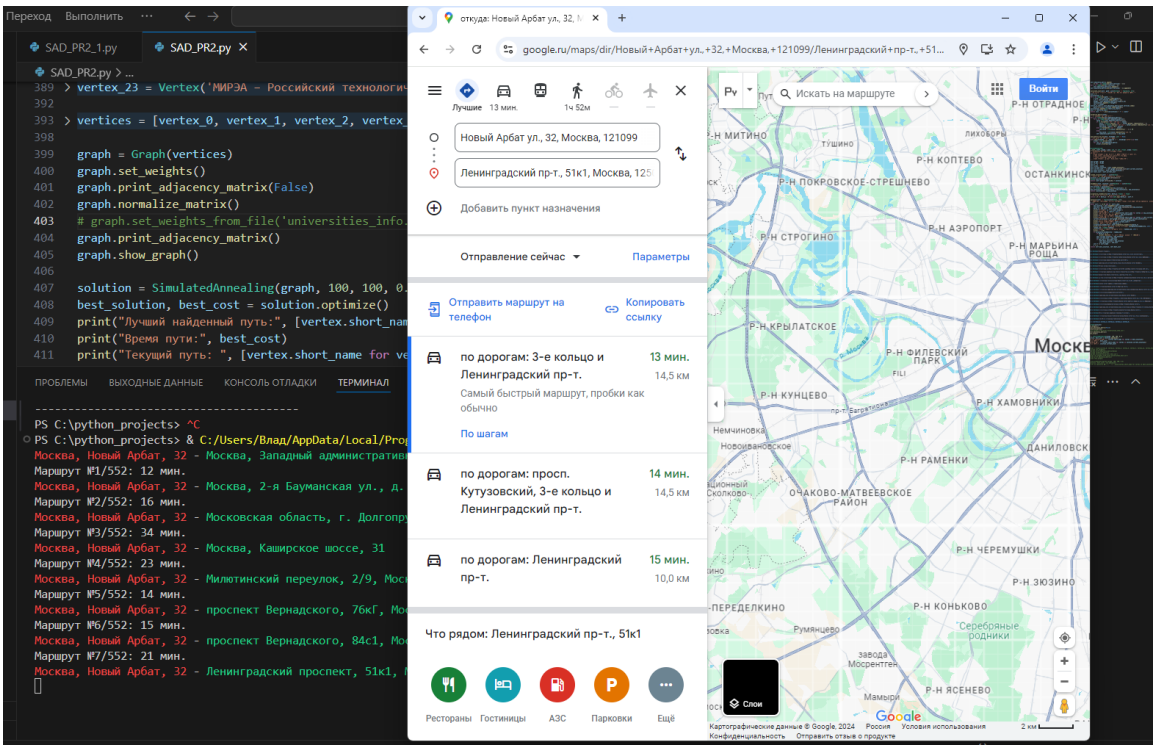

Рисунок 1.5.1 – Автоматическое заполнение матрицы весов

Выведенная матрица весов отображена на Рисунке 1.5.2.

Отель	МГУ	МГТУ	МФТИ	МИФИ	ВШЭ	МГИМО	РАНХиГС	ФУ	МГМУ	РЭУ	МИСИС	РУДН	РНИМУ	МАИ	МЭИ	МПГУ	РГУ	МПГУ	НИУ МГСУ	МГЛУ	ВАВТ	РХТУ	МИРЭА
0	12	16	34	24	13	17	20	14	9	11	7	24	22	28	20	7	18	23	24	8	7	11	19
16	0	23	39	25	24	11	14	19	12	17	15	19	16	29	24	18	9	17	30	15	10	21	13
13	23	0	33	22	12	24	29	17	17	12	15	29	30	32	6	12	24	32	19	15	19	13	31
32	39	33	0	53	36	42	39	23	37	39	36	39	38	28	37	30	42	40	24	35	33	27	39
23	24	21	45	0	27	23	28	28	21	15	18	26	27	43	20	24	19	30	32	20	18	27	27
11	23	10	35	27	0	24	29	16	14	12	12	31	31	31	16	10	25	31	20	11	19	11	30
22	10	27	43	27	28	0	5	25	14	20	17	8	7	31	26	22	9	8	36	18	15	25	3
23	12	31	45	29	32	5	0	27	19	25	21	7	6	29	31	26	14	4	37	22	17	26	9
10	18	13	27	29	16	22	25	0	16	19	14	28	29	20	17	9	22	29	19	14	12	9	27
10	13	17	39	23	17	14	19	19	0	10	6	23	21	34	20	10	15	23	27	6	10	14	21
10	16	11	39	19	14	19	24	19	9	0	6	23	23	33	14	11	16	26	23	7	12	15	22
10	13	11	38	20	14	15	21	19	9	5	0	21	20	33	14	11	14	23	23	7	9	15	20
27	17	31	48	28	33	8	8	29	21	24	22	0	24	30	32	28	13	8	41	24	21	31	6
27	16	31	49	28	34	7	7	29	20	25	23	25	0	29	33	27	14	5	40	23	19	31	5
27	31	30	37	45	31	32	29	18	31	35	31	30	29	0	34	25	35	31	36	30	28	25	29
18	25	6	36	24	16	25	30	20	20	16	18	31	30	35	0	16	25	32	20	18	20	17	29
2	13	14	33	24	11	16	21	13	7	10	5	25	23	28	19	0	18	24	22	6	9	9	23
15	9	18	39	20	21	10	15	19	10	13	10	14	14	33	20	16	0	18	30	12	10	19	14
27	15	34	48	32	35	8	3	30	21	28	24	9	6	30	34	29	16	0	41	25	20	29	13
20	30	18	23	38	20	33	39	21	25	25	23	40	40	32	21	18	33	41	0	24	24	17	38
8	12	13	36	22	14	13	18	16	4	7	3	21	20	30	17	7	14	20	24	0	12	11	19
13	8	20	34	22	20	12	17	15	12	15	11	20	19	30	22	13	14	20	26	11	0	16	16
8	20	12	26	31	10	23	27	9	14	16	12	31	29	24	17	7	25	30	18	12	14	0	26
24	13	29	46	28	31	3	10	27	17	23	20	5	4	28	29	24	12	14	38	20	18	28	0

Рисунок 1.5.2 – Матрица весов для графа из 24 вершин

Матрица весов дополнительно нормализуется после автоматического считывания данных. Отбрасываются окончания «мин» и «ч», соответствующие минутам и часам в пути. Часы при необходимости переводятся в минуты. Полученное число, представленное в строке, приводится к целому типу данных.

Создание и инициализация графа, запуск алгоритма имитации отжига представлены на Рисунке 1.5.3.

```
graph = Graph(vertices)
# graph.set_weights()
# graph.print_adjacency_matrix(False)
# graph.normalize_matrix()
graph.set_weights_from_file('universities_info.csv')
graph.print_adjacency_matrix(False)
graph.show_graph()

solution = SimulatedAnnealing(graph, 100, 100, 0.5)
best_solution, best_cost = solution.optimize()
print("Лучший найденный путь:", ' -> '.join([vertex.short_name for vertex in best_solution]))
print("Время пути:", best_cost)
print("Текущий путь: ", ' -> '.join([vertex.short_name for vertex in solution.current_solution]))
print("Время пути:", solution.current_cost)
```

Рисунок 1.5.3 – Инициализация графа, запуск алгоритма имитации отжига

Первые итерации работы алгоритма представлены на Рисунке 1.5.4.

```
Итерация: 1/100
Температура: 100.000000000000
Рабочий путь: Отель -> МИФИ -> РАНХиГС -> МГУ -> МИРЭА -> МГТУ -> МПГУ -> МГМУ -> МГИМО -> РХТУ -> МАИ -> РГУ -> ФУ -> МФТИ -> МГЮА -> РЭУ -> МИСИС -> ВАВТ -> МГЛУ -> ВШЭ -> РУДН -> НИУ МГСУ -> РНИМУ -> МЭИ
Расчёт длины рабочего пути: 24 + 28 + 12 + 13 + 29 + 32 + 21 + 14 + 25 + 24 + 35 + 19 + 27 + 30 + 10 + 6 + 9 + 11 + 14 + 31 + 41 + 40 + 33 + 18 = 546
Длина рабочего пути: 546
Текущий путь: Отель -> МИФИ -> РАНХиГС -> МГУ -> МИРЭА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> МАИ -> РГУ -> ФУ -> МФТИ -> МГЮА -> РЭУ -> МИСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> РНИМУ -> МЭИ
Расчёт длины текущего пути: 24 + 28 + 12 + 13 + 29 + 32 + 9 + 8 + 25 + 24 + 35 + 19 + 27 + 30 + 10 + 6 + 9 + 11 + 14 + 14 + 27 + 40 + 33 + 18 = 497
Длина текущего пути: 497
Разность энергий: 49
Вероятность перехода в новое состояние: 0.612626394184
Сгенерированное случайное число: 0.951312463618
----------------------------------------
Итерация: 2/100
Температура: 50.000000000000
Рабочий путь: Отель -> МИФИ -> РАНХиГС -> МГУ -> МИРЭА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> РНИМУ -> РГУ -> ФУ -> МФТИ -> МГЮА -> РЭУ -> МИСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> МАИ -> МЭИ
Расчёт длины рабочего пути: 24 + 28 + 12 + 13 + 29 + 32 + 9 + 8 + 25 + 29 + 14 + 19 + 27 + 30 + 10 + 6 + 9 + 11 + 14 + 14 + 27 + 32 + 34 + 18 = 474
Длина рабочего пути: 474
Текущий путь: Отель -> МИФИ -> РАНХиГС -> МГУ -> МИРЭА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> МАИ -> РГУ -> ФУ -> МФТИ -> МГЮА -> РЭУ -> МИСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> РНИМУ -> МЭИ
Расчёт длины текущего пути: 24 + 28 + 12 + 13 + 29 + 32 + 9 + 8 + 25 + 24 + 35 + 19 + 27 + 30 + 10 + 6 + 9 + 11 + 14 + 14 + 27 + 40 + 33 + 18 = 497
Длина текущего пути: 497
Разность энергий: -23
Вероятность перехода в новое состояние: 1.000000000000
Сгенерированное случайное число: 0.096400309898
----------------------------------------
Итерация: 3/100
Температура: 25.000000000000
Рабочий путь: Отель -> МИФИ -> РУДН -> МГУ -> МИРЭА -> МГТУ -> МПГУ -> РАНХиГС -> МГИМО -> РХТУ -> РНИМУ -> РГУ -> ФУ -> МФТИ -> МГЮА -> РЭУ -> МИСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> МАИ -> МЭИ
Расчёт длины рабочего пути: 24 + 26 + 17 + 13 + 29 + 32 + 3 + 5 + 25 + 29 + 14 + 19 + 27 + 30 + 10 + 6 + 9 + 11 + 14 + 14 + 27 + 32 + 34 + 18 = 468
Длина рабочего пути: 468
Текущий путь: Отель -> МИФИ -> РАНХиГС -> МГУ -> МИРЭА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> РНИМУ -> РГУ -> ФУ -> МФТИ -> МГЮА -> РЭУ -> МИСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> МАИ -> МЭИ
Расчёт длины текущего пути: 24 + 28 + 12 + 13 + 29 + 32 + 9 + 8 + 25 + 29 + 14 + 19 + 27 + 30 + 10 + 6 + 9 + 11 + 14 + 14 + 27 + 32 + 34 + 18 = 474
Длина текущего пути: 474
Разность энергий: -6
Вероятность перехода в новое состояние: 1.000000000000
Сгенерированное случайное число: 0.396250539086
----------------------------------------
```

Рисунок 1.5.4 – Первые итерации алгоритма отжига для задачи коммивояжера

Результат работы алгоритма отжига представлен на Рисунке 1.4.5.

```
----------------------------------------
Итерация: 39/100
Температура: 0.000000000364
Рабочий путь: Отель -> РНИМУ -> РУДН -> МГУ -> РЭУ -> РАНХиГС -> МПГУ -> МИРЭА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МФТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЮА -> МГЛУ -> МИСИС -> МИФИ -> НИУ МГСУ -> МЭИ -> МГМУ
Расчёт длины рабочего пути: 22 + 25 + 17 + 17 + 24 + 4 + 13 + 3 + 9 + 10 + 16 + 26 + 23 + 16 + 10 + 32 + 25 + 6 + 3 + 20 + 32 + 21 + 20 + 10 = 404
Длина рабочего пути: 404
Текущий путь: Отель -> РНИМУ -> РУДН -> МГУ -> РЭУ -> РАНХиГС -> МПГУ -> МИРЭА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МФТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЮА -> МГЛУ -> МИСИС -> МГМУ -> НИУ МГСУ -> МЭИ -> МИФИ
Расчёт длины текущего пути: 22 + 25 + 17 + 17 + 24 + 4 + 13 + 3 + 9 + 10 + 16 + 26 + 23 + 16 + 10 + 32 + 25 + 6 + 3 + 9 + 27 + 21 + 24 + 23 = 405
Длина текущего пути: 405
Разность энергий: -1
Вероятность перехода в новое состояние: 1.000000000000
Сгенерированное случайное число: 0.652316410145
----------------------------------------
Итерация: 40/100
Температура: 0.000000000182
Рабочий путь: Отель -> РНИМУ -> РУДН -> МГУ -> РЭУ -> РАНХиГС -> МПГУ -> МИРЭА -> МГИМО -> МИСИС -> ВАВТ -> РХТУ -> МФТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЮА -> МГЛУ -> РГУ -> МИФИ -> НИУ МГСУ -> МЭИ -> МГМУ
Расчёт длины рабочего пути: 22 + 25 + 17 + 17 + 24 + 4 + 13 + 3 + 17 + 9 + 16 + 26 + 23 + 16 + 10 + 32 + 25 + 6 + 14 + 20 + 32 + 21 + 20 + 10 = 422
Длина рабочего пути: 422
Текущий путь: Отель -> РНИМУ -> РУДН -> МГУ -> РЭУ -> РАНХиГС -> МПГУ -> МИРЭА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МФТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЮА -> МГЛУ -> МИСИС -> МИФИ -> НИУ МГСУ -> МЭИ -> МГМУ
Расчёт длины текущего пути: 22 + 25 + 17 + 17 + 24 + 4 + 13 + 3 + 9 + 10 + 16 + 26 + 23 + 16 + 10 + 32 + 25 + 6 + 3 + 20 + 32 + 21 + 20 + 10 = 404
Длина текущего пути: 404
Разность энергий: 18
Вероятность перехода в новое состояние: 0.000000000000
Сгенерированное случайное число: 0.332406617086
----------------------------------------
Лучший найденный путь: Отель -> РНИМУ -> РУДН -> МГУ -> РЭУ -> РАНХиГС -> МПГУ -> МИРЭА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МФТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЮА -> МГЛУ -> МИСИС -> МИФИ -> НИУ МГСУ -> МЭИ -> МГМУ
Время пути: 404
Текущий путь:  Отель -> РНИМУ -> РУДН -> МГУ -> РЭУ -> РАНХиГС -> МПГУ -> МИРЭА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МФТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЮА -> МГЛУ -> МИСИС -> МИФИ -> НИУ МГСУ -> МЭИ -> МГМУ
Время пути: 404
PS C:\python_projects>
```

Рисунок 1.5.5 – Результат работы алгоритма имитации отжига для задачи коммивояжера

Код алгоритма имитации отжига Коши для задачи поиска глобального минимума функции представлен в Приложении Б.

Результат поиска глобального минимума функции Гольдшейна-Прайса методом имитации отжига Коши представлен на Рисунке 1.5.6.

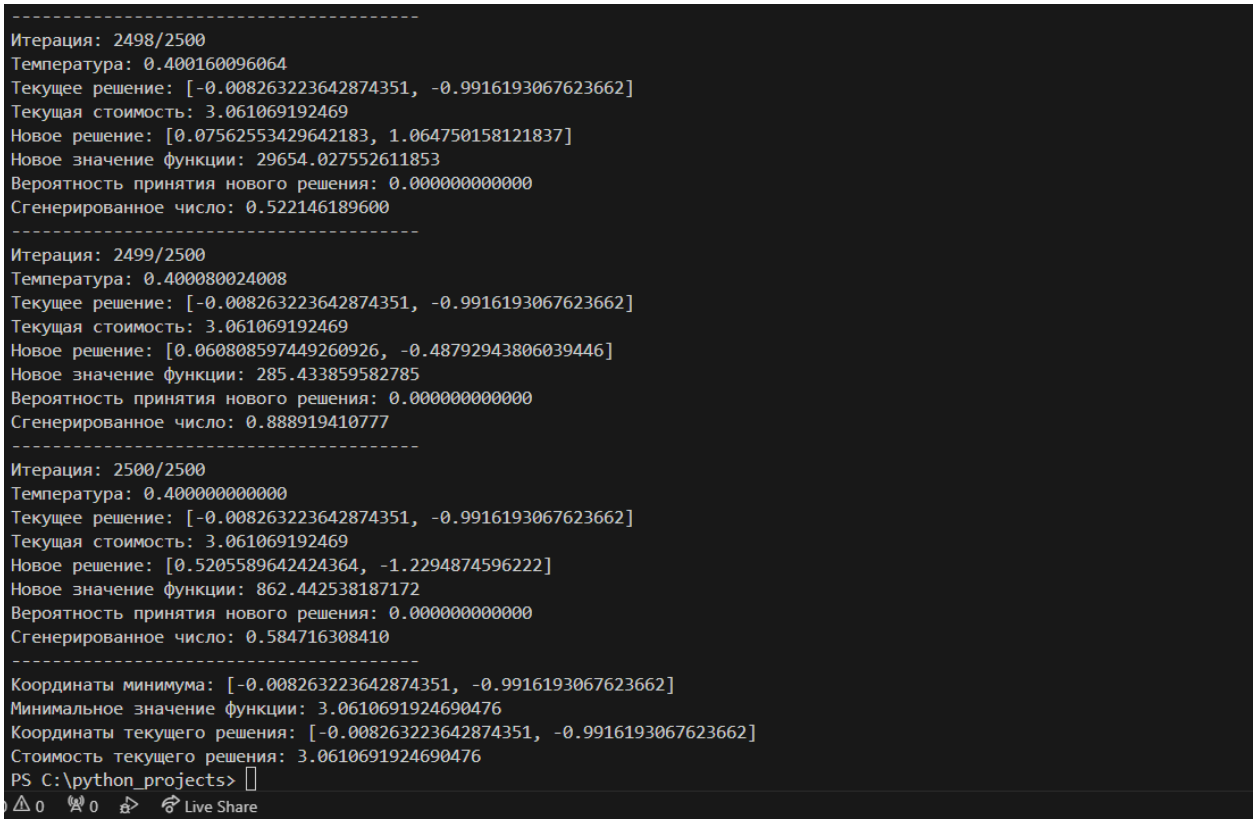

Рисунок 1.5.6 – Результат работы метода отжига Коши для поиска глобального минимума функции

Параметры запуска алгоритма отжига Коши представлены на Рисунке 1.5.7.

```
122  # Функция Гольдштейна-Прайса
123  def goldstein_price(x: float, y: float) -> float:
124      term1 = (1 + (x + y + 1)**2 * (19 - 14*x + 3*x**2 - 14*y + 6*x*y + 3*y**2))
125      term2 = (30 + (2*x - 3*y)**2 * (18 - 32*x +
126                  12*x**2 + 48*y - 36*x*y + 27*y**2))
127      return term1 * term2
128
129
130  bounds = [(-2, 2), (-2, 2)]
131  solution = SimulatedAnnealing(goldstein_price, bounds, k_max=2500, T0=20)
132  result = solution.optimize()
133  print("Координаты минимума:", result[0])
134  print("Минимальное значение функции:", result[1])
135  print("Координаты текущего решения:", solution.current_solution)
136  print("Стоимость текущего решения:", solution.current_cost)
137
```

Рисунок 1.5.7 – Параметры для отжига Коши

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы 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 задачи коммивояжера с обходом университетов и минимизации функции Гольдшейна-Прайса.

Основным преимуществом метода имитации отжига является его способность находить глобальные минимумы функций, даже если пространство решений содержит несколько локальных минимумов, что делает его полезным в сложных задачах оптимизации. Однако алгоритм зависит от корректного выбора параметров, таких как начальная температура и скорость её снижения, так как это влияет на вероятность принятия худших решений и, соответственно, на качество и скорость сходимости.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие / А. Б. Сорокин – Москва: Московский технологический университет (МИРЭА), 2019.
2. 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: 02.11.2024).
3. Google Maps. [Электронный ресурс]. URL: https://maps.google.com (Дата обращения: 06.11.2024).
4. Сорокин, А. Б. Безусловная оптимизация : учебно-методическое пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк ; Министерство науки и высшего образования Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА - Российский технологический университет» (РТУ МИРЭА). — Москва : МИРЭА - Российский технологический университет, 2020.

ПРИЛОЖЕНИЯ

Приложение А — Код реализации метода имитации отжига для задачи коммивояжёра

Приложение Б — Код реализации метода имитации отжига Коши для задачи поиска глобального минимума функции

Приложение А

Код реализации метода имитации отжига для задачи коммивояжёра

Листинг А – Реализация метода имитации отжига для задачи коммивояжёра

```
import random
import math
import certifi
import time
import csv
import re
import time
import functools
import matplotlib.pyplot as plt
import networkx as nx
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from dataclasses import dataclass, field
from tabulate import tabulate
from typing import List, Tuple

chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--ignore-certificate-errors')
chrome_options.add_argument('--ignore-ssl-errors')
chrome_options.add_argument(f"--ssl-certificates-path={certifi.where()}")
chrome_options.add_experimental_option(
    "excludeSwitches", ['enable-automation', 'enable-logging'])

@dataclass
class Vertex:
    '''Класс для представления узла графа, который включает название,
сокращенное имя и адрес.'''
    name: str
    short_name: str = field(compare=False)
    address: str
    is_visited: bool = field(default=False, repr=False,
                             compare=False, init=False)

    def __str__(self) -> str:
        return self.short_name

class Graph:
    def __init__(self, vertices: List[Vertex]):
        '''Инициализирует граф с заданными узлами и матрицей смежности. '''
        self.vertices = vertices
        self.adjacency_matrix: List[List[int]] = [[1 if i != j else 0 for j in
range(
            len(vertices))] for i in range(len(vertices))]

    @property
    def vertices(self) -> List[Vertex]:
        return self.__vertices

    @vertices.setter
    def vertices(self, vertices: List[Vertex]) -> None:
        self.__vertices = vertices

    @property
```

Продолжение Листинга А

```
    def adjacency_matrix(self) -> List[List[int]]:
        return self.__adjacency_matrix

    @adjacency_matrix.setter
    def adjacency_matrix(self, adjacency_matrix: List[List[int]]) -> None:
        self.__adjacency_matrix = adjacency_matrix

    def show_graph(self):
        '''Рисует граф, используя текущую матрицу весов.'''
        G = nx.Graph()
        for i, row in enumerate(self.adjacency_matrix):
            for j, weight in enumerate(row):
                if i < j and (weight != 0 or self.adjacency_matrix[j][i] != 0):
                    G.add_edge(self.vertices[i].short_name,
self.vertices[j].short_name,
                               weight_ab=weight,
weight_ba=self.adjacency_matrix[j][i])
        pos = nx.circular_layout(G)
        nx.draw(G, pos, with_labels=True, node_size=700, node_color="skyblue",
font_size=10, font_weight="bold")
        edge_labels = {}
        for u, v, d in G.edges(data=True):
            edge_labels[(u, v)] = f"{d['weight_ab']} / {d['weight_ba']}"
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_color="red", label_pos=0.6)
        plt.show()

    def print_adjacency_matrix(self, show_routes = True) -> None:
        '''Выводит матрицу смежности в консоль.'''
        column_names = [vertex.short_name for vertex in self.vertices]
        table = tabulate(self.adjacency_matrix, headers=column_names,
                         tablefmt='simple', maxcolwidths=3)
        print(table)
        if show_routes:
            for i, vertex_i in enumerate(self.vertices):
                for j, vertex_j in enumerate(self.vertices):
                    if i != j and self.adjacency_matrix[i][j] > 0:
                        print(f'Длина ребра от {vertex_i.short_name} до
{vertex_j.short_name}: {self.adjacency_matrix[i][j]}')

    @staticmethod
    def timer(func):
        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            start = time.perf_counter()
            val = func(*args, **kwargs)
            end = time.perf_counter()
            work_time = end - start
            print(f'Время выполнения {func.__name__}: {round(work_time, 4)}
сек.')
            return val
        return wrapper

    @timer
    def set_weights(self, gui: bool = True) -> None:
        '''Заполняет матрицу смежности временем достижения между узлами,
используя Google Maps.'''
        if not gui:
            chrome_options.add_argument('--headless')
        with webdriver.Chrome(options=chrome_options) as browser:
            url = 'https://www.google.ru/maps'
```

Продолжение Листинга А

```
            browser.get(url)
            route = WebDriverWait(browser, 3).until(
                EC.element_to_be_clickable((By.CLASS_NAME, 'hArJGc')))
            route.click()
            time.sleep(0.5)
            k = 1
            for vertex_i in self.vertices:
                for vertex_j in self.vertices:
                    if vertex_i != vertex_j:
                        print(f'\033[91m{vertex_i.address}\033[0m -
\033[92m{vertex_j.address}\033[0m')
                        departure_point = WebDriverWait(browser, 10).until(
                            EC.element_to_be_clickable((By.CLASS_NAME, 'tactile-
searchbox-input')))
                        departure_point.clear()
                        departure_point.send_keys(vertex_i.address)
                        destination_point = WebDriverWait(browser, 10).until(
                            EC.element_to_be_clickable((By.CSS_SELECTOR, '[aria-
controls="sbsg51"]')))
                        destination_point.clear()
                        destination_point.send_keys(vertex_j.address)
                        destination_point.send_keys(Keys.ENTER)
                        result = WebDriverWait(browser, 10).until(
                            EC.element_to_be_clickable((By.CLASS_NAME,
'Fk3sm'))).text
                        print(f'Маршрут №{
                              k}/{len(self.vertices) ** 2 - len(self.vertices)}:
{result}')
                        self.adjacency_matrix[self.vertices.index(
                            vertex_i)][self.vertices.index(vertex_j)] = result
                        k += 1

    def set_weights_from_file(self, filename: str) -> None:
        '''Устанавливает веса из файла с матрицей смежности.'''
        with open(filename, 'r', encoding='utf-8') as file:
            reader = csv.reader(file)
            self.adjacency_matrix = [[int(i) for i in row] for row in reader]

    def delete_vertex(self, vertex: Vertex) -> None:
        '''Удаляет узел и соответствующие ребра из графа.'''
        try:
            vertex_index = self.vertices.index(vertex)
        except ValueError:
            return
        self.vertices.remove(vertex)
        self.adjacency_matrix = [
            row[:vertex_index] + row[vertex_index+1:] for row in
self.adjacency_matrix
        ]
        self.adjacency_matrix = [
            row for i, row in enumerate(self.adjacency_matrix) if i !=
vertex_index
        ]

    def delete_edge(self, first_vertex: Vertex, second_vertex: Vertex) -> None:
        '''Удаляет ребро между двумя узлами.'''
        try:
            first_index = self.vertices.index(first_vertex)
            second_index = self.vertices.index(second_vertex)
        except ValueError:
            return
```

Продолжение Листинга А

```
        self.adjacency_matrix[first_index][second_index] = 0
        self.adjacency_matrix[second_index][first_index] = 0

    def set_edge(self, first_vertex: Vertex, second_vertex: Vertex, value: int)
-> None:
        '''Устанавливает вес ребра между двумя узлами.'''
        try:
            first_vertex_index = self.vertices.index(first_vertex)
            second_vertex_index = self.vertices.index(second_vertex)
        except ValueError:
            return
        self.adjacency_matrix[first_vertex_index][second_vertex_index] = value
        self.adjacency_matrix[second_vertex_index][first_vertex_index] = value

    def add_vertex(self, vertex: Vertex) -> None:
        '''Добавляет новый узел и обновляет матрицу смежности.'''
        self.vertices.append(vertex)
        self.adjacency_matrix.append(
            [1 for _ in range(len(self.vertices) - 1)])
        for i in range(len(self.vertices) - 1):
            self.adjacency_matrix[i].append(1)
        self.adjacency_matrix[len(self.vertices) - 1].append(0)

    def calculate_cost(self, path: List[Vertex]) -> Tuple[int, str]:
        '''Вычисляет стоимость (длину) маршрута для заданного пути.'''
        cost = 0
        calculations = []
        for i in range(len(path) - 1):
            v_from = self.vertices.index(path[i])
            v_to = self.vertices.index(path[i + 1])
            weight = self.adjacency_matrix[v_from][v_to]
            calculations.append(str(weight))
            cost += weight
        return_to_start = self.adjacency_matrix[self.vertices.index(
            path[-1])][self.vertices.index(path[0])]
        calculations.append(str(return_to_start))
        cost += return_to_start
        return cost, " + ".join(calculations) + f" = {cost}"

    def normalize_matrix(self):
        '''Нормализует матрицу весов'''
        for i in range(len(self.adjacency_matrix)):
            for j in range(len(self.adjacency_matrix)):
                value = str(self.adjacency_matrix[i][j])
                if re.fullmatch(r'\d+ ч \d+ мин.', value):
                    hours = int(re.search(r'\d+ ч',
value).group().removesuffix('ч'))
                    minutes = int(re.search(r'\d+ мин.',
value).group().removesuffix('мин.'))
                    new_value = hours * 60 + minutes
                elif re.fullmatch(r'\d ч.', value):
                    new_value = int(value.removesuffix('ч.')) * 60
                else:
                    new_value = int(value.removesuffix('мин.'))
                self.adjacency_matrix[i][j] = new_value

    def save_matrix_to_csv(self, filename: str) -> None:
        '''Сохраняет матрицу весов в файл.'''
        with open(filename, 'w', newline = '', encoding='utf-8') as file:
            writer = csv.writer(file)
            for row in self.adjacency_matrix:
```

Продолжение Листинга А

```
                writer.writerow(row)

class SimulatedAnnealing:
    def __init__(self, graph, k_max: int, T: int | float, alpha: float):
        '''Инициализирует алгоритм имитации отжига.
        Параметры:
            graph (Graph): Граф, на котором будет выполняться алгоритм.
            k_max (int): Максимальное количество итераций.
            T (int): Начальная температура.
            alpha (float): Параметр уменьшения температуры.
        '''
        self.graph = graph
        self.k_max = k_max
        self.T = T
        self.alpha = alpha
        self.current_solution = self.random_solution()
        self.current_cost, _ = self.graph.calculate_cost(self.current_solution)
        self.best_solution = self.current_solution[:]
        self.best_cost = self.current_cost

    def random_solution(self) -> List[Vertex]:
        '''Генерирует случайное начальное решение.'''
        solution = list(self.graph.vertices[1:])
        random.shuffle(solution)
        return [self.graph.vertices[0]] + solution

    def neighbour(self, solution: List[Vertex]) -> List[Vertex]:
        '''Модифицирует текущее решение.'''
        new_solution = solution[1:]
        i, j = random.sample(range(len(new_solution)), 2)
        new_solution[i], new_solution[j] = new_solution[j], new_solution[i]
        return [solution[0]] + new_solution

    def acceptance_probability(self, delta_e: float) -> float:
        '''Вычисляет вероятность принятия нового решения.'''
        return 1.0 if delta_e < 0 else math.exp(-delta_e / self.T)

    def optimize(self) -> Tuple[List[Vertex], int]:
        '''Запускает оптимизацию методом имитации отжига и возвращает лучшее
найденное решение.'''
        k = 0
        while self.T > 1e-10 and k < self.k_max:
            new_solution = self.neighbour(self.current_solution)
            new_cost, new_calculation = self.graph.calculate_cost(new_solution)
            delta_e = new_cost - self.current_cost
            print(f"Итерация: {k + 1}/{self.k_max}")
            print(f"Температура: {self.T:.12f}")
            print("Рабочий путь:", ' -> '.join([vertex.short_name for vertex in
new_solution]))
            print("Расчёт длины рабочего пути:", new_calculation)
            print(f"Длина рабочего пути: {new_cost}")
            print("Текущий путь:", ' -> '.join([vertex.short_name for vertex in
self.current_solution]))
            current_calculation =
self.graph.calculate_cost(self.current_solution)[1]
            print("Расчёт длины текущего пути:", current_calculation)
            print(f"Длина текущего пути: {self.current_cost}")
            print(f'Разность энергий: {delta_e}')
            acceptance_probability = self.acceptance_probability(delta_e)
            print(f'Вероятность перехода в новое состояние:
{acceptance_probability:.12f}')
```

Продолжение Листинга А

```
            random_num = random.random()
            print(f'Сгенерированное случайное число: {random_num:.12f}')
            print('-' * 40)
            if acceptance_probability > random_num:
                if delta_e >= 0:
                    print('\033[95m' +'Принято худшее решение' + '\033[0m')
                self.current_solution = new_solution
                self.current_cost = new_cost
                if new_cost < self.best_cost:
                    self.best_solution = new_solution
                    self.best_cost = new_cost
            self.T *= self.alpha
            k += 1
        return self.best_solution, self.best_cost

vertex_0 = Vertex('Звезды Арбата',
                  'Отель',
                  'Москва, Новый Арбат, 32')

vertex_1 = Vertex('Московский государственный университет им. М.В. Ломоносова',
                  'МГУ',
                  'Москва, Западный административный округ, район Раменки,
территория Ленинские Горы, 1, стр. 52')

vertex_2 = Vertex('Московский государственный технический университет им. Н.Э.
Баумана',
                  'МГТУ',
                  'Москва, 2-я Бауманская ул., д. 5, стр. 1')

vertex_3 = Vertex('Московский физико-технический институт',
                  'МФТИ',
                  'Московская область, г. Долгопрудный, Институтский переулок,
д. 9.')

vertex_4 = Vertex('Национальный исследовательский ядерный университет «МИФИ»',
                  'МИФИ',
                  'Москва, Каширское шоссе, 31')

vertex_5 = Vertex('Высшая школа экономики',
                  'ВШЭ',
                  'Милютинский переулок, 2/9, Москва, 101000')

vertex_6 = Vertex('Московский государственный институт международных отношений
МИД РФ',
                  'МГИМО',
                  'проспект Вернадского, 76кГ, Москва, 119454')

vertex_7 = Vertex('Российская академия народного хозяйства и государственной
службы при Президенте РФ',
                  'РАНХиГС',
                  'проспект Вернадского, 84с1, Москва, 119606')

vertex_8 = Vertex('Финансовый университет при Правительстве РФ',
                  'ФУ',
                  'Ленинградский проспект, 51к1, Москва, 125167')

vertex_9 = Vertex('Первый Московский государственный медицинский университет им.
И.М. Сеченова',
                  'МГМУ',
                  'Трубецкая улица, 8с2, Москва, 119048')
```

Продолжение Листинга А

```
vertex_10 = Vertex('Российский экономический университет им. Г.В. Плеханова',
                   'РЭУ',
                   'Стремянный переулок, 36, Москва, 115054')

vertex_11 = Vertex('Университет науки и технологий МИСИС',
                   'МИСИС',
                   'Ленинский проспект, 2/4, Москва, 119049')

vertex_12 = Vertex('Российский университет дружбы народов',
                   'РУДН',
                   'улица Миклухо-Маклая, 6, Москва, 117198')

vertex_13 = Vertex('Российский национальный исследовательский медицинский
университет им. Н.И. Пирогова',
                   'РНИМУ',
                   'улица Островитянова, 1с7, Москва, 117513')

vertex_14 = Vertex('Московский авиационный институт',
                   'МАИ',
                   'Волоколамское шоссе, 4к6, Москва, 125310')

vertex_15 = Vertex('Национальный исследовательский университет «МЭИ»',
                   'МЭИ',
                   'Красноказарменная ул., 17 строение 1Г, Москва, 111250')

vertex_16 = Vertex('Московский государственный юридический университет им. О.Е.
Кутафина',
                   'МГЮА',
                   'Садовая-Кудринская улица, 9с1, Москва, 123242')

vertex_17 = Vertex('Российский государственный университет нефти и газа им. И.
М. Губкина',
                   'РГУ',
                   'Ленинский проспект, 65к1, Москва, 119296')

vertex_18 = Vertex('Московский педагогический государственный университет',
                   'МПГУ',
                   'проспект Вернадского, 88, Москва, 119571')

vertex_19 = Vertex('Национальный исследовательский Московский государственный
строительный университет',
                   'НИУ МГСУ',
                   'Ярославское шоссе, 26к1, Москва, 129337')

vertex_20 = Vertex('Московский государственный лингвистический университет',
                   'МГЛУ',
                   'улица Остоженка, 38с1, Москва, 119034')

vertex_21 = Vertex('Всероссийская академия внешней торговли',
                   'ВАВТ',
                   'Воробьёвское шоссе, 6А, Москва, 119285')

vertex_22 = Vertex('Российский химико-технологический университет им. Д.И.
Менделеева',
                   'РХТУ',
                   'Миусская площадь, 9, Москва')

vertex_23 = Vertex('МИРЭА – Российский технологический университет',
                   'МИРЭА',
                   'проспект Вернадского, 86с2, Москва')
```

Окончание Листинга А

```
vertices = [vertex_0, vertex_1, vertex_2, vertex_3, vertex_4,
            vertex_5, vertex_6, vertex_7, vertex_8, vertex_9,
            vertex_10, vertex_11, vertex_12, vertex_13, vertex_14,
            vertex_15, vertex_16, vertex_17, vertex_18, vertex_19,
            vertex_20, vertex_21, vertex_22, vertex_23]

graph = Graph(vertices)
# graph.set_weights()
# graph.print_adjacency_matrix(False)
# graph.normalize_matrix()
graph.set_weights_from_file('universities_info.csv')
graph.print_adjacency_matrix(False)
graph.show_graph()

solution = SimulatedAnnealing(graph, 100, 100, 0.5)
best_solution, best_cost = solution.optimize()
print("Лучший найденный путь:", ' -> '.join([vertex.short_name for vertex in
best_solution]))
print("Время пути:", best_cost)
print("Текущий путь: ", ' -> '.join([vertex.short_name for vertex in
solution.current_solution]))
print("Время пути:", solution.current_cost)

# test_graph = Graph([vertex_0, vertex_2, vertex_3, vertex_4, vertex_7,
vertex_11, vertex_23])
# test_graph.set_weights()
# test_graph.print_adjacency_matrix(False)
# test_graph.normalize_matrix()
# test_graph.save_matrix_to_csv('shit.csv')
# # test_graph.set_weights_from_file('universities_test.csv')
# test_graph.print_adjacency_matrix()
# test_graph.show_graph()

# sol = SimulatedAnnealing(test_graph, 100, 100, 0.5)
# best_solution, best_cost = sol.optimize()
# print("Лучший найденный путь:", ' -> '.join([vertex.short_name for vertex in
best_solution]))
# print("Время пути:", best_cost)
# print("Текущий путь: ", ' -> '.join([vertex.short_name for vertex in
sol.current_solution]))
# print("Время пути:", sol.current_cost)
```

Приложение Б

Код реализации метода имитации отжига Коши для задачи поиска глобального минимума функции

Листинг Б – Реализация метода имитации отжига Коши для задачи поиска глобального минимума

```
from typing import Callable, List, Tuple
import random
import math

class SimulatedAnnealing:
    def __init__(self, func: Callable[..., float], bounds: List[Tuple[float, float]], k_max: int, T0: float):
        '''Инициализирует алгоритм имитации отжига с заданной целевой функцией, границами поиска, максимальным числом итераций и начальной температурой.
        Параметры:
            func (Callable[[float, float], float]): Целевая функция, минимизация которой требуется.
            bounds (List[Tuple[float, float]]): Границы для каждой переменной в формате [(min, max), ...].
            k_max (int): Максимальное количество итераций.
            T0 (float): Начальная температура.
        '''
        self.func = func
        self.bounds = bounds
        self.k_max = k_max
        self.T0 = T0
        self.D = len(bounds)  # Размерность пространства состояний
        self.current_solution = self.random_solution()
        self.current_cost = self.func(*self.current_solution)

    def random_solution(self) -> List[float]:
        '''Генерирует случайное начальное решение в пределах указанных границ.
        Возвращает:
            List[float]: Список значений переменных, представляющих решение.
        '''
        return [random.uniform(b[0], b[1]) for b in self.bounds]

    @staticmethod
    def cauchy_distribution(x: float, main_x: float, temperature: float) -> float:
        '''Вычисляет распределение Коши для данной точки.
        Параметры:
            x (float): Точка, в которой вычисляется распределение.
            main_x (float): Основная точка, определяющая центр распределения.
            temperature (float): Текущая температура.
        Возвращает:
            float: Значение распределения.
        '''
        return (1 / math.pi) * temperature / ((x - main_x) ** 2 + temperature ** 2)

    def generate_solution(self, temperature: float) -> List[float]:
        '''Генерирует новое решение на основе текущего, используя распределение Коши.
        Параметры:
            temperature (float): Текущая температура.
        Возвращает:
```

Продолжение Листинга Б

```
            List[float]: Новое решение.
        '''
        new_solution = []
        for i in range(self.D):
            while True:
                main_x = self.current_solution[i]
                new_x = random.uniform(self.bounds[i][0], self.bounds[i][1])
                p_distribute = self.cauchy_distribution(
                    new_x, main_x, temperature)
                p = random.random()
                if p <= p_distribute:
                    new_solution.append(new_x)
                    break
        return new_solution

    def temperature(self, k: int) -> float:
        '''Вычисляет температуру на текущей итерации.
        Параметры:
            k (int): Текущий номер итерации.
        Возвращает:
            float: Значение температуры.
        '''
        return self.T0 / (k ** (1 / self.D))

    def acceptance_probability(self, e_old: float, e_new: float, T: float) ->
float:
        '''Вычисляет вероятность принятия нового решения.
        Параметры:
            e_old (float): Энергия текущего решения.
            e_new (float): Энергия нового решения.
            T (float): Текущая температура.
        Возвращает:
            float: Вероятность принятия нового решения.
        '''
        if e_new < e_old:
            return 1.0
        return math.exp(-(e_new - e_old) / T)

    def optimize(self) -> Tuple[List[float], float]:
        '''Запускает алгоритм оптимизации для поиска минимального значения
функции.
        Возвращает:
            Tuple[List[float], float]: Координаты минимального решения и
значение функции в этой точке.
        '''
        best_solution = self.current_solution
        best_cost = self.current_cost
        k = 1
        while k <= self.k_max:
            T = self.temperature(k)
            new_solution = self.generate_solution(T)
            new_cost = self.func(*new_solution)
            print(f"Итерация: {k}/{self.k_max}")
            print(f"Температура: {T:.12f}")
            print("Текущее решение:", self.current_solution)
            print(f"Текущая стоимость: {self.current_cost:.12f}")
            print("Новое решение:", new_solution)
            print(f"Новое значение функции: {new_cost:.12f}")
            acceptance_probability = self.acceptance_probability(
                self.current_cost, new_cost, T)
            print(f"Вероятность принятия нового решения: {
```

Окончание Листинга Б

```
                    acceptance_probability:.12f}")
            random_num = random.random()
            print(f"Сгенерированное число: {random_num:.12f}")
            if acceptance_probability >= random_num:
                if new_cost - self.current_cost > 0:
                    print('\033[95m' + 'Принято худшее решение' + '\033[0m')
                self.current_solution = new_solution
                self.current_cost = new_cost
                if new_cost < best_cost:
                    best_solution = new_solution
                    best_cost = new_cost
            print('-' * 40)
            k += 1
        return best_solution, best_cost

# Функция Гольдштейна-Прайса
def goldstein_price(x: float, y: float) -> float:
    term1 = (1 + (x + y + 1)**2 * (19 - 14*x + 3*x**2 - 14*y + 6*x*y + 3*y**2))
    term2 = (30 + (2*x - 3*y)**2 * (18 - 32*x +
             12*x**2 + 48*y - 36*x*y + 27*y**2))
    return term1 * term2

bounds = [(-2, 2), (-2, 2)]
solution = SimulatedAnnealing(goldstein_price, bounds, k_max=2500, T0=20)
result = solution.optimize()
print("Координаты минимума:", result[0])
print("Минимальное значение функции:", result[1])
print("Координаты текущего решения:", solution.current_solution)
print("Стоимость текущего решения:", solution.current_cost)
```