



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

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

(наименование дисциплины)

Тема курсовой работы «Киноиндустрия»

Студент группы ИКБО-04-22 Кликушин В.И.

(учебная группа, фамилия, имя отчество, студента)

(подпись студента)

Руководитель курсовой работы к.т.н., доцент Сорокин А.Б.

(должность, звание, ученая степень)

(подпись руководителя)

Рецензент (при наличии)

(должность, звание, ученая степень)

(подпись рецензента)

Работа представлена к защите «27» 12 2024г.

Допущен к защите «27» 12 2024г.

отл.

Москва 2024 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий

Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой

Платонова О.В.

ФИО

«15» сентябрь 2024г.

ЗАДАНИЕ

на выполнение курсовой работы по дисциплине

«Системный анализ данных в системах поддержки принятия решений»

Студент Кликушин В.И. Группа ИКБО-04-22

Тема: «Киноиндустрия»

Исходные данные: Набор данных Pima Indians Diabetes Database, программа Protege, библиотеки scikit-learn, pandas, numpy, matplotlib.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:
реализовать консольное приложение: 1. Для онтологии по предметной области, 2. Алгоритма имитации отжига, 3. Алгоритма роя частиц, 4. Алгоритма пчелиной колонии, 5. Муравьиный алгоритм.

Срок представления к защите курсовой работы:

до «28» декабрь 2024г.

Задание на курсовую работу выдал

Подпись руководителя

(Сорокин А.Б.)

Ф.И.О. руководителя

Задание на курсовую работу получил

Подпись обучающегося

«15» сентябрь 2024г

(Кликушин В.И.)

Ф.И.О. исполнителя

Москва 2024г.

ОТЗЫВ

на курсовую работу

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

Студент Кликушин В.И.
(ФИО студента)

ИКБО-04-22
(Группа)

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	+		
2. Соответствие курсовой работы заданию	+		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	+		
4. Полнота выполнения всех пунктов задания	+		
5. Логичность и системность содержания курсовой работы	+		
6. Отсутствие фактических грубых ошибок	+		

Замечаний: *нет*

Рекомендуемая оценка: *отлично*


Подпись руководителя

Сорокин А.Б.

ФИО руководителя

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ОНТОЛОГИЯ	7
1.1 Понятие онтологии	7
1.2 Постановка задачи.....	8
1.3 Описание предметной области	8
1.4 Иерархия классов	9
1.5 Реализация в Protégé	9
1.6 Программная реализация	13
2 МЕТОД ИМИТАЦИИ ОТЖИГА	16
2.1 Описание алгоритма	16
2.2 Постановка задачи.....	16
2.3 Задача коммивояжёра	18
2.3.1 Математическая модель	18
2.3.2 Ручной расчёт	21
2.4 Поиск глобального минимума	23
2.5 Программная реализация	25
3 АЛГОРИТМ РОЯ ЧАСТИЦ.....	29
3.1 Описание алгоритма	29
3.2 Постановка задачи.....	30
3.3 Математическая модель	30
3.4 Глобальный роевой алгоритм	32
3.4.1 Особенность реализации	32
3.4.2 Ручной расчёт	33
3.5 Локальный роевой алгоритм	38
3.5.1 Особенность реализации	38
3.5.2 Ручной расчёт	39
3.6 Программная реализация	39
4 МУРАВЬИНЫЙ АЛГОРИТМ	40

4.1 Описание алгоритма	40
4.2 Постановка задачи.....	41
4.3 Математическая модель	41
4.4 Ручной расчёт	45
4.5 Программная реализация	50
5 ПЧЕЛИНЫЙ АЛГОРИТМ	51
5.1 Описание алгоритма	51
5.2 Постановка задачи.....	52
5.3 Математическая модель	52
5.4 Ручной расчёт	53
5.5 Программная реализация	58
6 АЛГОРИТМ РОЯ СВЕТЛЯЧКОВ.....	59
6.1 Описание алгоритма	59
6.2 Постановка задачи.....	60
6.3 Математическая модель	60
6.4 Ручной расчёт	63
6.5 Программная реализация	65
ЗАКЛЮЧЕНИЕ	68
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ	69
ПРИЛОЖЕНИЯ.....	71

ВВЕДЕНИЕ

Системный анализ данных является ключевым направлением, объединяющим гуманитарные и формальные методы познания. Современные подходы в этой области опираются на интеграцию концепций философии, математики и кибернетики, позволяя моделировать сложные системы и анализировать их поведение. Одним из таких инструментов является использование онтологий — представлений о моделях мира, которые обеспечивают структуризацию информации, упрощая её анализ и обработку.

Колоссальный рост объёмов данных, их зашумленность, противоречивость и разноуровневость создают потребность в точных моделях, которые могут быть интерпретированы разными специалистами для решения прикладных задач. Онтологии, как средство структуризации знаний, являются фундаментом для разработки интеллектуальных систем и систем поддержки принятия решений. Они позволяют эффективно решать задачи интеграции знаний и поиска информации.

Вместе с онтологиями в системах анализа данных и поддержки принятия решений активно применяются алгоритмы оптимизации, такие как имитация отжига, алгоритмы роя частиц, муравьиные алгоритмы, пчелиные алгоритмы и алгоритмы роя светлячков. Эти методы основаны на принципах самоорганизации и коллективного поведения, которые наблюдаются в природе. Использование таких подходов позволяет моделировать сложные системы и находить оптимальные решения в многомерных пространствах.

Работа посвящена изучению и реализации перечисленных методов, начиная с построения онтологии и заканчивая сравнением различных алгоритмов оптимизации. Основной акцент сделан на изучении их практического применения в задачах оптимизации функции, что позволяет глубже понять их возможности и ограничения.

1 ОНТОЛОГИЯ

1.1 Понятие онтологии

В современном мире, характеризующемся бурным развитием информационных технологий, возникает острая необходимость в систематизации и структуризации знаний. Традиционные методы хранения и обработки информации оказываются неэффективными перед лицом колоссальных объемов данных, их разнообразных форматов и чрезвычайной зашумленности. В этих условиях важную роль играет онтология – область знаний, занимающаяся формальным описанием концепций и отношений между ними в какой-либо предметной области.

Появление онтологий стало ответом ряда наук, связанных с информационными технологиями и системами искусственного интеллекта на перечисленные проблемы. Именно они обеспечили возможность их перехода на новый качественный уровень обработки и поиска информации. Наиболее распространенными стали следующие определения. Формальная модель онтологии (O) может быть представлена как упорядоченная тройка элементов $\langle X, R, F \rangle$, где X – конечное множество концептов, R – конечное множество отношений между концептами; F – конечное множество функций интерпретации.

Онтология – это формальная точная спецификация совместно используемой концептуализации.

Онтологии – это базы знаний специального типа, которые могут читаться и пониматься, отчуждаться от разработчика и/или физически разделяться их пользователями.

Под концептуализацией понимается абстрактная модель явлений (процессов) в мире, составленная посредством определения существенных для описания данных явлений понятий, концептов. Точность подразумевает, что типы используемых понятий и ограничения на область применения данных

понятий явно определены. Формальность означает, что онтология должна быть ориентирована на компьютерное представление, что исключает использование естественных языков в полной мере, в связи с их неоднозначностью и сложностью. Совместное использование отражает понятие того, что онтология описывает всеобщие знания, то есть не персональные знания одного человека, а знания, принятые в группе, сообществе.

1.2 Постановка задачи

Цель: реализовать онтологию выбранной предметной области.

Задачи: выбрать предметную область по личному интересу и изучить её, составить модель онтологии по предметной области, реализовать модель созданной онтологии в программе Protégé, написать программный продукт на выбранном языке программирования, реализующий построенную модель онтологии.

Выбранная предметная область: «Киноиндустрия».

1.3 Описание предметной области

Актуальность выбранной предметной области заключается в том, что просмотр фильмов и сериалов – неотъемлемая часть времяпровождения многих людей на планете. Онтология киноиндустрии — это мощный инструмент, который может значительно повысить эффективность работы в этой сфере, улучшить качество информации и стимулировать развитие инновационных сервисов. Основными понятиями для рассматриваемой предметной области являются фильмы, сериалы, актёры и режиссёры. Модель онтологии позволяет найти взаимоотношения между объектами, например, узнать в каких фильмах снимался определённый актёр. Яркий пример сервиса, использующего схожую онтологию – «Кинопоиск». В этом сервисе онтология используется для организации контента и предоставления персонализированных рекомендаций.

1.4 Иерархия классов

В рассматриваемой онтологии предполагается иерархия классов, в которой, родительским классом является непосредственно абстрактный класс «Киноиндустрия». Киноиндустрия включает в себя фильмы, актёров и режиссёров. В свою очередь фильмы включают в себя многосерийные фильмы и полнометражные фильмы. Иерархия классов отображена на Рисунке 1.4.1.

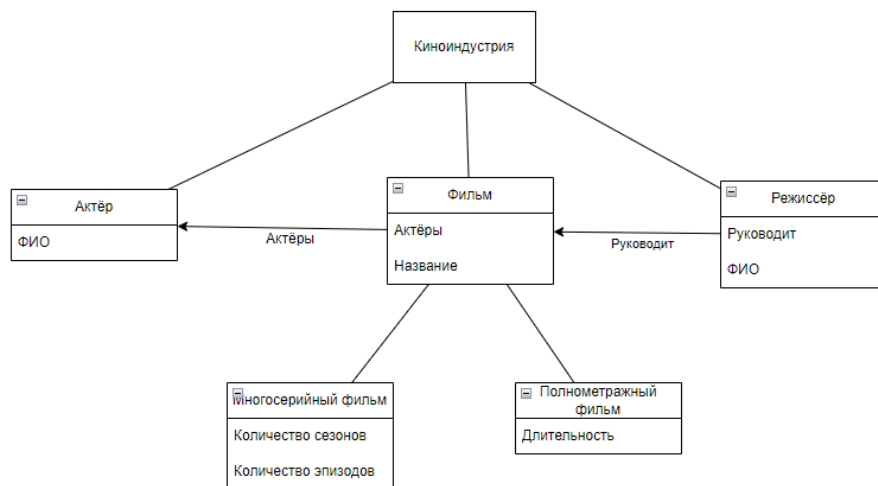


Рисунок 1.4.1 – Иерархия классов онтологии

1.5 Реализация в Protégé

Иерархия онтологии перенесена в Protégé [1] (Рисунок 1.5.1).

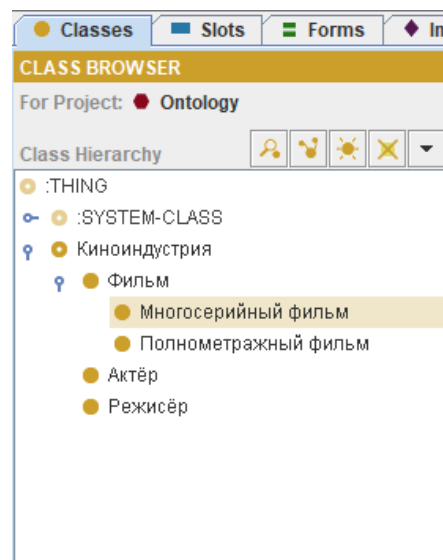


Рисунок 1.5.1 – Иерархия классов в Protégé

Класс «Киноиндустрия» является абстрактным. Созданы слоты для класса «Фильм» (Рисунок 1.5.2).

Name	Cardinality	Type	Other Facets
Актёры	multiple	Instance of Актёр	
Название	single	String	

Рисунок 1.5.2 – Слоты для класса «Фильм»

Слот «Актёры» содержит ссылку на объект класса «Актёр», слот «Название» имеет строковый тип. Созданы слоты для класса «Многосерийный фильм» (Рисунок 1.5.3).

Name	Cardinality	Type	Other Facets
Актёры	multiple	Instance of Актёр	
Количество сезонов	single	Integer	minimum=1
Количество серий	single	Integer	minimum=1
Название	single	String	

Рисунок 1.5.3 - Слоты для класса «Многосерийный фильм»

Класс «Многосерийный фильм» содержит дополнительные слоты «Количество сезонов» и «Количество эпизодов». Созданы слоты для класса «Полнометражный фильм» (Рисунок 1.5.4).

Name	Cardinality	Type	Other Facets
Актёры	multiple	Instance of Актёр	
Длительность	single	Integer	
Название	single	String	

Рисунок 1.5.4 - Слоты для класса «Полнометражный фильм»

Объекты класса «Полнометражный фильм» и «Многосерийный фильм» также являются экземплярами класса «Фильм» и содержат слоты, определенные в классе «Фильм». Созданы слоты для класса «Актёр» (Рисунок 1.5.5).

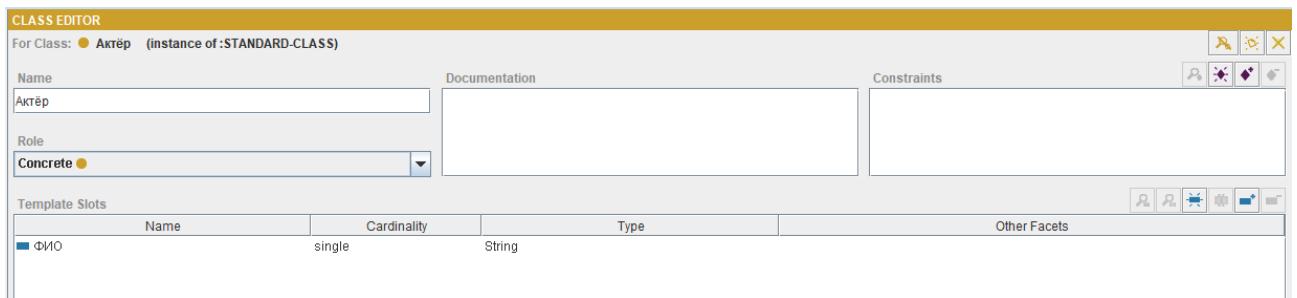


Рисунок 1.5.5 – Слоты для класса «Актёр»

Класс «Актёр» содержит единственный слот «ФИО», имеющий строковый тип. Созданы слоты для класса «Режиссёр» (Рисунок 1.5.6).

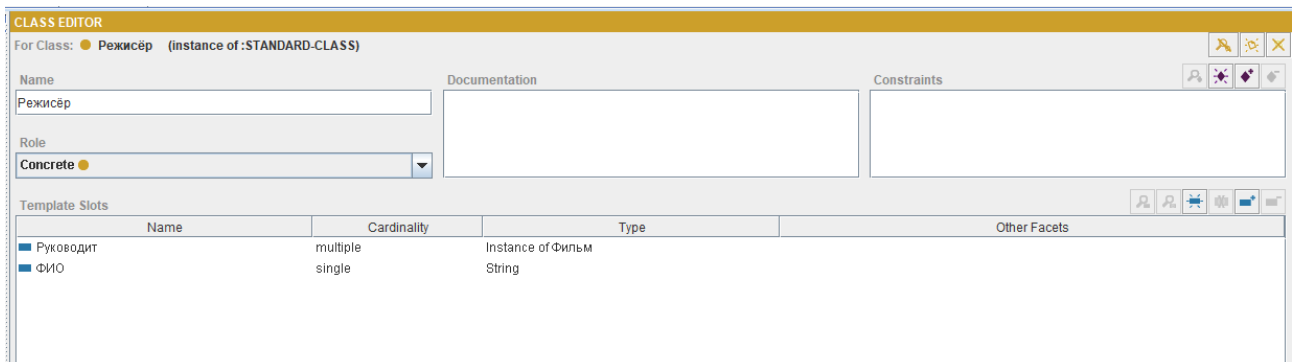


Рисунок 1.5.6 – Слоты для класса «Режиссёр»

Класс «Режиссёр» содержит слот «ФИО», имеющий строковый тип и слот «Руководит», содержащий ссылки на экземпляры класса «Фильм».

Создано несколько объектов классов (Рисунок 1.5.7).

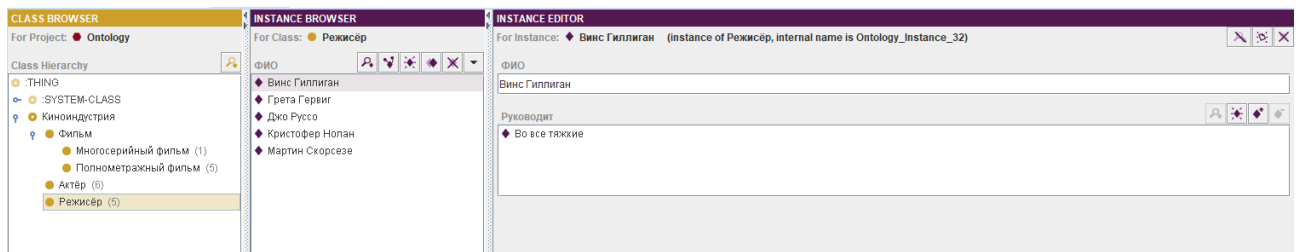


Рисунок 1.5.7 – Созданные объекты классов

Для каждого объекта заполнены соответствующие слоты (Рисунок 1.5.8).

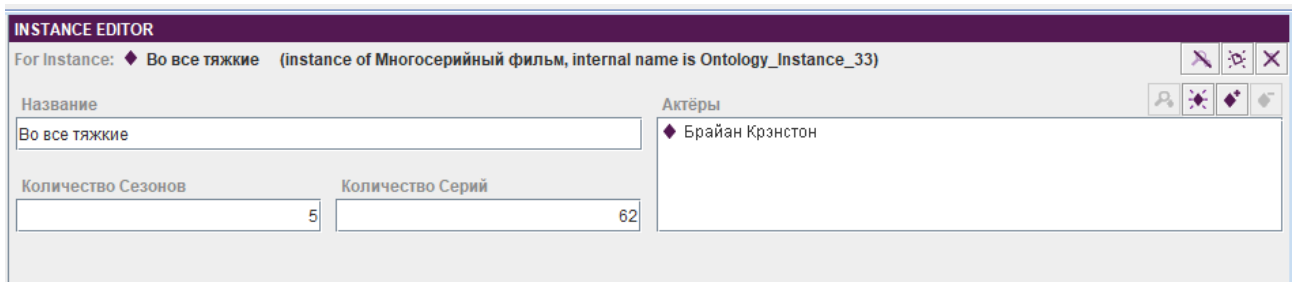


Рисунок 1.5.8 – Пример заполнения слотов для созданного экземпляра класса

Произведено несколько запросов (Рисунки 1.5.9 – 1.5.11).

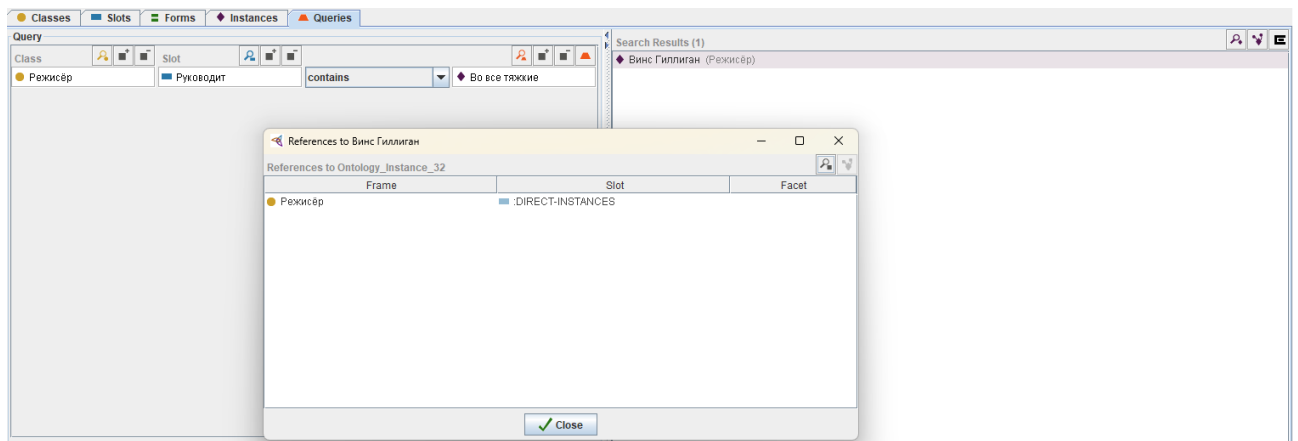


Рисунок 1.5.9 – Запрос от класса «Режиссёр»

Запрос позволяет узнать режиссёров, которые руководят выбранным фильмом.

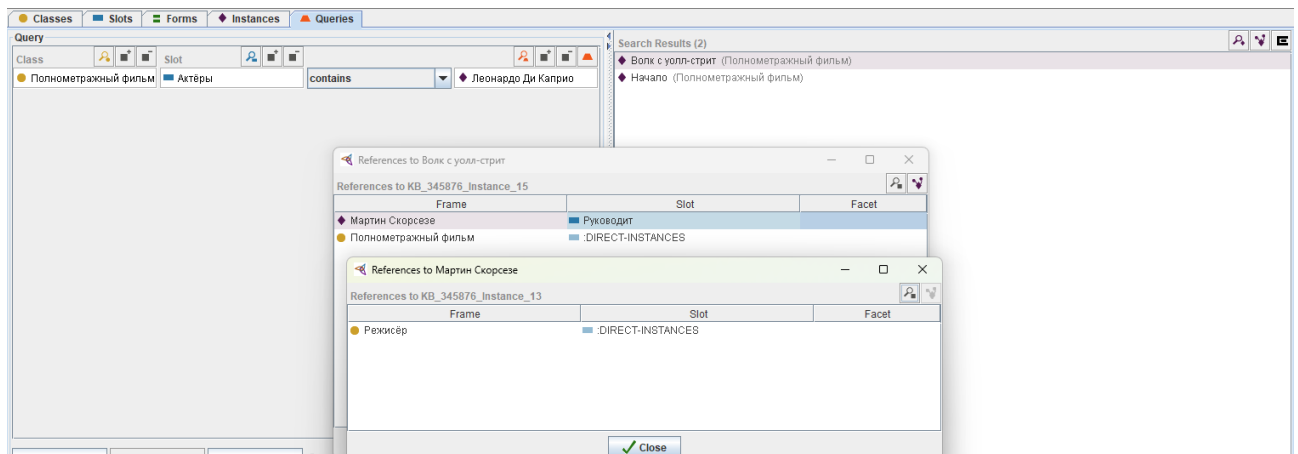


Рисунок 1.5.10 – Запрос от класса «Полнометражный фильм»

Запрос показывает сущности актёров, которые снимаются в выбранном фильме.

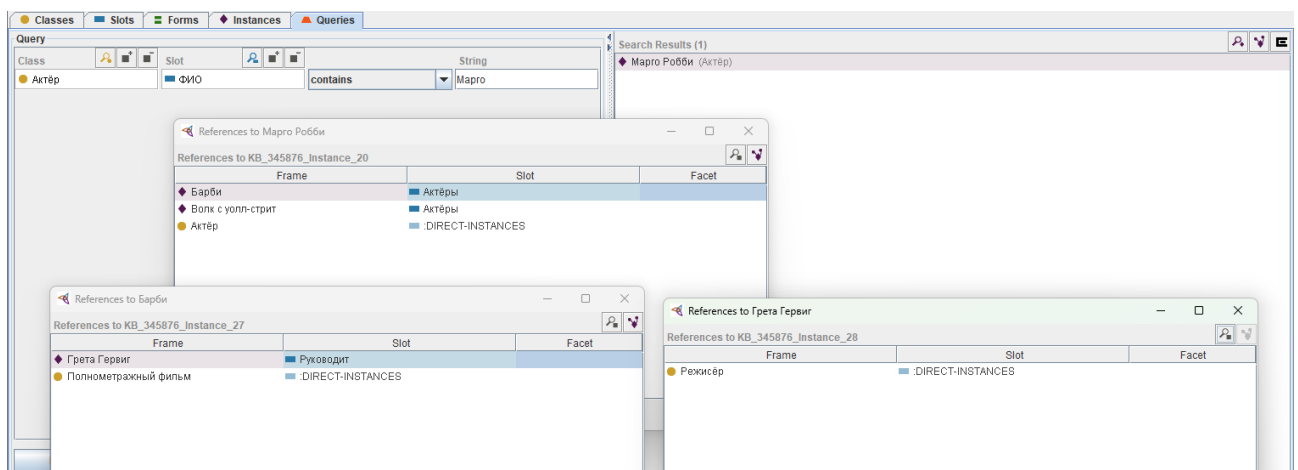


Рисунок 1.5.11 – Запрос от класса «Актёр»

Запрос позволяет узнать, в каких фильмах снимался определенный актёр, а затем узнать режиссеров этих фильмов.

1.6 Программная реализация

Разработаны классы «MovieIndustry», «Film», «SerialFilm», «FeatureFilm», «FilmDirector», «Actor», которые соответствуют созданным классам в Protégé.

Код реализации онтологии представлен в приложении А.

Созданные объекты классов представлены на Рисунке 1.6.1.

```
291 Actor('Leonardo DiCaprio'), Actor('Jonah Hill'),
292 Actor('Margot Robbie'), Actor('Kyle Chandler'),
293 Actor('Rob Reiner'), Actor('P.J. Byrne'),
294 Actor('Jon Bernthal'), Actor('Cristin Millioti'),
295 Actor('Jean Dujardin'), Actor('Matthew McConaughey'),
296 Actor('Ryan Gosling'), Actor('America Ferrera'),
297 Actor('Ariana Greenblatt'), Actor('Kate McKinnon'),
298 Actor('Issa Rae'), Actor('Will Ferrell'),
299 Actor('Michael Cera'), Actor('Simu Liu'),
300 Actor('Alexandra Shipp'), Actor('Joseph Gordon-Levitt'),
301 Actor('Elliot Page'), Actor('Tom Hardy'),
302 Actor('Ken Watanabe'), Actor('Dileep Rao'),
303 Actor('Tom Berenger'), Actor('Marion Cotillard'),
304 Actor('Pete Postlethwaite'), Actor('Paul Anderson'),
305 Actor('Sophie Rundle'), Actor('Helen McCrory'),
306 Actor('Ned Dennehy'), Actor('Finn Cole'),
307 Actor('Natasha O'Keefe'), Actor('Ian Peck'),
308 Actor('Harry Kirtan'), Actor('Packy Lee')]
309
310 serial_films = [SerialFilm('Breaking Bad', actors[:10], 5, 62),
311                 SerialFilm('Peaky Blinders', [actors[10]] + actors[47:56], 6, 36)]
312
313 feature_films = [FeatureFilm('Oppenheimer', actors[10:20], 180),
314                 FeatureFilm('The Wolf of Wall Street', actors[20:30], 172),
315                 FeatureFilm('Barbie', [actors[22]] + actors[30:39], 104),
316                 FeatureFilm('Inception', [actors[20]] + [actors[10]] + actors[39:47], 148)]
317
318 film_directors = [FilmDirector('Michelle MacLaren', [serial_films[0]]),
319                  FilmDirector('Adam Bernstein', [serial_films[0]]),
320                  FilmDirector('Vince Gilligan', [serial_films[0]]),
321                  FilmDirector('Christopher Nolan', [...
322                  FilmDirector('Martin Scorsese', [feature_films[1]]),
323                  FilmDirector('Greta Gerwig', [feature_films[2]]),
324                  FilmDirector('Anthony Byrne', [serial_films[1]]),
325                  FilmDirector('Colm McCarthy', [serial_films[1]]),
326                  FilmDirector('Tim Mielants', [serial_films[1]]),
327                  FilmDirector('David Caffrey', [serial_films[1]]),
328                  FilmDirector('Otto Bathurst', [serial_films[1]]),
329                  FilmDirector('Tom Harper', [serial_films[1]])]
```

Рисунок 1.6.1 – Созданные объекты классов

Результаты выполнения запросов от каждого класса представлены на Рисунках 1.6.2–1.6.6.

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
1
Выберите слот:
1 - Актёры
2 - Название
1
Выберите функцию запроса:
1 - contains
2 - does not contain
1
Введите значение запроса: Margot Robbie
• The Wolf of Wall Street
  • Martin Scorsese
• Barbie
  • Greta Gerwig
```

Рисунок 1.6.2 – Запрос от класса «Фильм»

Запрос отображает все фильмы (сериалы и полнометражные фильмы), в которых снималась Марго Робби, а также режиссёров этих фильмов.

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
2
Выберите слот:
1 - Актёры
2 - Название
3 - Количество сезонов
4 - Количество эпизодов
3
Выберите функцию запроса:
1 - is
2 - is greater then
3 - is less then
2
Введите значение запроса: 5
• Peaky Blinders
  • Anthony Byrne
  • Colm McCarthy
  • Tim Mielants
  • David Caffrey
  • Otto Bathurst
  • Tom Harper
PS C:\python_projects>
```

Рисунок 1.6.3 – Запрос от класса «Многосерийный фильм»

Запрос выводит все многосерийные фильмы, в которых количество сезонов более, чем пять, а также режиссёров этих фильмов.

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
3
Выберите слот:
1 - Актёры
2 - Название
3 - Длительность
2
Выберите функцию запроса:
1 - contains
2 - does not contain
3 - is
4 - is not
5 - begins with
6 - ends with
1
Введите значение запроса: i
• Oppenheimer
  • Christopher Nolan
• Barbie
  • Greta Gerwig
• Inception
  • Christopher Nolan
PS C:\python_projects>
```

Рисунок 1.6.4 – Запрос от класса «Полнометражный фильм»

Запрос отображает все полнометражные фильмы, в названии которых присутствует буква «i», а также режиссёров этих фильмов.

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
4
Выберите слот:
1 - ФИО
1
Выберите функцию запроса:
1 - contains
2 - does not contain
3 - is
4 - is not
5 - begins with
6 - ends with
1
Введите значение запроса: Mar
• Margot Robbie
  • The Wolf of Wall Street
  • Martin Scorsese
  • Barbie
  • Greta Gerwig
• Marion Cotillard
  • Inception
  • Christopher Nolan
PS C:\python_projects>
```

Рисунок 1.6.5 – Запрос от класса «Актёр»

Запрос отображает всех актёров, в именах и фамилиях которых присутствует сочетание букв «Mar», а также фильмы, в которых снимались актёры, режиссёров этих фильмов.

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
5
Выберите слот:
1 - ФИО
2 - Руководит
2
Выберите функцию запроса:
1 - contains
2 - does not contain
1
Введите значение запроса: Breaking Bad
• Michelle MacLaren
• Adam Bernstein
• Vince Gilligan
PS C:\python_projects>
```

Рисунок 1.6.6 – Запрос от класса «Режиссёр»

Запрос отображает режиссёров, которые участвовали в создании многосерийного фильма «Breaking Bad».

2 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 [2].

2.1 Описание алгоритма

Принцип работы алгоритма [3]:

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 (Формула 2.3.2.2).
4. Температура постепенно снижается, уменьшая вероятность принятия худших решений, что приводит к сужению области поиска и стабилизации алгоритма (Формула 2.3.2.1).
5. Алгоритм завершает работу, когда температура становится ниже заданного порогового значения.

2.2 Постановка задачи

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

Задачи: 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 2.2.1).

$$f(x, y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]. \quad (2.2.1)$$

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

2.3 Задача коммивояжёра

2.3.1 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 2.3.1.1.

$$(n - 1)! = (7 - 1)! = 6! = 1 * 2 * 3 * 4 * 5 * 6 = 720 \quad (2.3.1.1)$$

Для полного взвешенного ориентированного графа, состоящего из семи вершин, число рёбер E , вес которых необходимо заполнить, рассчитано по Формуле 2.3.1.2.

$$E = n * (n - 1) = 7 * 6 = 42 \quad (2.3.1.2)$$

Местоположения приёмных комиссий университетов и отеля представлены 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 2.3.1.1.

Таблица 2.3.1.1 – 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» [4]. 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 2.3.1.2 с указанием сокращённых названий вершин, составляющих ребро.

Таблица 2.3.1.2 – 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 2.3.1.1.

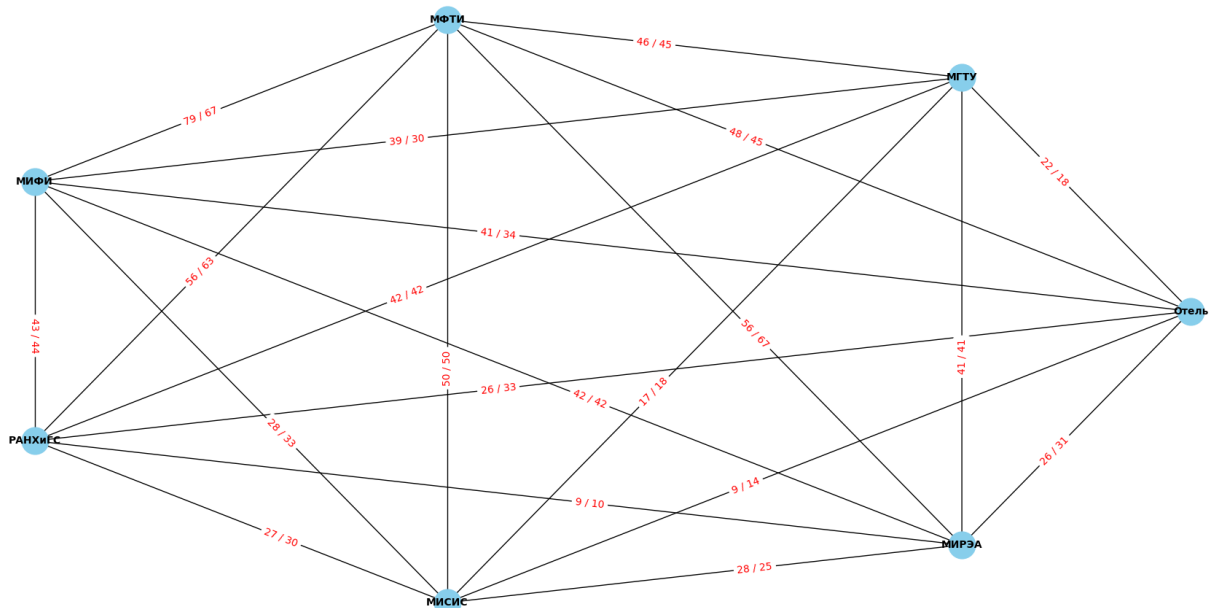


Рисунок 2.3.1.1 – Граф модели задачи

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

2.3.2 Ручной расчёт

Случайным образом составлен первоначальный маршрут (Листинг 2.3.2.1):

Листинг 2.3.2.1 – 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 2.3.2.1.

$$T_{k+1} = 0.5 * T_k \quad (2.3.2.1)$$

Верхняя граница перехода на худшее решение рассчитывается по Формуле 2.3.2.2.

$$h(\Delta E, T) = \exp\left(\frac{-\Delta E}{T}\right) \quad (2.3.2.2)$$

Путём перестановки двух вершин в текущем маршруте получено рабочее решение (Листинг 2.3.2.2).

Листинг 2.3.2.2 – 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 (Формула 2.3.2.3).

$$h(\Delta E, T) = \exp\left(\frac{-\Delta E}{T}\right) = \exp\left(\frac{281-297}{100}\right) = 0.852143788966 \quad (2.3.2.3)$$

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 2.3.2.3).

Листинг 2.3.2.3 – 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: $T_2 = T_1 / 2 = 50 / 2 = 25$.

2.4 Поиск глобального минимума

В качестве функции для поиска глобального минимума выбрана функция Гольдшейна-Прайса (Формула 2.2.1).

График функции представлен на Рисунке 2.4.1.

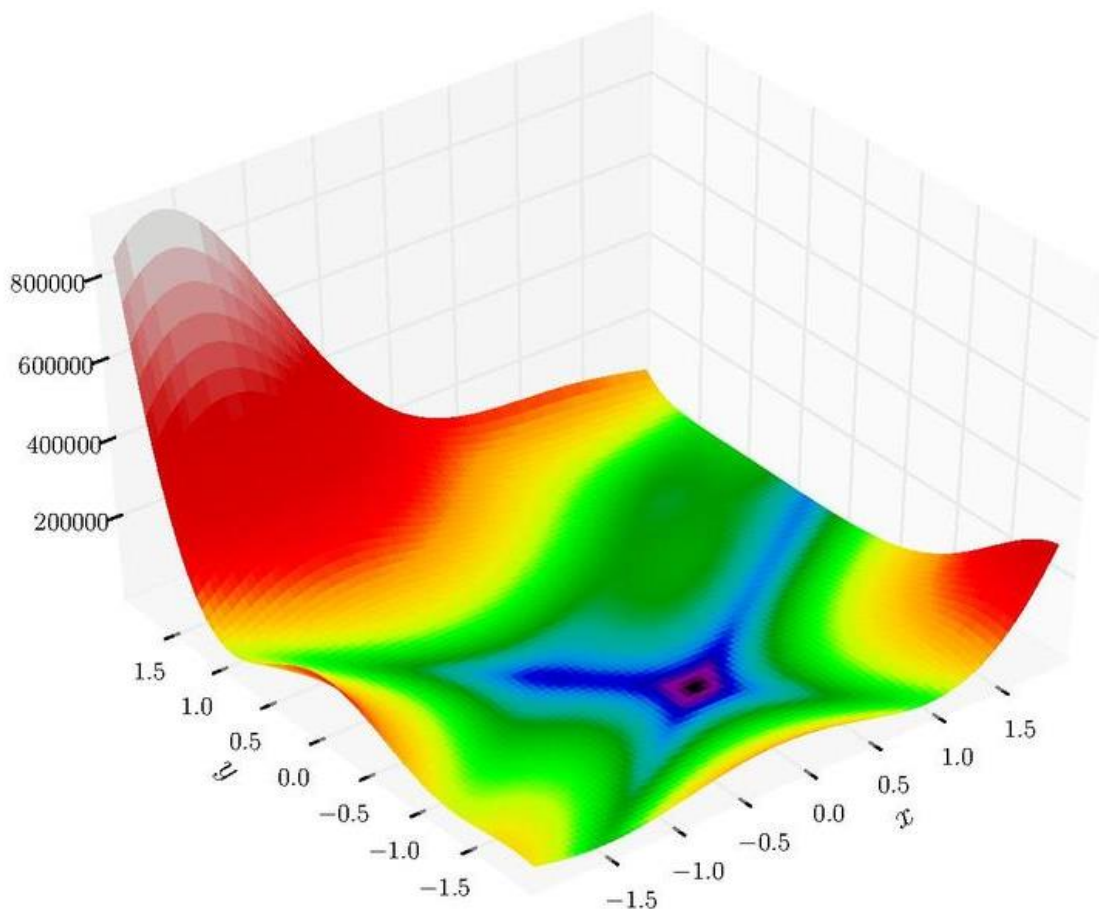


Рисунок 2.4.1 – График функции Гольдшейна-Прайса

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 2.4.1.

$$T_k = \frac{T_0}{k^{\frac{1}{D}}} \quad (2.4.1)$$

Начальная температура $T_0 = 200$ °C.

Текущее решение на каждой итерации генерируется с использованием распределения Коши (Формула 2.4.2), где $D = 2$, так как задача рассматривается в двумерном пространстве.

$$g(x', x, T) = \frac{1}{\pi^D} \prod_{i=1}^D \frac{T}{|x' - x|^2 + T^2} \quad (2.4.2)$$

В начале итерации уменьшается температура (Формула 2.4.3):

$$T_1 = \frac{T_0}{k^{\frac{1}{D}}} = \frac{200}{1^{\frac{1}{2}}} = 200 \quad (2.4.3)$$

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 2.4.4).

$$h(\Delta E, T_0) = e^{-\frac{\Delta f}{T_0}} = e^{-\frac{1618.40 - 123.72}{200}} \approx 0.000567 \quad (2.4.4)$$

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

Перед второй итерацией температура вновь уменьшается в соответствии с законом охлаждения Коши (Формула 2.4.5).

$$T_2 = \frac{T_1}{k^{\frac{1}{D}}} = \frac{200}{2^{\frac{1}{2}}} = 141.42 \quad (2.4.5)$$

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 2.4.6).

$$h(\Delta E, T_0) = e^{-\frac{\Delta f}{T_0}} = e^{-\frac{344.59 - 123.72}{141.42}} \approx 0.209758 \quad (2.4.6)$$

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.

2.5 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 Б.1.

Процесс автоматического заполнения матрицы весов представлен на Рисунке 2.5.1. Функция может не отработать и выбросить исключение из-за плохого интернет-соединения.

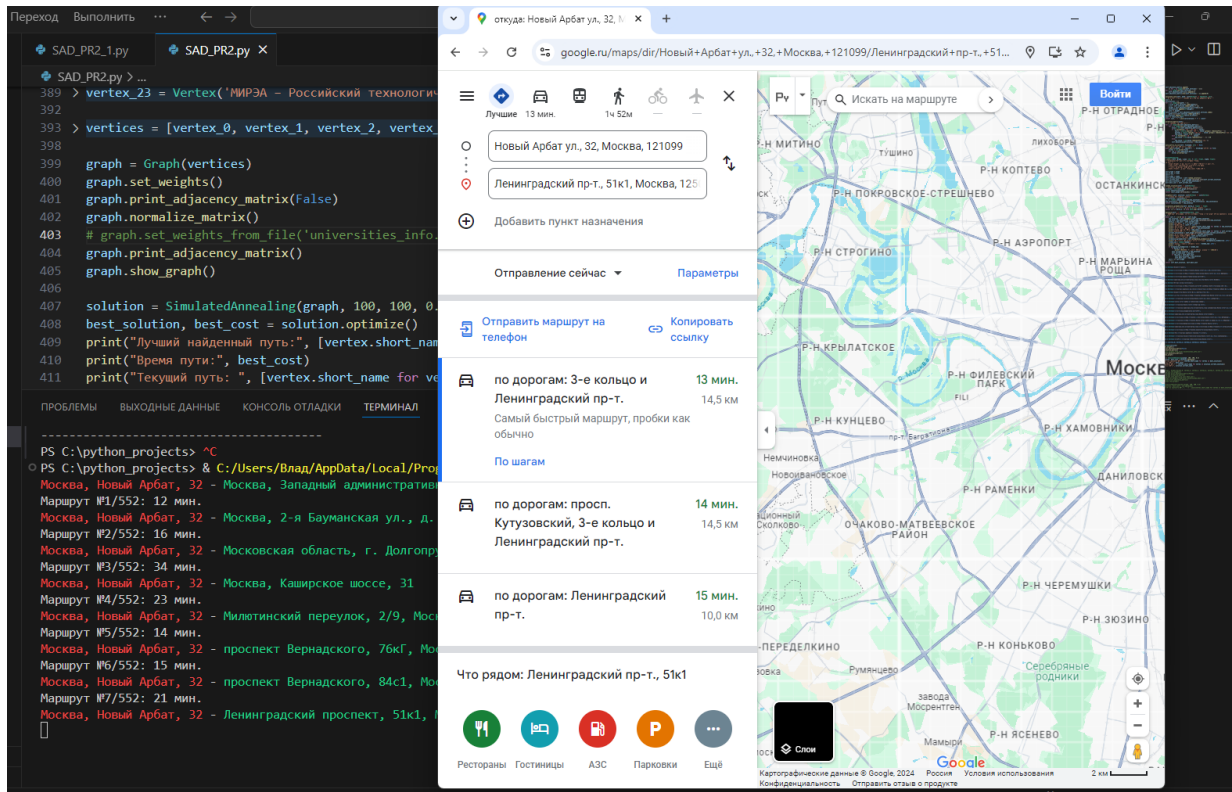


Рисунок 2.5.1 – Автоматическое заполнение матрицы весов

Выведенная матрица весов отображена на Рисунке 2.5.2.

Отель	МГУ	МГУТ	МФТИ	МФАН	ВШЭ	МИФИ	РАНХиГС	ФУ	МГУ	РЭУ	МИСИС	РУДН	РНИМУ	МАИ	МЭИ	МГВА	РГУ	МПУ	НИУ МГУ	МГУ	ВБВТ	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	16	29	24	18	9	17	30	15	10	21	13	13
13	23	0	33	22	12	24	29	17	17	12	15	29	38	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	9	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 2.5.2 – 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 2.5.3.

```

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 2.5.3 – Инициализация графа, запуск алгоритма имитации отжига

Первые итерации работы алгоритма представлены на Рисунке 2.5.4.

```

Итерация: 1/100
Температура: 100.000000000000
Рабочий путь: Огелъ -> МММ -> РАНХГС -> МГУ -> ММРЗА -> МГТУ -> МПГУ -> МГМУ -> МГИМО -> РХТУ -> МАИ -> РГУ -> ФУ -> МТИ -> МГЛА -> РЭУ -> МКСИС -> ВАВТ -> МГЛУ -> ВШЭ -> РУДН -> НИУ МГСУ -> РНФМУ -> 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: Огелъ -> МММ -> РАНХГС -> МГУ -> ММРЗА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> МАИ -> РГУ -> ФУ -> МТИ -> МГЛА -> РЭУ -> МКСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> РНФМУ -> 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: Огелъ -> МММ -> РАНХГС -> МГУ -> ММРЗА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> РНФМУ -> РГУ -> ФУ -> МТИ -> МГЛА -> РЭУ -> МКСИС -> 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: Огелъ -> МММ -> РАНХГС -> МГУ -> ММРЗА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> МАИ -> РГУ -> ФУ -> МТИ -> МГЛА -> РЭУ -> МКСИС -> ВАВТ -> МГЛУ -> ВШЭ -> МГМУ -> НИУ МГСУ -> РНФМУ -> 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: Огелъ -> МММ -> РУДН -> МГУ -> ММРЗА -> МГТУ -> МПГУ -> РАНХГС -> МГИМО -> РХТУ -> РНФМУ -> РГУ -> ФУ -> МТИ -> МГЛА -> РЭУ -> МКСИС -> 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: Огелъ -> МММ -> РАНХГС -> МГУ -> ММРЗА -> МГТУ -> МПГУ -> РУДН -> МГИМО -> РХТУ -> РНФМУ -> РГУ -> ФУ -> МТИ -> МГЛА -> РЭУ -> МКСИС -> 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: 0.396250539006
-----

```

Рисунок 2.5.4 – Первые итерации алгоритма отжига для задачи коммивояжера

Результат работы алгоритма отжига представлен на Рисунке 2.5.5.

```

-----
Итерация: 39/100
Температура: 0.000000000364
Рабочий путь: Огелъ -> РНФМУ -> РУДН -> МГУ -> РЭУ -> РАНХГС -> МПГУ -> ММРЗА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЛА -> МГЛУ -> МКСИС -> МММ -> 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: Огелъ -> РНФМУ -> РУДН -> МГУ -> РЭУ -> РАНХГС -> МПГУ -> ММРЗА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЛА -> МГЛУ -> МКСИС -> МММ -> 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 + 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: Огелъ -> РНФМУ -> РУДН -> МГУ -> РЭУ -> РАНХГС -> МПГУ -> ММРЗА -> МГИМО -> МКСИС -> ВАВТ -> РХТУ -> МТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЛА -> МГЛУ -> МММ -> 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: Огелъ -> РНФМУ -> РУДН -> МГУ -> РЭУ -> РАНХГС -> МПГУ -> ММРЗА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЛА -> МГЛУ -> МКСИС -> МММ -> 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: Огелъ -> РНФМУ -> РУДН -> МГУ -> РЭУ -> РАНХГС -> МПГУ -> ММРЗА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЛА -> МГЛУ -> МКСИС -> МММ -> НИУ МГСУ -> МЭИ -> МГМУ
Время пути: 404
Текущий путь: Огелъ -> РНФМУ -> РУДН -> МГУ -> РЭУ -> РАНХГС -> МПГУ -> ММРЗА -> МГИМО -> РГУ -> ВАВТ -> РХТУ -> МТИ -> ФУ -> ВШЭ -> МГТУ -> МАИ -> МГЛА -> МГЛУ -> МКСИС -> МММ -> НИУ МГСУ -> МЭИ -> МГМУ
Время пути: 404

```

Рисунок 2.5.5 – Результат работы алгоритма имитации отжига для задачи коммивояжера

Код алгоритма имитации отжига Коши для задачи поиска глобального минимума функции представлен в Приложении Б.2.

Результат поиска глобального минимума функции Гольдштейна-Прайса методом имитации отжига Коши представлен на Рисунке 2.5.6.

```
-----
Итерация: 2498/2500
Температура: 0.400160096064
Текущее решение: [-0.008263223642874351, -0.9916193067623662]
Текущая стоимость: 3.061069192469
Новое решение: [0.07562553429642183, 1.064750158121837]
Новое значение функции: 29654.027552611853
Вероятность принятия нового решения: 0.000000000000
Сгенерированное число: 0.522146189600
-----
Итерация: 2499/2500
Температура: 0.400080024008
Текущее решение: [-0.008263223642874351, -0.9916193067623662]
Текущая стоимость: 3.061069192469
Новое решение: [0.060808597449260926, -0.48792943806039446]
Новое значение функции: 285.433859582785
Вероятность принятия нового решения: 0.000000000000
Сгенерированное число: 0.888919410777
-----
Итерация: 2500/2500
Температура: 0.400000000000
Текущее решение: [-0.008263223642874351, -0.9916193067623662]
Текущая стоимость: 3.061069192469
Новое решение: [0.5205589642424364, -1.2294874596222]
Новое значение функции: 862.442538187172
Вероятность принятия нового решения: 0.000000000000
Сгенерированное число: 0.584716308410
-----
Координаты минимума: [-0.008263223642874351, -0.9916193067623662]
Минимальное значение функции: 3.0610691924690476
Координаты текущего решения: [-0.008263223642874351, -0.9916193067623662]
Стоимость текущего решения: 3.0610691924690476
PS C:\python_projects>
```

Рисунок 2.5.6 – Результат работы метода отжига Коши для поиска глобального минимума функции

Параметры запуска алгоритма отжига Коши представлены на Рисунке 2.5.7.

```
121
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 2.5.7 – Параметры для отжига Коши

3 АЛГОРИТМ РОЯ ЧАСТИЦ

Алгоритм роя является методом численной оптимизации, поддерживающий общее количество возможных решений, которые называются частицами или агентами, и перемещая их в пространстве к наилучшему найденному в этом пространстве решению, всё время находящемуся в изменении из-за нахождения агентами более выгодных решений [5].

3.1 Описание алгоритма

Роевый алгоритм использует рой частиц, где каждая частица представляет потенциальное решение проблемы. Поведение частицы в гиперпространстве поиска решения все время подстраивается в соответствии со своим опытом и опытом своих соседей. Кроме этого, каждая частица помнит свою лучшую позицию с достигнутым локальным лучшим значением целевой функции и знает наилучшую позицию частиц - своих соседей, где достигнут глобальный на текущий момент оптимум. В процессе поиска частицы роя обмениваются информацией о достигнутых лучших результатах и изменяют свои позиции и скорости по определенным правилам на основе имеющейся на текущий момент информации о локальных и глобальных достижениях. При этом глобальный лучший результат известен всем частицам и немедленно корректируется в том случае, когда некоторая частица роя находит лучшую позицию с результатом, превосходящим текущий глобальный оптимум [6].

Обобщая выше сказанное, алгоритм состоит из следующих ключевых шагов:

1. Создание роя частиц.
2. Нахождение лучшего решения для каждой частицы.
3. Нахождение лучшего решения для всех частиц.
4. Коррекция скорости каждой частицы (Формула 3.3.5).
5. Перемещение каждой частицы (Формула 3.3.4).

6. Завершение работы, если критерий останова выполнен, иначе возврат к пункту номер два.

3.2 Постановка задачи

Цель работы: реализовать преобразование Коши методом роя частиц для нахождения приближённого глобального минимума целевой функции.

Задачи: изучить алгоритм роя частиц, выбрать тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт двух итераций алгоритма для трёх частиц, разработать программную реализацию алгоритма роя частиц для задачи минимизации функции.

Нахождение глобального минимума функции от многих переменных состоит в поиске точки в многомерном пространстве, где значение функции будет минимальным.

Выбранная функция для оптимизации: функция Гольдшейна-Прайса (Формула 3.2.1).

$$f(x, y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] \quad (3.2.1)$$

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

3.3 Математическая модель

Начальная популяция состоит из N частиц. Каждая частица описывается вектором координат и вектором скоростей (Формула 3.3.1).

$$\vec{x}_j^k = (x_{j1}^k, x_{j2}^k, \dots, x_{ji}^k, x_{jn}^k); \vec{v}_j^k = (v_{j1}^k, v_{j2}^k, \dots, v_{ji}^k, v_{jn}^k); j = 1 \dots N; i = 1 \dots n, \quad (3.3.1)$$

где x – координата частицы в гиперпространстве;

j – номер частицы;

k – номер итерации алгоритма;

i – номер координаты;

v – скорость координаты;

N – общее количество частиц;

n – количество измерений.

Лучшее решение для j -й частицы на итерациях $k = 0 \dots t$ обозначено в соответствии с Формулой 3.3.2.

$$\vec{p}_j^k = (p_{j1}^k, p_{j2}^k, \dots, p_{ji}^k, p_{jn}^k) \quad (3.3.2)$$

Лучшее решение для всех частиц на итерациях $k = 0 \dots t$ обозначено в соответствии с Формулой 3.3.3.

$$\vec{b}^k = (b_1^k, b_2^k, \dots, b_i^k, b_n^k) \quad (3.3.3)$$

Перемещение частицы осуществляется в соответствии с Формулой 3.3.4.

$$x_{ji}^{k+1} = x_{ji}^k + v_{ji}^{k+1} \quad (3.3.4)$$

Вектор скорости управляет процессом поиска решения и его компоненты определяются с учетом когнитивной и социальной составляющей. На каждой $k + 1$ итерации для j -й частицы i -я компонента скорости определяется в соответствии с Формулой 3.3.5.

$$v_{ji}^{k+1} = v_{ji}^k + a_p r_p (p_{ji}^k - x_{ji}^k) + a_b r_b (b_i^k - x_{ji}^k); \quad i = 1, \dots, n; \quad j = 1, \dots, N, \quad (3.3.5)$$

где a_p – подбираемый положительный коэффициент ускорения;

r_p – случайное число из диапазона $[0;1]$, которое вносит элемент случайности в процесс поиска;

a_b – подбираемый положительный коэффициент ускорения;

r_b – случайное число из диапазона $[0;1]$, которое вносит элемент случайности в процесс поиска.

3.4 Глобальный роевой алгоритм

Существует два основных подхода в оптимизации роя частиц, под названиями *lbest* и *gbest*, отличающиеся топологией соседства, используемой для обмена опытом между частицами.

3.4.1 Особенность реализации

При коррекции скорости частицы используется информация о положении достигнутого глобального оптимума, которая определяется на основании информации, передаваемой всеми частицами роя, то есть для модели *gbest* лучшая частица определяется из всего роя. Это классическая и наиболее популярная версия алгоритма.

На Рисунке 3.4.1 представлена структура «звезда», где все частицы связаны друг с другом (образуют полный граф) и могут соответственно обмениваться информацией.

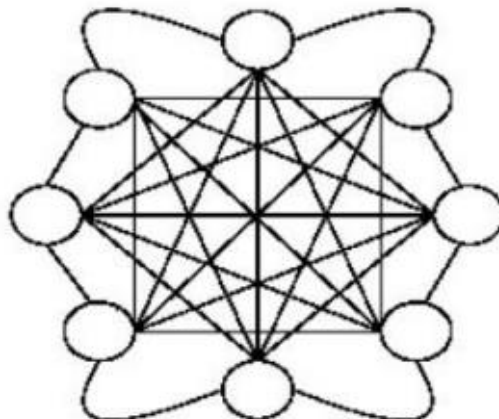


Рисунок 3.4.1 – Структура сети «звезда»

В этом случае каждая частица стремится сместиться в сторону глобальной лучшей позиции, которую нашел рой. Основной роевой алгоритм (глобальный) использует по умолчанию фактически структуру «звезда» на всем рое.

3.4.2 Ручной расчёт

Подбираемые коэффициенты ускорения приняты равными 2.

Случайным образом сгенерированы начальные координаты и компоненты скоростей для трёх частиц.

Первая частица. Координаты: (-0.0842, 0.4813); Скорость: (0.6218, -0.6931); Лучшее значение функции: 4619.98.

Вторая частица. Координаты: (-1.6099, 1.9618); Скорость: (-0.9485, -0.6894); Лучшее значение функции: 937558.12.

Третья частица. Координаты: (1.0100, -1.7479); Скорость: (-0.9026, -0.0767); Лучшее значение функции: 10001.47.

На первой итерации первая частица перемещается в соответствии с Формулами 3.4.2.1–3.4.2.2.

$$x_{11}^1 = x_{11}^0 + v_{11}^0 = -0.0842 + 0.6219 = 0.5377; \quad (3.4.2.1)$$

$$x_{12}^1 = x_{12}^0 + v_{12}^0 = 0.4813 - 0.6931 = -0.2118 \quad (3.4.2.2)$$

Значение функции в полученной точке равно 761.44. Получено лучшее решение, значит для первой частицы переписывается вектор координат и лучшее значение функции (в пределах частицы).

Обновляются координаты и значение глобального минимума для всего роя.

На первой итерации вторая частица перемещается в соответствии с Формулами 3.4.2.3–3.4.2.4.

$$x_{21}^1 = x_{21}^0 + v_{21}^0 = -1.6099 - 0.9485 = -2; \quad (3.4.2.3)$$

$$x_{22}^1 = x_{22}^0 + v_{22}^0 = 1.9618 - 0.6894 = 1.2723 \quad (3.4.2.4)$$

Так как для первой компоненты получена координата, выходящая за пределы диапазона поиска, её значением становится нижняя граница диапазона -2.

Значение фитнес-функции в полученной точке: 65561.66. Получено лучшее решение, значит для второй частицы переписывается вектор координат и лучшее значение функции (в пределах частицы).

Лучшее глобальное решение остаётся без изменений.

На первой итерации третья частица перемещается в соответствие с Формулами 3.4.2.5–3.4.2.6.

$$x_{31}^1 = x_{31}^0 + v_{31}^0 = 1.0100 - 0.9026 = 0.1073; \quad (3.4.2.5)$$

$$x_{32}^1 = x_{32}^0 + v_{32}^0 = -1.7479 - 0.0767 = -1.8246 \quad (3.4.2.6)$$

Значение функции в полученной точке равно 22400.52. Получено худшее решение, значит для третьей частицы сохраняется вектор координат и значение функции в этой точке.

Лучшее глобальное решение остаётся без изменений.

На первой итерации компоненты вектора скорости для первой частицы обновляются в соответствие с Формулами 3.4.2.7–3.4.2.8. Сгенерированные случайные числа: [0.2445, 0.8954]; [0.8667, 0.0035].

$$v_{11}^1 = v_{11}^0 + a_p r_p (p_{11}^1 - x_{11}^1) + a_b r_b (b_1^1 - x_{11}^1) = 0.6219 + 2 * 0.2445 * (0.5377 - 0.5377) + 2 * 0.8954 * (0.5377 - 0.5377) = 0.6218; \quad (3.4.2.7)$$

$$v_{12}^1 = v_{12}^0 + a_p r_p (p_{12}^1 - x_{12}^1) + a_b r_b (b_2^1 - x_{12}^1) = -0.6931 + 2 * 0.8667 * \\ * (-0.2118 + 0.2118) + 2 * 0.0035 * (-0.2118 + 0.2118) = -0.6931 \quad (3.4.2.8)$$

На первой итерации компоненты вектора скорости для второй частицы обновляются в соответствии с Формулами 3.4.2.9–3.4.2.10. Сгенерированные случайные числа: [0.1084,0.5134]; [0.9882,0.2198].

$$v_{21}^1 = v_{21}^0 + a_p r_p (p_{21}^1 - x_{11}^1) + a_b r_b (b_1^1 - x_{21}^1) = -0.9485 + 2 * 0.1084 * \\ * (-2 + 2) + 2 * 0.5134 * (0.5377 + 2) = 1.6573; \quad (3.4.2.9)$$

$$v_{22}^1 = v_{22}^0 + a_p r_p (p_{22}^1 - x_{22}^1) + a_b r_b (b_2^1 - x_{22}^1) = -0.6894 + 2 * 0.9882 * \\ * (1.2723 - 1.2723) + 2 * 0.2198 * (-0.2118 - 1.2723) = -1.3420 \quad (3.4.2.10)$$

На первой итерации компоненты вектора скорости для третьей частицы обновляются в соответствии с Формулами 3.4.2.11-3.4.2.12. Сгенерированные случайные числа: [0.0267,0.3430]; [0.4052,0.5252].

$$v_{31}^1 = v_{31}^0 + a_p r_p (p_{31}^1 - x_{31}^1) + a_b r_b (b_1^1 - x_{31}^1) = -0.9025 + 2 * 0.0267 * \\ * (1.0099 - 0.1073) + 2 * 0.3430 * (0.5377 - 0.1073) = -0.5589; \quad (3.4.2.11)$$

$$v_{32}^1 = v_{32}^0 + a_p r_p (p_{32}^1 - x_{32}^1) + a_b r_b (b_2^1 - x_{32}^1) = -0.0767 + 2 * 0.4052 * \\ * (-1.7479 + 1.8246) + 2 * 0.5252 * (-0.2118 + 1.8246) = 1.6798 \quad (3.4.2.12)$$

Под конец первой итерации лучшая точка минимума: (0.5377; -0.2118). Значение функции в этой точке: 761.44.

На второй итерации первая частица перемещается в соответствии с Формулами 3.4.2.13-3.4.2.14.

$$x_{11}^2 = x_{11}^1 + v_{11}^1 = 0.5377 + 0.6218 = 1.1596; \quad (3.4.2.13)$$

$$x_{12}^2 = x_{12}^1 + v_{12}^1 = -0.2118 - 0.6930 = -0.9049 \quad (3.4.2.14)$$

Значение функции в полученной точке равно 9514.66. Получено худшее решение, значит для первой частицы вектор координат и лучшее значение функции сохраняются без изменений (в пределах частицы).

На второй итерации вторая частица перемещается в соответствие с Формулами 3.4.2.15-3.4.2.16.

$$x_{21}^2 = x_{21}^1 + v_{21}^1 = -2 + 1.6573 = -0.3426; \quad (3.4.2.15)$$

$$x_{22}^2 = x_{22}^1 + v_{22}^1 = 1.2723 - 1.3420 = -0.0697 \quad (3.4.2.16)$$

Значение функции в полученной точке равно 349.83. Получено лучшее решение, значит для второй частицы вектор координат и лучшее значение функции переписываются (в пределах частицы).

Обновляются глобальная лучшая точка и значение функции в этой точке для всего роя.

На второй итерации третья частица перемещается в соответствие с Формулами 3.4.2.17-3.4.2.18.

$$x_{31}^2 = x_{31}^1 + v_{31}^1 = 0.1073 - 0.5589 = -0.4515; \quad (3.4.2.17)$$

$$x_{32}^2 = x_{32}^1 + v_{32}^1 = -1.8246 + 1.6798 = -0.1448 \quad (3.4.2.18)$$

Значение функции в полученной точке равно 201.20. Получено лучшее решение, значит для третьей частицы вектор координат и лучшее значение функции переписываются (в пределах частицы).

Обновляются глобальная лучшая точка и значение функции в этой точке для всего роя.

На второй итерации компоненты вектора скорости для первой частицы обновляются в соответствии с Формулами 3.4.2.19-3.4.2.20. Сгенерированные случайные числа: [0.3548, 0.4196]; [0.6361, 0.7756].

$$v_{11}^2 = v_{11}^1 + a_p r_p (p_{11}^2 - x_{11}^2) + a_b r_b (b_1^2 - x_{11}^2) = 0.6218 + 2 * 0.3548 * (0.5377 - 1.1596) + 2 * 0.4196 * (-0.4515 - 1.1596) = -1.1718; \quad (3.4.2.19)$$

$$v_{12}^2 = v_{12}^1 + a_p r_p (p_{12}^2 - x_{12}^2) + a_b r_b (b_2^2 - x_{12}^2) = -0.6930 + 2 * 0.6361 * (-0.2118 + 0.9049) + 2 * 0.7756 * (-0.1448 + 0.9049) = 1.3678 \quad (3.4.2.20)$$

На второй итерации компоненты вектора скорости для второй частицы обновляются в соответствии с Формулами 3.4.2.21-3.4.2.22. Сгенерированные случайные числа: [0.1800, 0.2495]; [0.0777, 0.1417].

$$v_{21}^2 = v_{21}^1 + a_p r_p (p_{21}^2 - x_{21}^2) + a_b r_b (b_1^2 - x_{21}^2) = 1.6573 + 2 * 0.1800 * (-0.3426 + 0.3426) + 2 * 0.2495 * (-0.4515 + 0.3426) = 1.6029; \quad (3.4.2.21)$$

$$v_{22}^2 = v_{22}^1 + a_p r_p (p_{22}^2 - x_{22}^2) + a_b r_b (b_2^2 - x_{22}^2) = -1.3420 + 2 * 0.0777 * (-0.0697 + 0.0697) + 2 * 0.1417 * (-0.1448 + 0.0697) = -1.3633 \quad (3.4.2.22)$$

На второй итерации компоненты вектора скорости для третьей частицы обновляются в соответствии с Формулами 3.4.2.23-3.4.2.24. Сгенерированные случайные числа: [0.8495, 0.2512]; [0.3009, 0.9099].

$$v_{31}^2 = v_{31}^1 + a_p r_p (p_{31}^2 - x_{31}^2) + a_b r_b (b_1^2 - x_{31}^2) = -0.5589 + 2 * 0.8495 * (-0.4515 + 0.4515) + 2 * 0.2512 * (-0.4515 + 0.4515) = -0.5589; \quad (3.4.2.23)$$

$$v_{32}^2 = v_{32}^1 + a_p r_p (p_{32}^2 - x_{32}^2) + a_b r_b (b_2^2 - x_{32}^2) = 1.6798 + 2 * 0.3009 * (-0.1448 + 0.1448) + 2 * 0.9099 * (-0.1448 + 0.1448) = 1.6798 \quad (3.4.2.24)$$

По окончании второй итерации координаты точки глобального минимума: (-0.4515; -0.1448). Значение функции в этой точке: 201.20.

3.5 Локальный роевой алгоритм

3.5.1 Особенность реализации

Локальный роевой алгоритм использует для коррекции вектора скорости частицы только локальный оптимум, который определяется на множестве соседних (ближайших в некотором смысле) частиц. То есть считается, что данной частице может передавать полезную информацию только ее ближайшее окружение. При этом отношение соседства задается некоторой «социальной» сетевой структурой, которая образует перекрывающиеся множества соседних частиц, которые могут влиять друг на друга. Соседние частицы обмениваются между собой информацией о достигнутых лучших результатах и поэтому стремятся двигаться в сторону локального в данной окрестности оптимума [6].

Выбранная социальная сеть: кольцо. Каждая частица обменивается информацией с двумя соседними частицами. В таком случае каждая частица стремится сместиться в сторону лучшего соседа. Социальная структура кольца представлена на Рисунке 3.5.1.1.

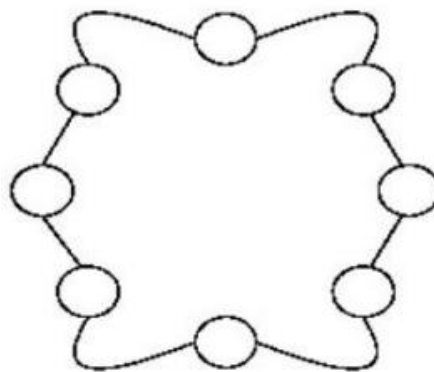


Рисунок 3.5.1.1 – Социальная структура «Кольцо»

3.5.2 Ручной расчёт

Для $n = 3$ частиц ручной расчёт локального роевого алгоритма с кольцевой структурой социальной сети полностью совпадает с расчётом глобального роевого алгоритма, потому что каждая из частиц может обмениваться с двумя оставшимися соседями (со всеми частицами в рое).

3.6 Программная реализация

Код реализации глобального роевого алгоритма представлен в Приложении В.1.

Количество частиц, как и число итераций, в алгоритме принято равным ста. Для каждой итерации выводится её номер, точка глобального минимума и значение функции в этой точке. Результат работы глобального роевого алгоритма представлен на Рисунке 3.6.1.

```
-----
Итерация: 99/100
Лучшая позиция: [0.007849194860625186, -1.0012552558296983]
Лучшее значение функции: 3.018486793979
-----
Итерация: 100/100
Лучшая позиция: [0.007849194860625186, -1.0012552558296983]
Лучшее значение функции: 3.018486793979
-----
Лучшая позиция: [0.007849194860625186, -1.0012552558296983]
Лучшее значение функции: 3.018486793979
PS C:\python_projects>
```

Рисунок 3.6.1 - Результат работы глобального роевого алгоритма

Код реализации локального роевого алгоритма с кольцевой социальной сетью представлен в Приложении В.2.

Количество частиц и число итераций совпадают с параметрами для глобального алгоритма роя частиц.

```
PS C:\python_projects> & C:/Users/Влад/AppData/Local/Programs/Python/Python312/python.exe "c:/python_projects/Local Best PSO.py"
Лучшая позиция: [-0.008192219040955018, -1.0089374014914096]
Лучшее значение функции: 3.035971410552
PS C:\python_projects>
```

Рисунок 3.6.2 - Результат работы локального роевого алгоритма

4 МУРАВЬИНЫЙ АЛГОРИТМ

4.1 Описание алгоритма

Муравьиный алгоритм использует множество агентов-муравьев, каждый из которых представляет потенциальное решение задачи. Эти агенты перемещаются в дискретном пространстве поиска, обычно представленном в виде графа, где вершины соответствуют элементам задачи, а ребра — возможным переходам между ними. Ключевая особенность алгоритма заключается в использовании искусственных феромонов, которые обновляются в процессе работы. Муравьи, выполняя переходы между вершинами графа, оставляют феромонный след, интенсивность которого зависит от качества найденного решения. Каждый муравей строит свое решение последовательно, переходя от одной вершины графа к другой на основе вероятностного правила. В ходе работы алгоритма реализуется механизм испарения феромона, который предотвращает его чрезмерное накопление и позволяет избежать преждевременной сходимости. Также обновление феромона способствует усилению тех маршрутов, которые найдены наиболее успешными муравьями [7].

Обобщая выше сказанное, алгоритм состоит из следующих ключевых шагов:

1. Инициализация начальных параметров (количество муравьёв, коэффициент испарения феромона и другие).
2. Построение решения каждым муравьем (Формула 4.4.1).
3. Уменьшение концентрации феромона на рёбрах графа (Формула 4.4.2).
4. Добавление феромона на рёбра, которые участвовали в построенных маршрутах (Формулы 4.4.3–4.4.4).
5. Завершение работы, если критерий останова выполнен, иначе возврат к пункту номер два.

4.2 Постановка задачи

Цель работы: реализовать задачу коммивояжера муравьиным алгоритмом для нахождения приближённого оптимального маршрута.

Задачи: изучить муравьиный алгоритм, выбрать предметную область для задачи коммивояжера, произвести ручной расчёт двух итераций алгоритма для двух муравьёв, разработать программную реализацию муравьиного алгоритма для задачи коммивояжера.

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.

4.3 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.

Представленная модель ориентированного взвешенного графа не только абстрактно описывает структуру задачи, но и отражает её практическое содержание.

Для ручного расчёта число университетов в задаче взято равным шести. С начальной стартовой точкой в отеле граф содержит семь вершин. Количество возможных маршрутов для рассматриваемой задачи может быть рассчитано по Формуле 4.3.1.

$$(n - 1)! = (7 - 1)! = 6! = 1 * 2 * 3 * 4 * 5 * 6 = 720, \quad (4.3.1)$$

где n – количество вершин в полном взвешенном ориентированном графе.

Будем считать, что между двумя вершинами в описанном графе существует условно два ребра, каждое из которых представлено уникальным весом. В таком случае, число рёбер, вес которых необходимо заполнить, рассчитано по Формуле 4.3.2.

$$E = n * (n - 1) = 7 * 6 = 42, \quad (4.3.2)$$

где E – количество рёбер в полном взвешенном ориентированном графе.

Местоположения приёмных комиссий университетов и отеля представлены их реальными адресами. Адрес объекта представлен названием города, наименованием улицы, номером дома (строения). Допустимо указание

почтового индекса. В качестве отправной точки выбран отель «Звёзды Арбата». Информация о пунктах маршрута отображена в Таблице 4.3.1.

Таблица 4.3.1 – 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.

Сведения о маршрутах могут быть получены из любого другого навигационного сервиса, предоставляющего актуальную информацию о времени в пути.

Рассчитанные длины рёбер сведены в Таблицу 4.3.2 с указанием сокращённых названий вершин, составляющих ребро.

Таблица 4.3.2 – 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 4.3.1.

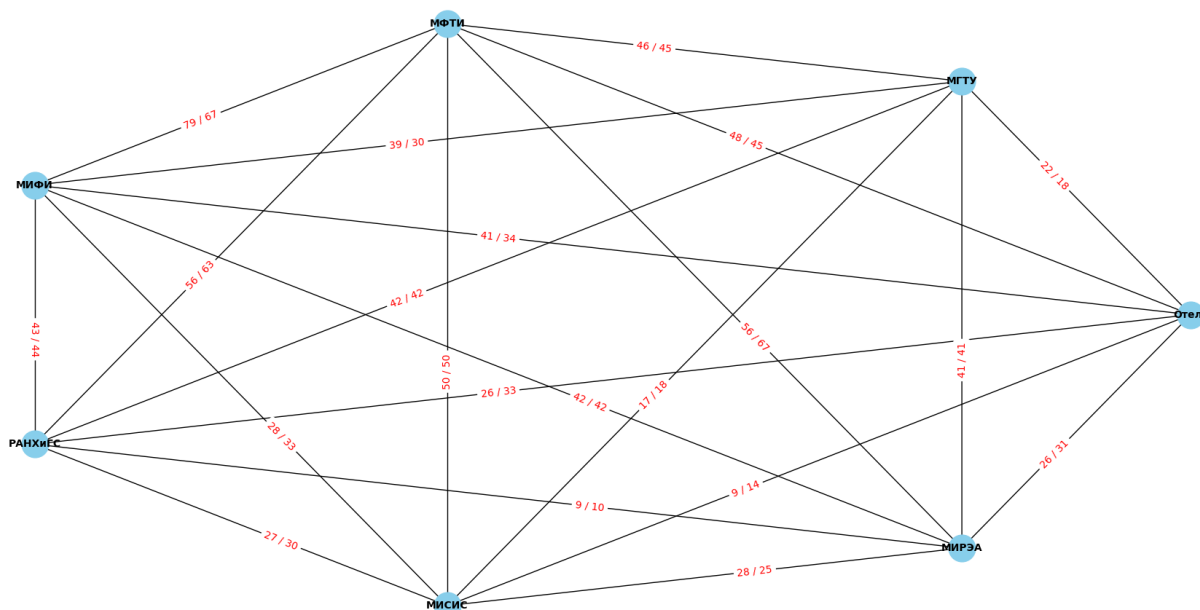


Рисунок 4.3.1 – Граф модели задачи

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

4.4 Ручной расчёт

Коэффициент α принят равным 1,1. Константа p , определяющая скорость испарения равна 0,25. Феромоны для рёбер инициализированы случайными значениями от нуля до единицы. Начальной точкой для всех муравьёв является «Отель».

Ниже приведены случайно заполненные концентрации феромонов для каждой дуги, соединяющий отель с университетами:

$$\tau_{01}(0) = 0.8329;$$

$$\tau_{02}(0) = 0.7895;$$

$$\tau_{03}(0) = 0.8434;$$

$$\tau_{04}(0) = 0.3393;$$

$$\tau_{05}(0) = 0.0015;$$

$$\tau_{06}(0) = 0.8038,$$

где τ_{ij} – концентрация феромона между вершинами (i,j),

0 – номер итерации.

В каждой вершине каждый муравей должен выбрать следующую дугу пути на основе вероятностей перехода. (Формула 4.4.1).

$$p_{ij}^k(t) = \frac{\tau_{ij}^\alpha(t)}{\sum_{j \in N_i^k} \tau_{ij}^\alpha(t)}, \quad (4.4.1)$$

где p – вычисленная вероятность;

k – номер муравья;

i – индекс вершины отправления;

j – индекс вершины назначения;

t – номер итерации алгоритма;

τ_{ij} – концентрация феромона между вершинами (i,j);

α – положительная константа, которая определяет влияние концентрации феромона;

N_i^k – множество возможных вершин, связанных с вершиной с индексом i .

Ниже рассчитана вероятность перехода из вершины под номером 0 (Отель) в вершины, которые не посещены для первого муравья на первой итерации:

$$p_{01}^1(0) = \frac{\tau_{01}^\alpha(0)}{\sum_{j \in N_0^1} \tau_{0j}^\alpha(0)} = \frac{0,8329^{1,1}}{3,5098} = \frac{0,8178}{3,5098} = 0,2330$$

$$p_{02}^1(0) = \frac{\tau_{02}^\alpha(0)}{\sum_{j \in N_0^2} \tau_{0j}^\alpha(0)} = \frac{0,7895^{1,1}}{3,5098} = \frac{0,7710}{3,5098} = 0,2196$$

$$p_{03}^1(0) = \frac{\tau_{03}^\alpha(0)}{\sum_{j \in N_0^3} \tau_{0j}^\alpha(0)} = \frac{0,8434^{1,1}}{3,5098} = \frac{0,8291}{3,5098} = 0,2362$$

$$p_{04}^1(0) = \frac{\tau_{04}^\alpha(0)}{\sum_{j \in N_0^4} \tau_{0j}^\alpha(0)} = \frac{0,3393^{1,1}}{3,5098} = \frac{0,3045}{3,5098} = 0,0867$$

$$p_{05}^1(0) = \frac{\tau_{05}^\alpha(0)}{\sum_{j \in N_0^5} \tau_{0j}^\alpha(0)} = \frac{0,0015^{1,1}}{3,5098} = \frac{0,0007}{3,5098} = 0,0002$$

$$p_{06}^1(0) = \frac{\tau_{06}^\alpha(0)}{\sum_{j \in N_0^6} \tau_{0j}^\alpha(0)} = \frac{0,8038^{1,1}}{3,5098} = \frac{0,7864}{3,5098} = 0,2240$$

Далее генерируется случайное число $r = 0,3714$. Объявляется переменная-счётчик, которая будет хранить нарастающую сумму нескольких вероятностей $p_{ij}^k(t)$. Как только полученная сумма превысит сгенерированное число, будет принят узел под номером j . На первой итерации сумма равна 0,2330. Значит, вершина под номером 1 отбрасывается. На следующей итерации сумма равна 0,4526, что больше, чем сгенерированное число. Таким образом, вторая вершина, в которую направится муравей, имеет номер 2.

Вершина заносится в список посещённых и продолжается построение маршрута таким же образом.

Построенный маршрут для первого муравья на первой итерации представлен в Листинге 4.4.1.

Листинг 4.4.1 – Маршрут первого муравья на первой итерации

ОТЕЛЬ -> МФТИ -> МИФИ -> РАНХиГС -> МГТУ -> МИСИС -> МИРЭА -> ОТЕЛЬ

Стоимость построенного маршрута: $48 + 79 + 43 + 42 + 17 + 28 + 31 = 288$.

Аналогичным образом построен маршрут для второго муравья на первой итерации (Листинг 4.4.2).

Листинг 4.4.2 – Маршрут второго муравья на первой итерации

ОТЕЛЬ -> МИФИ -> МГТУ -> МИРЭА -> РАНХиГС -> МФТИ -> МИСИС -> ОТЕЛЬ

Стоимость построенного маршрута: $41 + 30 + 41 + 10 + 63 + 50 + 14 = 249$.

В конце итерации уменьшается концентрация феромона на дугах графа в соответствии с Формулой 4.4.2.

$$\tau_{ij}(t + 1) = (1 - p)\tau_{ij}(t) \quad (4.4.2)$$

где p – константа, определяющая скорость испарения.

Процесс уменьшения концентрации феромона для рёбер от вершины под номером нуль (ОТЕЛЬ) до всех университетов описан ниже:

$$\begin{aligned}
\tau_{01}(1) &= (1 - 0,25)\tau_{01}(0) = 0,75 * 0,8329 = 0,6246 \\
\tau_{02}(1) &= (1 - 0,25)\tau_{02}(0) = 0,75 * 0,7895 = 0,5921 \\
\tau_{03}(1) &= (1 - 0,25)\tau_{03}(0) = 0,75 * 0,8434 = 0,6325 \\
\tau_{04}(1) &= (1 - 0,25)\tau_{04}(0) = 0,75 * 0,3395 = 0,2545 \\
\tau_{05}(1) &= (1 - 0,25)\tau_{05}(0) = 0,75 * 0,0015 = 0,0011 \\
\tau_{06}(1) &= (1 - 0,25)\tau_{06}(0) = 0,75 * 0,8038 = 0,6028
\end{aligned}$$

Муравей помечает свой пройденный путь, отложив на каждой дуге феромон в соответствие с Формулой 4.4.3.

$$\Delta\tau_{ij}^k(t) = \frac{1}{L^k(t)}, \quad (4.4.3)$$

где L^k – длина пути, построенная муравьём под номером k на итерации t.

Для каждой дуги графа концентрация феромона определяется по Формуле 4.4.4.

$$\tau_{ij}(t + 1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t), \quad (4.4.4)$$

где n_k – число муравьёв.

Маршрут второго муравья может быть задан с использованием порядковых номеров вершин (Листинг 4.4.3).

Листинг 4.4.3 – Маршрут второго муравья на первой итерации, заданным индексами

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

Процесс обновления концентрации феромона на дугах, по которым прополз второй муравей представлен ниже:

$$\begin{aligned}
\tau_{03}(1) &= \tau_{03}(0) + \sum_{k=1}^{n_k} \Delta\tau_{03}^k(0) = 0,6325 + \frac{1}{249} = 0,6365; \\
\tau_{31}(1) &= \tau_{31}(0) + \sum_{k=1}^{n_k} \Delta\tau_{31}^k(0) = 0,2175 + \frac{1}{249} = 0,2215;
\end{aligned}$$

$$\tau_{16}(1) = \tau_{16}(0) + \sum_{k=1}^{n_k} \Delta\tau_{16}^k(0) = 0,7071 + \frac{1}{249} = 0,7111;$$

$$\tau_{64}(1) = \tau_{64}(0) + \sum_{k=1}^{n_k} \Delta\tau_{64}^k(0) = 0,3531 + \frac{1}{249} = 0,3571;$$

$$\tau_{42}(1) = \tau_{42}(0) + \sum_{k=1}^{n_k} \Delta\tau_{42}^k(0) = 0,7092 + \frac{1}{249} = 0,7133;$$

$$\tau_{25}(1) = \tau_{25}(0) + \sum_{k=1}^{n_k} \Delta\tau_{25}^k(0) = 0,6243 + \frac{1}{249} = 0,6283;$$

$$\tau_{50}(1) = \tau_{50}(0) + \sum_{k=1}^{n_k} \Delta\tau_{50}^k(0) = 0,0186 + \frac{1}{249} = 0,0226;$$

Таким образом происходит обновление концентрации феромонов для каждой дуги, которая участвовала в маршрутах муравьёв.

Под конец первой итерации лучший путь найден вторым муравьём, и его длина составляет 249.

На второй итерации путь, построенный первым муравьём представлен в Листинге 4.4.4.

Листинг 4.4.4 – Маршрут первого муравья на второй итерации

ОТЕЛЬ -> МГТУ -> РАНХиГС -> МФТИ -> МИРЭА -> МИСИС -> МИФИ -> ОТЕЛЬ

Длина пути: $22 + 42 + 63 + 56 + 25 + 33 + 34 = 275$.

Путь, проложенный вторым муравьём представлен в Листинге 4.4.5.

Листинг 4.4.5 – Маршрут второго муравья на второй итерации

ОТЕЛЬ -> МГТУ -> МФТИ -> МИСИС -> МИРЭА -> МИФИ -> РАНХиГС -> ОТЕЛЬ

Длина пути: $22 + 46 + 50 + 28 + 42 + 43 + 33 = 264$.

Процесс испарения и обновления концентрации феромона остаётся прежним. Муравьи выбрали худшие маршруты в силу недостаточной концентрации феромона после первой итерации. С ростом количества муравьёв-агентов, числа итераций, а также уменьшением коэффициента, определяющего скорость испарения, алгоритм будет сходиться к одному маршруту.

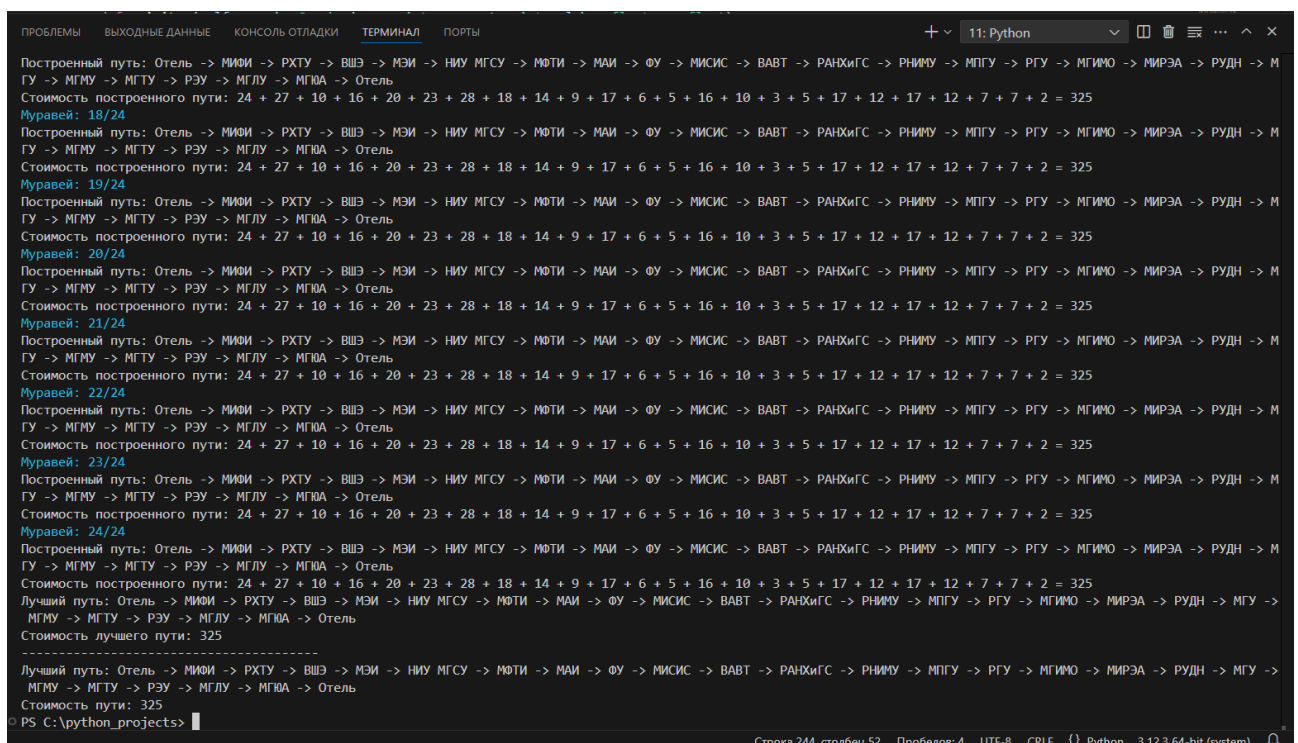
4.5 Программная реализация

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. Для реализации муравьиного алгоритма написан отдельный класс.

Код простого муравьиного алгоритма для задачи коммивояжера представлен в Приложении Г.

Результат работы муравьиного алгоритма представлен на Рисунке 4.5.1.



```
ПРОБЛЕМЫ Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ
+ 11: Python
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 18/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 19/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 20/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 21/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 22/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 23/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 24/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Лучший путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость лучшего пути: 325
-----
Лучший путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость пути: 325
PS C:\python_projects>
```

Рисунок 4.5.1 – Результат работы муравьиного алгоритма для задачи коммивояжера

Муравьи сходятся к одному маршруту, который является наилучшим для выбранного графа.

5 ПЧЕЛИНЫЙ АЛГОРИТМ

5.1 Описание алгоритма

В алгоритме участвуют искусственные агенты, которые исследуют пространство решений, оценивают их качество и концентрируются на наиболее перспективных областях, улучшая решения итерационно.

Процесс оптимизации начинается с инициализации множества случайных точек, представляющих начальные решения. Затем выполняются итерации, каждая из которых состоит из нескольких фаз: исследование новых решений разведчиками, поиск локальных улучшений собирателями и сосредоточение усилий на наиболее перспективных направлениях. Если в течение заданного числа итераций улучшений не наблюдается, алгоритм завершает работу [8].

Обобщая выше сказанное, алгоритм состоит из следующих ключевых шагов:

1. Инициализация начальных параметров: количества пчел-разведчиков, максимального расстояния для объединения точек, размера локальной области поиска и других.
2. Случайное размещение пчел-разведчиков в пространстве решений (Формула 5.3.1).
3. Оценка значений целевой функции для каждой точки.
4. Выделение областей, основанных на расстоянии между точками, и определение точек с наибольшим значением функции в каждой области (Формула 5.3.3).
5. Локальный поиск новых решений в выделенных областях с учётом текущего оптимального значения.
6. Сравнение новых решений с текущими лучшими. При обнаружении лучшего значения обновляется глобальное решение.
7. Проверка условия остановки. Если улучшений не наблюдается в течение заданного числа итераций или достигнуто максимальное

число итераций, алгоритм завершает работу; иначе возвращается к шагу четыре.

5.2 Постановка задачи

Цель работы: реализовать преобразование Коши методом пчелиной колонии для нахождения приближённого глобального минимума функции.

Задачи: изучить пчелиный алгоритм, выбрать тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт итерации алгоритма, разработать программную реализацию пчелиного алгоритма для задачи минимизации функции.

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 5.2.1).

$$f(x, y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]. \quad (5.2.1)$$

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

5.3 Математическая модель

Количество пчел-разведчиков равно S . Случайным образом генерируется S точек, куда отправляются пчелы (Формула 5.3.1).

$$X_{N,K} \in D, \quad (5.3.1)$$

где N – номер пчелы-разведчика, $N \in [1:S]$;

K – номер итерации;

D – область поиска.

Каждая точка представлена своими координатами (Формула 5.3.2).

$$X = (x_1, x_2, \dots, x_n), \quad (5.3.2)$$

где n – номер координаты.

Подобласти, в которые объединяются точки формируются на основе Евклидова расстояния между точками. Для точек $A = (x_1, x_2, \dots, x_n)$ и $B = (y_1, y_2, \dots, y_n)$ евклидово расстояние считается по Формуле 5.3.3.

$$d(A, B) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2} \quad (5.3.3)$$

где d – рассчитанное расстояние.

5.4 Ручной расчёт

Количество пчел-разведчиков: $S = 10$. Область локального поиска: $l = 0,5$.

Допустимое Евклидово расстояние для объединения точек: $d = 1,5$.

Случайным образом размещено S точек в области поиска D . Координаты точек представлены ниже:

$$X_{1,0} = (1,082, 1,309);$$

$$X_{2,0} = (1,382, -1,130);$$

$$X_{3,0} = (-0,410, -1,284);$$

$$\begin{aligned}
X_{4,0} &= (0,751, 0,690); \\
X_{5,0} &= (0,007, 1,862); \\
X_{6,0} &= (0,006, 1,522); \\
X_{7,0} &= (-1,153, 0,236); \\
X_{8,0} &= (-0,922, -0,675); \\
X_{9,0} &= (0,023, -0,661); \\
X_{10,0} &= (1,067, 1,370).
\end{aligned}$$

Рассчитанные значения целевой функции во всех точках представлены ниже:

$$\begin{aligned}
F(X_{1,0}) &= 6454,259; \\
F(X_{2,0}) &= 32906,002; \\
F(X_{3,0}) &= 78,617; \\
F(X_{4,0}) &= 1216,079; \\
F(X_{5,0}) &= 176789,654; \\
F(X_{6,0}) &= 97662,703; \\
F(X_{7,0}) &= 1088,499; \\
F(X_{8,0}) &= 564,689; \\
F(X_{9,0}) &= 102,551; \\
F(X_{10,0}) &= 8889,103.
\end{aligned}$$

Наименьшее значение целевая функция имеет в точке $F(X_{3,0})$. Глобальный минимум функции обновляется и становится равным 78,617. Точка, в которой достигается глобальный минимум на текущей итерации: $(-0,410, -1,284)$.

Далее рассчитывается евклидово расстояние между точками для объединения их в области по Формуле 5.3.3:

$$d(X_{1,0}, X_{2,0}) = \sqrt{(1,082 - 1,382)^2 + (1,309 + 1,130)^2} = 2,458 > 1,5$$

Расстояние больше допустимого, поэтому точки не объединяются в одну область.

$$d(X_{1,0}, X_{3,0}) = \sqrt{(1,082 + 0,410)^2 + (1,309 + 1,284)^2} = 2,994 > 1,5$$

$$d(X_{1,0}, X_{4,0}) = \sqrt{(1,082 - 0,751)^2 + (1,309 - 0,690)^2} = 0,702 < 1,5$$

Получено расстояние, меньшее допустимого. Первая и четвёртая точки объединяются в одну область. На текущей итерации четвёртая точка больше не может быть размещена в другую область.

$$d(X_{1,0}, X_{5,0}) = \sqrt{(1,082 - 0,007)^2 + (1,309 - 1,862)^2} = 1,209 < 1,5$$

$$d(X_{1,0}, X_{6,0}) = \sqrt{(1,082 - 0,006)^2 + (1,309 - 1,522)^2} = 1,096 < 1,5$$

$$d(X_{1,0}, X_{7,0}) = \sqrt{(1,082 + 1,153)^2 + (1,309 - 0,236)^2} = 2,480 > 1,5$$

$$d(X_{1,0}, X_{8,0}) = \sqrt{(1,082 + 0,922)^2 + (1,309 + 0,675)^2} = 2,821 > 1,5$$

$$d(X_{1,0}, X_{9,0}) = \sqrt{(1,082 - 0,023)^2 + (1,309 + 0,661)^2} = 2,237 > 1,5$$

$$d(X_{1,0}, X_{10,0}) = \sqrt{(1,082 - 1,067)^2 + (1,309 - 1,370)^2} = 0,062 < 1,5$$

В первую область попадают точки $(X_{1,0}, X_{4,0}, X_{5,0}, X_{6,0}, X_{10,0})$.

Точки, распределённые в область, не участвуют в дальнейшем переборе.

Переходим к созданию второй области. Евклидово расстояние для следующих комбинаций точек рассчитано ниже:

$$d(X_{2,0}, X_{3,0}) = \sqrt{(1,382 + 0,410)^2 + (-1,130 + 1,284)^2} = 1,799 > 1,5$$

$$d(X_{2,0}, X_{7,0}) = \sqrt{(1,382 + 1,153)^2 + (-1,130 - 0,236)^2} = 2,880 > 1,5$$

$$d(X_{2,0}, X_{8,0}) = \sqrt{(1,382 + 0,922)^2 + (-1,130 + 0,675)^2} = 2,349 > 1,5$$

$$d(X_{2,0}, X_{9,0}) = \sqrt{(1,382 - 0,023)^2 + (-1,130 + 0,661)^2} = 1,437 < 1,5$$

Во вторую область попадают точки $(X_{2,0}, X_{9,0})$. Дальнейший расчёт:

$$d(X_{3,0}, X_{7,0}) = \sqrt{(-0,410 + 1,153)^2 + (-1,284 - 0,236)^2} = 1,692 > 1,5$$

$$d(X_{3,0}, X_{8,0}) = \sqrt{(-0,410 + 0,922)^2 + (-1,284 + 0,675)^2} = 0,796 < 1,5$$

В третью область попадают точки $(X_{3,0}, X_{8,0})$. Остаётся единственная точка, не распределённая в область $X_{7,0}$. Эта точка представляет четвёртую область и является единственной точкой в своей области.

В каждой области выбрана точка, в которой функция имеет наименьшее значение.

Для первой области центральная точка с минимальным значением целевой функции $X_{4,0}$. Определена область поиска в первой области: $0,251 < x < 1,251$; $0,190 < y < 1,190$.

Генерируется S-1 точек в подобласти поиска, которые обозначаются в соответствии с Формулой 5.4.1.

$$X_{N,K}^M \in D, \quad (5.4.1)$$

где N – номер пчелы-разведчика, $N \in [1: S - 1]$;

K – номер итерации;

M – номер подобласти;

D – подобласть поиска.

Сгенерированные точки представлены ниже (первая точка – центра подобласти):

$$X_{1,0}^1 = (0,751, 0,690);$$

$$X_{2,0}^1 = (0,714, 1,161);$$

$$X_{3,0}^1 = (0,713, 1,013);$$

$$X_{4,0}^1 = (1,216, 0,739);$$

$$X_{5,0}^1 = (0,931, 0,365);$$

$$X_{6,0}^1 = (0,861, 1,037);$$

$$X_{7,0}^1 = (0,759, 0,749);$$

$$X_{8,0}^1 = (0,863, 1,117);$$

$$X_{9,0}^1 = (1,238, 0,996);$$

$$X_{10,0}^1 = (1,070, 0,636).$$

Посчитаны значения целевой функции в выбранных точках:

$$F(X_{1,0}^1) = 1216,079;$$

$$F(X_{2,0}^1) = 8434,740;$$

$$F(X_{3,0}^1) = 4775,814;$$

$$F(X_{4,0}^1) = 860,193;$$

$$F(X_{5,0}^1) = 1103,519;$$

$$F(X_{6,0}^1) = 3347,560;$$

$$F(X_{7,0}^1) = 1423,953;$$

$$F(X_{8,0}^1) = 4680,991;$$

$$F(X_{9,0}^1) = 1084,610;$$

$$F(X_{10,0}^1) = 897,479.$$

Новая точка для первой подобласти с минимальным значением функции
 $X_{4,0}^1 = 860,193.$

Аналогичным образом выбирается лучшая точка в каждой выделенной подобласти.

Лучшее значение функции: 78,61; Лучшая точка: $X = (-0,410, -1,284).$

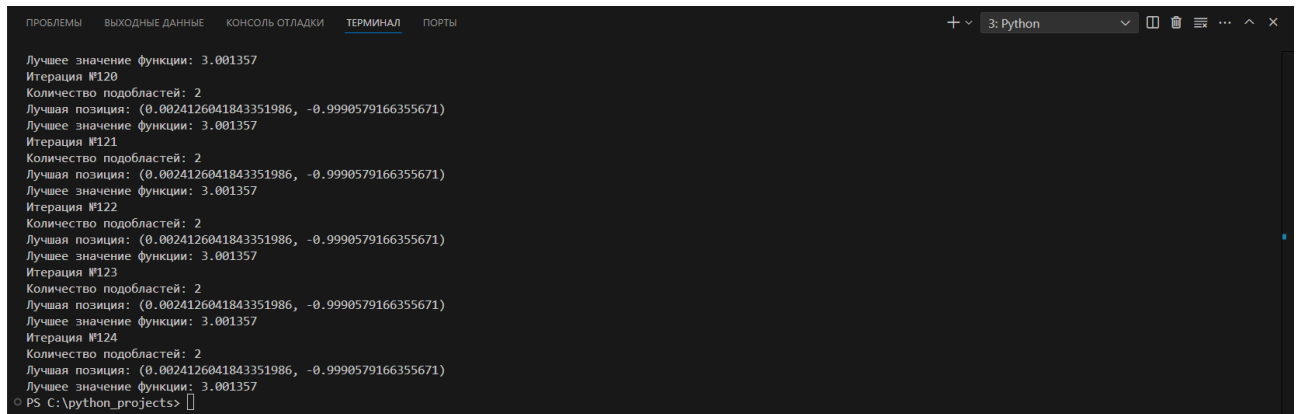
5.5 Программная реализация

Разработан класс BeeColony, представляющий колонию пчёл.

Код алгоритма пчелиной колонии для задачи поиска глобального минимума функции представлен в Приложении Д.

Количество пчел $S = 100$, количество итераций без улучшения до остановки алгоритма $k = 100$, максимальное евклидово расстояние $d = 1,5$, область локального поиска $l = 0,5$.

Результат работы алгоритма пчелиной колонии представлен на Рисунке 5.5.1.



```
ПРОБЛЕМЫ Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ
Лучшее значение функции: 3.001357
Итерация №120
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №121
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №122
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №123
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
Итерация №124
Количество подобластей: 2
Лучшая позиция: (0.0024126041843351986, -0.9990579166355671)
Лучшее значение функции: 3.001357
PS C:\python_projects>
```

Рисунок 5.5.1 – Результат работы алгоритма пчелиной колонии для задачи поиска глобального минимума функции

Визуализация процесса поиска представлена на Рисунке 5.5.2.

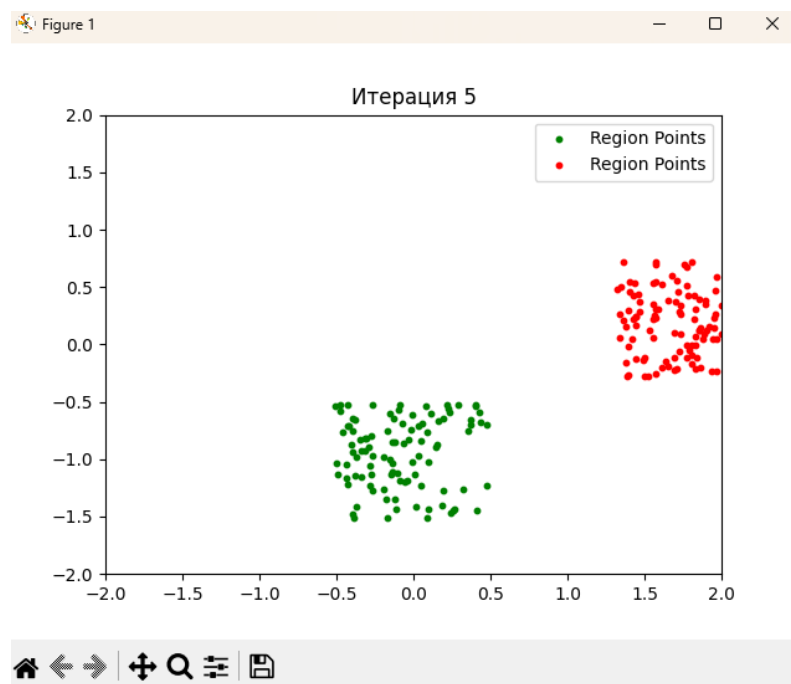


Рисунок 5.5.2 – Визуализация работы алгоритма пчелиной колонии

6 АЛГОРИТМ РОЯ СВЕТЛЯЧКОВ

6.1 Описание алгоритма

Алгоритм роя светлячков использует искусственных агентов (светлячков), которые взаимодействуют друг с другом, перемещаясь в поисковом пространстве в направлении более качественных решений. Каждый светлячок характеризуется своей позицией, которая соответствует возможному решению задачи, и уровнем «свечения» (яркости), зависящим от значения целевой функции в данной позиции.

Оптимизация начинается с инициализации популяции светлячков, которые случайным образом размещаются в пространстве решений. Затем алгоритм выполняет итерации, каждая из которых включает следующие этапы:

1. Обновление яркости. Уровень свечения каждого светлячка пересчитывается в зависимости от значения целевой функции в его текущей позиции. Это позволяет определить, насколько «привлекателен» светлячок для других.
2. Формирование окрестности. Для каждого светлячка определяется список соседей — агентов, находящихся в пределах заданного радиуса, которые обладают более высокой яркостью.
3. Выбор соседа и перемещение. Светлячок перемещается в направлении одного из соседей, вероятность выбора которого пропорциональна его яркости. Если в окрестности нет более ярких соседей, светлячок перемещается случайным образом.
4. Модификация радиуса окрестности. Радиус видимости светлячка корректируется в зависимости от количества соседей, чтобы адаптироваться к текущей плотности агентов в пространстве.
5. Обновление глобального решения. Если положение какого-либо светлячка превосходит текущее лучшее значение, обновляется глобальное решение.

Процесс продолжается до тех пор, пока не выполнено условие остановки: либо достигнуто максимальное количество итераций, либо отсутствуют значительные улучшения в течение заданного числа шагов [10].

6.2 Постановка задачи

Цель работы: реализовать преобразование Коши методом роя светлячков для нахождения приближённого глобального минимума функции.

Задачи: изучить алгоритм роя светлячков, выбрать тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт итерации алгоритма, разработать программную реализацию алгоритма роя светлячков для задачи минимизации функции.

Нахождение глобального минимума функции от многих переменных состоит в поиске точки в многомерном пространстве, где значение функции будет минимальным.

Выбранная функция для оптимизации: функция Гольдшейна-Прайса (Формула 6.2.1).

$$f(x, y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]. \quad (6.2.1)$$

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

6.3 Математическая модель

Алгоритм имеет следующие входные параметры: β — коэффициент изменения радиуса окрестности; ρ — коэффициент уменьшения уровня люциферина; δ — коэффициент изменения позиции; r_0 — начальный радиус окрестности; N — максимальное количество итераций алгоритма; K — размер

популяции светлячков; γ — коэффициент привлекательности светлячков; x_{min}, x_{max} — минимальные и максимальные границы пространства; m — длина вектора позиции агента [11].

Позиция каждого светлячка инициализируется случайным образом (Формула 6.3.1).

$$x_k = (x_{k1}, x_{k2}, \dots, x_{km}), x_{kj} = x_j^{min} + (x_j^{max} - x_j^{min})rand(), \quad (6.3.1)$$

где k — номер агента.

Изначально все светлячки имеют одинаковое количество люциферина $l_k = l_0$. Радиус окрестности также инициализируется предварительно заданным значением $r_k = r_0$.

Обновление уровня люциферина зависит от позиции агента в пространстве (значения его целевой функции). Вычисление уровня люциферина осуществляется по Формуле 6.3.2.

$$l_k(t + 1) = (1 - \rho)l_k(t) + \gamma F(x_k)(t + 1), \quad (6.3.2)$$

где l — количество люциферина;

k — номер агента;

t — номер итерации.

Каждый агент выбирает того агента внутри радиуса окрестности поиска, у которого уровень люциферина выше, чем его собственный. Окрестность светлячка определяется в соответствие с Формулой 6.3.3.

$$U_k = \{m | |x_m - x_k| < r_k, l_k < l_m, m \in \overline{1, K}\}, \quad (6.3.3)$$

где U_k — окрестность светлячка;

m — светлячок в окрестности светлячка k .

Таким образом, окрестность U_k включает светлячков m , которые находятся в пределах радиуса r_k ($\|x_m - x_k\| < r_k$), имеют уровень люциферина выше, чем у светлячка k ($l_k < l_m$).

Вычисление вероятности перемещения к соседям осуществляется по Формуле 6.3.4.

$$P_{km} = \frac{l_m - l_k}{\sum_{s \in U_k} (l_s - l_k)}, m \in 1, K, \quad (6.3.4)$$

где P_{km} – вычисленная вероятность движения светлячка k к соседу m .

Говоря иначе, P_{km} показывает относительную привлекательность соседа m по сравнению с остальными соседями.

Светлячок k выбирает номер соседа m в своей окрестности, используя метод колеса рулетки (Формула 6.3.5).

$$\text{Если } \sum_{s=1}^{c-1} P_{ks} < rand() \leq \sum_{s=1}^c P_{ks}, \text{ то } m = c, \quad (6.3.5)$$

где $rand()$ – случайное число в интервале $[0;1]$.

Обновленная позиция агента k определяется по Формуле 6.3.6.

$$x_k(t+1) = x_k(t) + \delta \frac{x_m(t) - x_k(t)}{\|x_m(t) - x_k(t)\|}, \quad (6.3.6)$$

где $x_k(t+1)$ – новая позиция агента;

В числителе второго слагаемого записан вектор смещения, направленный от текущей позиции светлячка k к соседу m . В знаменателе дроби записана длина этого вектора (евклидово расстояние). Это нормализующий фактор, чтобы перемещение происходило по направлению, но не зависело от расстояния.

Обновление радиуса окрестности r осуществляется по Формуле 6.3.7.

$$r_k = \min(r_{\max}, \max(r_{\min}, r_k + \beta(n_u - |U_k|))), \quad (6.3.7)$$

где r_k – обновленный радиус окрестности для светлячка k ;

n_u – желаемое количество соседей для светлячка k ;

$|U_k|$ – текущее количество соседей светлячка k в его окрестности.

В конце итерации определяется наименьшее значение функции, обновляется глобальный минимум, если найдено значение, меньшее текущего глобального минимума.

6.4 Ручной расчёт

Коэффициент изменения радиуса окрестности $\beta = 0,6$; коэффициент уменьшения уровня люциферина $\rho = 0,4$; коэффициент изменения позиции $\delta = 0,2$; начальный радиус окрестности $r_0 = 0,5$; размер популяции светлячков $K = 6$; коэффициент привлекательности светлячков $\gamma = 1$.

Светлячки случайным образом размещены в гиперпространстве поиска. Начальное количество люциферина для светлячков равно 0, начальный радиус равен 0,5. Ниже представлены начальные координаты светлячков:

$$x_1(0) = (-0,1979; 0,6889);$$

$$x_2(0) = (1,5653; 0,4466);$$

$$x_3(0) = (-1,9749; 0,6136);$$

$$x_4(0) = (-1,0235; -1,6855);$$

$$x_5(0) = (1,5296; -1,2012);$$

$$x_6(0) = (-1,3872; -0,3630).$$

В начале итерации обновляется количество люциферина у каждого светлячка по Формуле 6.3.2. Расчет количества люциферина представлен ниже:

$$l_1(1) = (1 - 0,4) * 0 + 1 * 14444,18^{-1} = 6,9232 * 10^{-5};$$

$$\begin{aligned}
l_2(1) &= (1 - 0,4) * 0 + 1 * 747,40^{-1} = 0,0013; \\
l_3(1) &= (1 - 0,4) * 0 + 1 * 47626,687^{-1} = 2,0996 * 10^{-5}; \\
l_4(1) &= (1 - 0,4) * 0 + 1 * 695,491^{-1} = 0,0014; \\
l_5(1) &= (1 - 0,4) * 0 + 1 * 54225,864^{-1} = 1,8441 * 10^{-5}; \\
l_6(1) &= (1 - 0,4) * 0 + 1 * 5577,416^{-1} = 0,0001;
\end{aligned}$$

Вычисляется множество соседей для каждого светлячка по Формуле 6.3.3. Расчет представлен ниже:

$$\begin{aligned}
U_1(1) &= \{\}; \\
U_2(1) &= \{\}; \\
U_3(1) &= \{\}; \\
U_4(1) &= \{\}; \\
U_5(1) &= \{\}; \\
U_6(1) &= \{\}.
\end{aligned}$$

На первой итерации ни у одного из светлячков нет соседей.

Изменяется радиус окрестности светлячков в соответствии с Формулой 6.3.7. Расчет радиуса представлен ниже:

$$\begin{aligned}
r_1(1) &= \min(4, \max(0,1, r_1(0) + 0,6(5 - 0))) = 3,5; \\
r_2(1) &= \min(4, \max(0,1, r_2(0) + 0,6(5 - 0))) = 3,5; \\
r_3(1) &= \min(4, \max(0,1, r_3(0) + 0,6(5 - 0))) = 3,5; \\
r_4(1) &= \min(4, \max(0,1, r_4(0) + 0,6(5 - 0))) = 3,5; \\
r_5(1) &= \min(4, \max(0,1, r_5(0) + 0,6(5 - 0))) = 3,5; \\
r_6(1) &= \min(4, \max(0,1, r_6(0) + 0,6(5 - 0))) = 3,5;
\end{aligned}$$

Лучшее глобальное значение функции на первой итерации равно 695,491. Точка, в которой достигается минимальное значение: $(-1,0235; -1,6855)$.

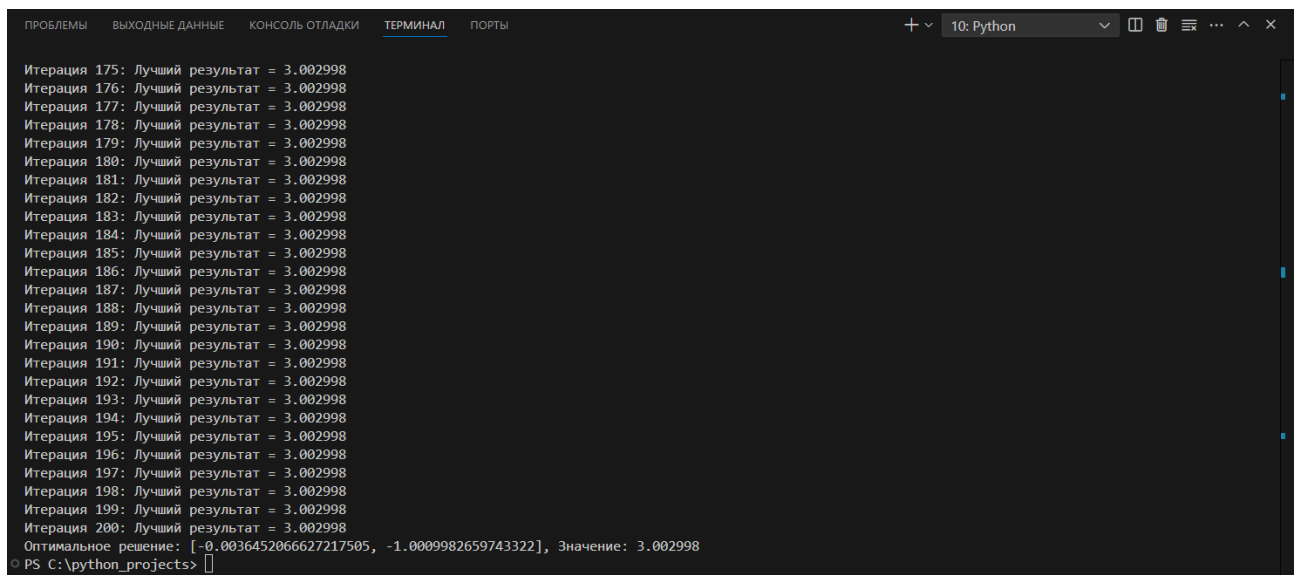
6.5 Программная реализация

Разработаны классы Firefly и FireflySwarm, которые реализуют отдельного светлячка и рой светлячков.

Код алгоритма роя светлячков для задачи поиска глобального минимума функции представлен в Приложении Е.

Коэффициент изменения радиуса окрестности $\beta = 0,6$; коэффициент уменьшения уровня люциферина $\rho = 0,4$; коэффициент изменения позиции $\delta = 0,2$; начальный радиус окрестности $r_0 = 0,5$; размер популяции светлячков $K = 100$; коэффициент привлекательности светлячков $\gamma = 1$; максимальное количество итераций $N = 200$.

Результат работы алгоритма роя светлячков представлен на Рисунке 6.5.1.



```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ
+ ~ 10: Python
Итерация 175: Лучший результат = 3.002998
Итерация 176: Лучший результат = 3.002998
Итерация 177: Лучший результат = 3.002998
Итерация 178: Лучший результат = 3.002998
Итерация 179: Лучший результат = 3.002998
Итерация 180: Лучший результат = 3.002998
Итерация 181: Лучший результат = 3.002998
Итерация 182: Лучший результат = 3.002998
Итерация 183: Лучший результат = 3.002998
Итерация 184: Лучший результат = 3.002998
Итерация 185: Лучший результат = 3.002998
Итерация 186: Лучший результат = 3.002998
Итерация 187: Лучший результат = 3.002998
Итерация 188: Лучший результат = 3.002998
Итерация 189: Лучший результат = 3.002998
Итерация 190: Лучший результат = 3.002998
Итерация 191: Лучший результат = 3.002998
Итерация 192: Лучший результат = 3.002998
Итерация 193: Лучший результат = 3.002998
Итерация 194: Лучший результат = 3.002998
Итерация 195: Лучший результат = 3.002998
Итерация 196: Лучший результат = 3.002998
Итерация 197: Лучший результат = 3.002998
Итерация 198: Лучший результат = 3.002998
Итерация 199: Лучший результат = 3.002998
Итерация 200: Лучший результат = 3.002998
Оптимальное решение: [-0.0036452066627217505, -1.0009982659743322], Значение: 3.002998
PS C:\python_projects>
```

Рисунок 6.5.1 – Результат работы алгоритма роя светлячков для задачи поиска глобального минимума функции

Для каждой итерации выводится её номер и глобальный текущий минимум функции.

Визуализация процесса поиска представлена на Рисунках 6.5.2–6.5.3.

В верхней части графика выводится номер итерации, зелёные точки соответствуют светлячкам в гиперпространстве поиска решений. Светлячки с более высоким уровнем люциферина свяжутся ярче. На вертикальной оси – значение координаты по оси y , и на горизонтальной оси – значение координаты по оси x .

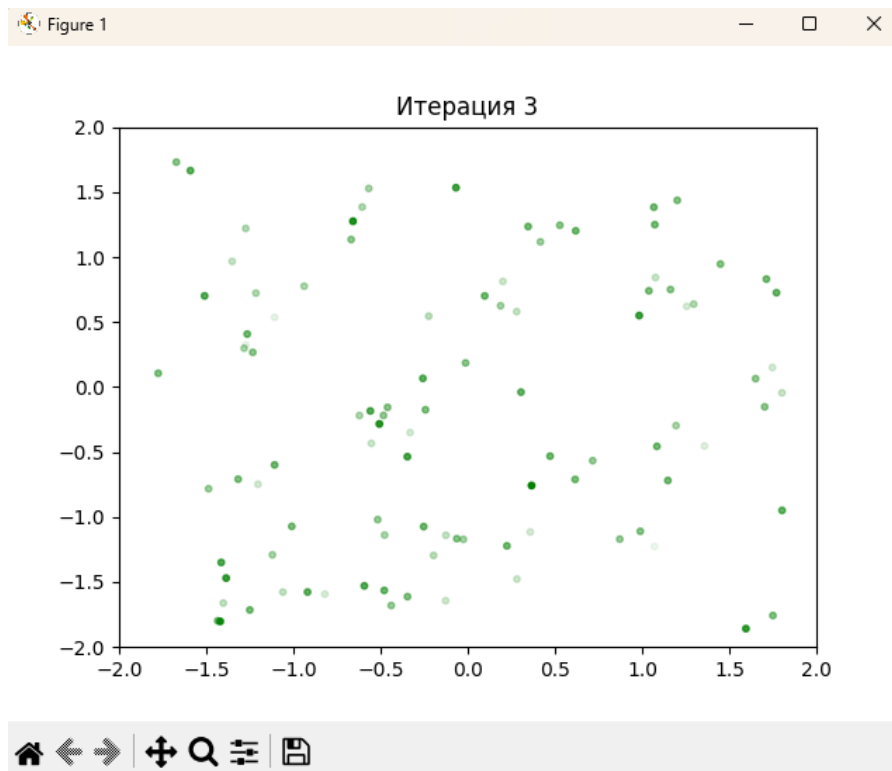


Рисунок 6.5.2 – Визуализация работы алгоритма роя светлячков на начальных итерациях

На Рисунке 6.5.2 видно, что светлячки распределены по всему гиперпространству поиска на начальных итерациях работы алгоритма.

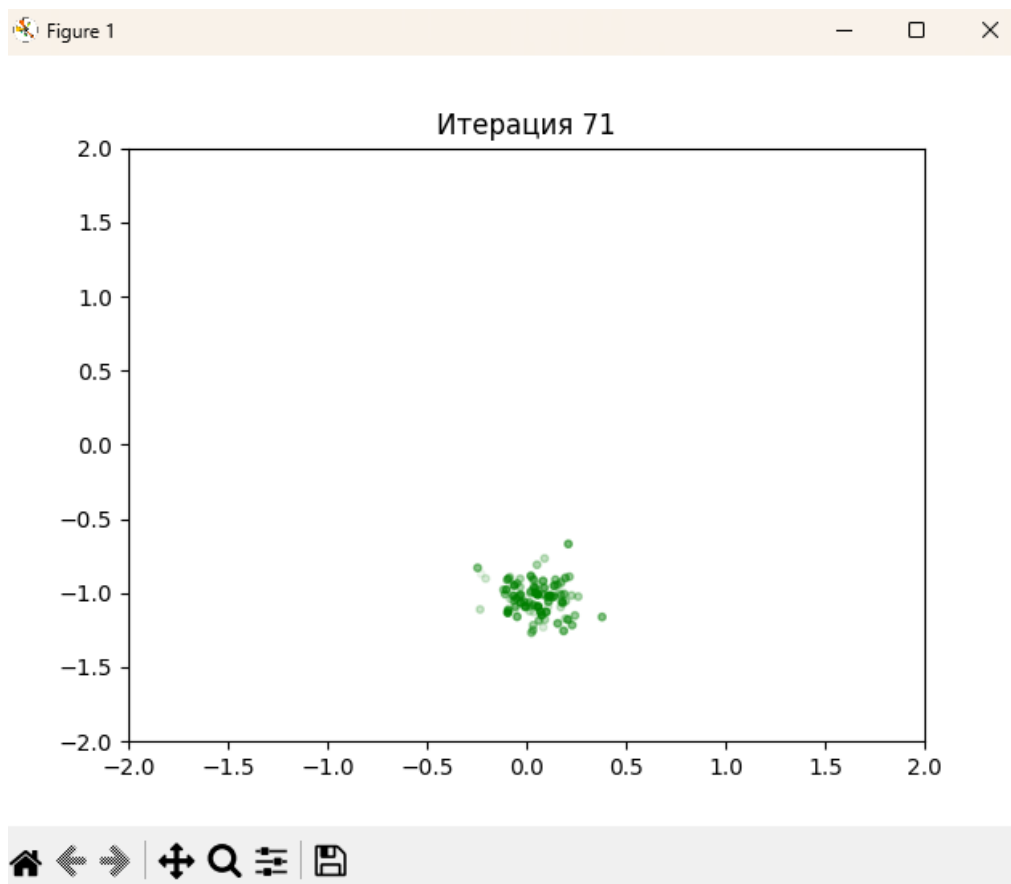


Рисунок 6.5.3 – Сходимость светлячков к глобальному минимуму

На Рисунке 6.5.3 светлячки сходятся к одной точке – глобальному минимуму рассматриваемой функции оптимизации.

Объединение светлячков гиперпространстве поиска в один кучный рой говорит о сходимости алгоритма.

График сходимости алгоритма роя светлячков представлен на Рисунке 6.5.4.

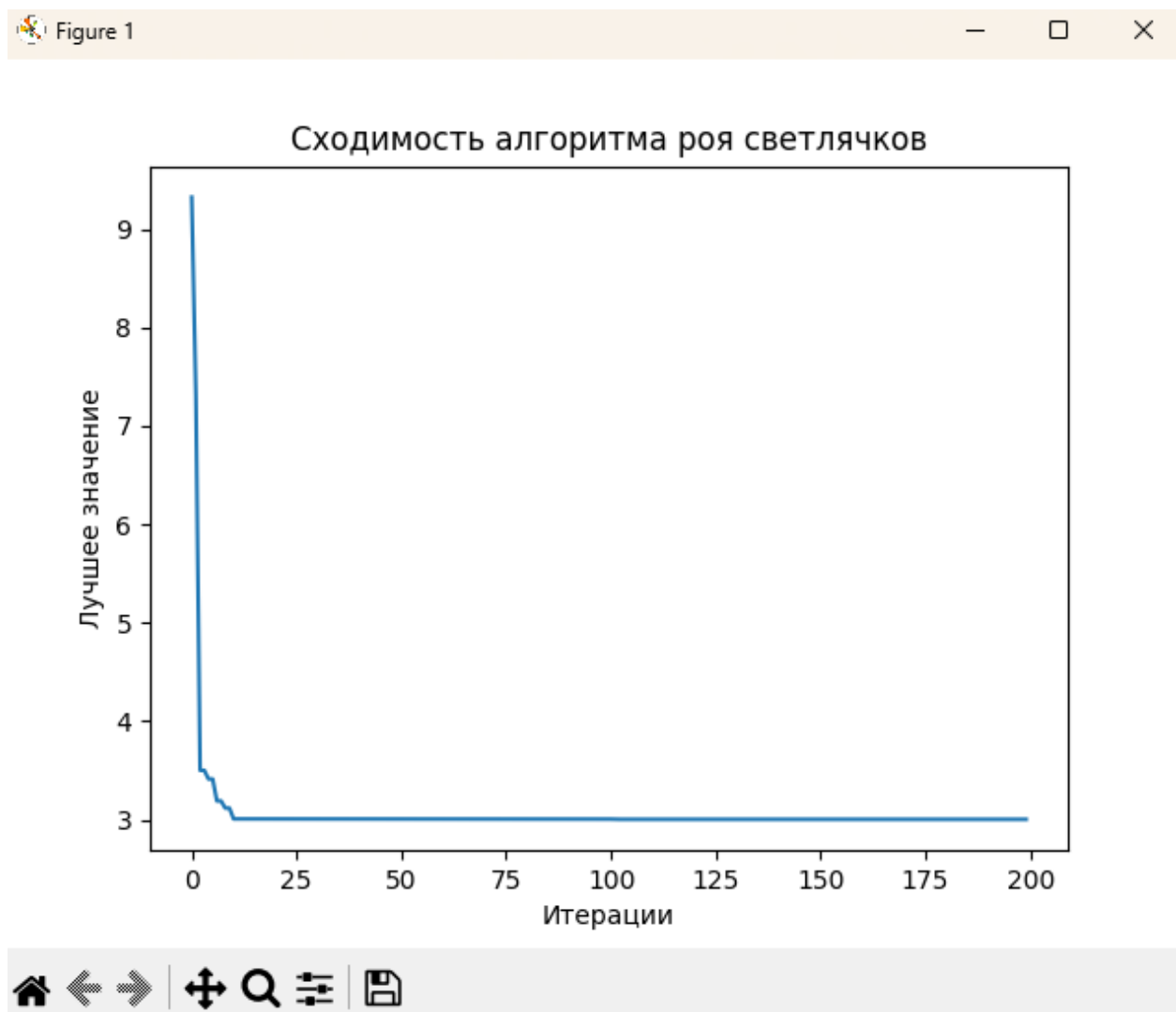


Рисунок 6.5.4 – График сходимости алгоритма роя светлячков

На горизонтальной оси отложен номер итерации, на вертикальной оси – глобальный текущий минимум функции.

По графику видно, что алгоритм находит приближённый глобальный минимум функции Гольдшейна-Прайса примерно спустя десять итераций работы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы изучены основы системного анализа данных и применения онтологий, а также изучены и реализованы различные методы оптимизации, вдохновлённые природой.

Разработана онтология киноиндустрии, включающая классы, слоты и экземпляры. Построенная модель онтологии помогает наглядно определить взаимоотношения и связи между объектами в системе. С помощью инструмента работы с онтологиями Protégé выполнены запросы на получение объектов по различным атрибутам. Написана программа на языке Python, реализующая построенную модель.

Реализованы метод имитации отжига и муравьиный алгоритм для решения NP-полной комбинаторной задачи коммивояжера с обходом университетов. Муравьиный алгоритм показывает гораздо лучший результат, чем метод имитации отжига, поскольку последний использует случайную генерацию новых решений. Однако, метод отжига гораздо проще в реализации и не требует настройки большого количества гиперпараметров. Для сходимости муравьиного алгоритма к оптимальному решению необходимо точно выбрать количество муравьев, итераций, коэффициент испарения феромона и прочие параметры.

Реализованы метод имитации отжига Коши, локальный и глобальный алгоритмы роя светлячков, пчелиный алгоритм и алгоритм роя светлячков для задачи поиска глобального минимума функции Гольдшейна-Прайса. Наиболее точными алгоритмами являются пчелиный и рой светлячков, однако за высокую точность приходится платить сложной реализацией алгоритмов с большим количеством параметров. Алгоритм роя частиц очень прост в реализации и показывает решение, близкое к решению, полученным алгоритмами пчелиной колонии и роя светлячков.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Protege. [Электронный ресурс]. URL: <https://protegewiki.stanford.edu/wiki/ProtegeDocs> (Дата обращения: 16.09.2024).
2. 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.
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: 02.11.2024).
4. Google Maps. [Электронный ресурс]. URL: <https://maps.google.com> (Дата обращения: 06.11.2024).
5. Казакова, Е. М. Краткий обзор методов оптимизации на основе роя частиц // Вест. Краунц. Физ.-мат. науки. 2022. №2. URL: <https://cyberleninka.ru/article/n/kratkiy-obzor-metodov-optimizatsii-na-osnove-roya-chastits> (Дата обращения: 11.11.2024).
6. 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.
7. Муравьиный алгоритм. [Электронный ресурс]: Википедия. — URL: https://ru.wikipedia.org/wiki/Муравьиный_алгоритм (Дата обращения: 18.11.2024).
8. Алгоритм пчелиной колонии. [Электронный ресурс]: Википедия. —

URL: https://ru.wikipedia.org/wiki/Алгоритм_пчелиной_колонии (Дата обращения: 25.11.2024).

9. Бахтигозин, Т. Э. Анализ алгоритма пчелиной колонии и его применение в криптографии // Молодой исследователь Дона / Т. Э. Бахтигозин, М. Д. Пивоваров, О. А. Сафарьян. 2022. URL: <https://cyberleninka.ru/article/n/analiz-algoritma-pchelinoy-kolonii-i-ego-primeneniye-v-kriptografii> (Дата обращения: 26.11.2024).
10. Firefly algorithm. [Электронный ресурс]: Википедия. – URL: https://en.wikipedia.org/wiki/Firefly_algorithm (Дата обращения: 30.11.2024).
11. Карпенко, А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновлённые природой : учебное пособие / А. П. Карпенко. – 3-е изд., испр. – Москва : Издательство МГТУ им. Н. Э. Баумана, 2021. – 446 с.

ПРИЛОЖЕНИЯ

Приложение А — Код реализации онтологии.

Приложение Б.1 — Код реализации метода имитации отжига для задачи коммивояжёра.

Приложение Б.2 — Код реализации метода имитации отжига Коши для задачи поиска глобального минимума функции.

Приложение В.1 — Код реализации глобального роевого алгоритма.

Приложение В.2 — Код реализации локального роевого алгоритма.

Приложение Г — Код реализации простого муравьиного алгоритма.

Приложение Д — Код реализации алгоритма пчелиной колонии.

Приложение Е — Код реализации алгоритма роя светлячков.

Приложение А

Код реализации онтологии

Листинг А – Реализация онтологии

```
QUERY = {'Фильм': ('Актёры', 'Название'),
         'Многосерийный фильм': ('Актёры', 'Название', 'Количество сезонов',
                                   'Количество эпизодов'),
         'Полнометражный фильм': ('Актёры', 'Название', 'Длительность'),
         'Актёр': ('ФИО', ),
         'Режиссёр': ('ФИО', 'Руководит')}
```

```
FUNCTION = {'Актёры': ('contains', 'does not contain'),
            'Название': ('contains', 'does not contain', 'is',
                          'is not', 'begins with', 'ends with'),
            'Количество сезонов': ('is', 'is greater then', 'is less then'),
            'Количество эпизодов': ('is', 'is greater then', 'is less then'),
            'Длительность': ('is', 'is greater then', 'is less then'),
            'ФИО': ('contains', 'does not contain', 'is',
                    'is not', 'begins with', 'ends with'),
            'Руководит': ('contains', 'does not contain')}
```

```
class MovieIndustry:
    def __init__(self, *args, **kwargs):
        raise TypeError(f"Can't instantiate abstract class {
                        __class__.__name__}")

class Film(MovieIndustry):
    def __init__(self, name, actors):
        self._name = name
        self._actors = list(actors)

    def __str__(self):
        return self._name

    def __repr__(self):
        return f"{__class__.__name__}('{self._name}', {self._actors})"

    def __eq__(self, other):
        if type(other) is __class__:
            return other._name == self._name
        elif type(other) is str:
            return other == self._name
        return NotImplemented

    def __contains__(self, obj):
        return obj in self._name

    def has_actor(self, actor):
        return actor in self._actors

class SerialFilm(Film):
    def __init__(self, name, actors, num_seasons, num_episodes):
        super().__init__(name, actors)
        self._num_seasons = num_seasons
        self._num_episodes = num_episodes

    def __repr__(self):
        return f"{__class__.__name__}('{self._name}', {self._actors},
        {self._num_seasons}, {self._num_episodes})"
```



```
class FeatureFilm(Film):
    def __init__(self, name, actors, length):
        super().__init__(name, actors)
        self._length = length

    def __repr__(self):
        return f"{{__class__.__name__}}('{{self._name}}', {{self._actors}},
{{self._length}})"

class Actor(MovieIndustry):
    def __init__(self, name):
        self._name = name

    def __str__(self):
        return self._name

    def __repr__(self):
        return f"{{__class__.__name__}}('{{self._name}}'"

    def __eq__(self, other):
        if type(other) is __class__:
            return other._name == self._name
        elif type(other) is str:
            return other == self._name
        return NotImplemented

    def __contains__(self, obj):
        return obj in self._name

    def startswith(self, value):
        return self._name.startswith(value)

    def endswith(self, value):
        return self._name.endswith(value)

class FilmDirector(MovieIndustry):
    def __init__(self, name, films):
        self._name = name
        self._films = list(films)

    def __str__(self):
        return self._name

    def __repr__(self):
        return f"{{__class__.__name__}}('{{self._name}}', {{self._films}})"

    def __eq__(self, other):
        if type(other) is __class__:
            return other._name == self._name
        elif type(other) is str:
            return other == self._name
        return NotImplemented

    def __contains__(self, obj):
        return obj in self._name

    def startswith(self, value):
        return self._name.startswith(value)

    def endswith(self, value):
```

```
        return self._name.endswith(value)

    def has_movie(self, value):
        return value in self._films

class Queries:
    def __init__(self, cls, slot, function, value):
        self._cls = cls
        self._slot = slot
        self._value = value
        match function:
            case 'contains':
                self._func = __class__.contains
            case 'does not contain':
                self._func = __class__.does_not_contains
            case 'is':
                self._func = __class__.equal
            case 'is not':
                self._func = __class__.not_equal
            case 'begins with':
                self._func = __class__.startswith
            case 'ends with':
                self._func = __class__.endswith
            case 'is greater then':
                self._func = function
            case 'is less then':
                self._func = function
        self._result = None

    def find(self):
        match self._cls:
            case 'Фильм':
                if self._slot == 'Название':
                    found_films = list(filter(lambda object: self._func(self,
object), feature_films + serial_films))
                elif self._func == __class__.contains:
                    found_films = list(filter(lambda object:
object.has_actor(self._value), feature_films + serial_films))
                else:
                    found_films = list(filter(lambda object: not
object.has_actor(self._value), feature_films + serial_films))
                for film in found_films:
                    print(f'• {film} ({type(film).__name__})')
                    for film_director in film_directors:
                        if film_director.has_movie(film):
                            print(f'      • {film_director}
({type(film_director).__name__})')
            case 'Многосерийный фильм':
                if self._slot == 'Актёры' and self._func == __class__.contains:
                    found_films = list(filter(lambda object:
object.has_actor(self._value), serial_films))
                elif self._slot == 'Актёры' and self._func ==
__class__.does_not_contains:
                    found_films = list(filter(lambda object: not
object.has_actor(self._value), serial_films))
                elif self._slot == 'Количество сезонов' and self._func ==
__class__.equal:
                    found_films = list(filter(lambda object: object._num_seasons
== int(self._value), serial_films))
                elif self._slot == 'Количество сезонов' and self._func == 'is
greater then':
```

Продолжение Листинга А

```
        found_films = list(filter(lambda object: object._num_seasons
> int(self._value), serial_films))
        elif self._slot == 'Количество сезонов' and self._func == 'is
less then':
            found_films = list(filter(lambda object: object._num_seasons
< int(self._value), serial_films))
            elif self._slot == 'Количество эпизодов' and self._func ==
__class__.equal:
                found_films = list(filter(lambda object:
object._num_episodes == int(self._value), serial_films))
                elif self._slot == 'Количество эпизодов' and self._func == 'is
greater then':
                    found_films = list(filter(lambda object:
object._num_episodes > int(self._value), serial_films))
                    elif self._slot == 'Количество эпизодов' and self._func == 'is
less then':
                        found_films = list(filter(lambda object:
object._num_episodes < int(self._value), serial_films))
                        else:
                            found_films = list(filter(lambda object: self._func(self,
object), serial_films))
                            for film in found_films:
                                print(f'• {film} ({type(film).__name__})')
                                for film_director in film_directors:
                                    if film_director.has_movie(film):
                                        print(f'        • {film_director}
({type(film_director).__name__})')
                            case 'Полнометражный фильм':
                                if self._slot == 'Актёры' and self._func == __class__.contains:
                                    found_films = list(filter(lambda object:
object.has_actor(self._value), feature_films))
                                    elif self._slot == 'Актёры' and self._func ==
__class__.does_not_contains:
                                        found_films = list(filter(lambda object: not
object.has_actor(self._value), feature_films))
                                        elif self._slot == 'Длительность' and self._func ==
__class__.equal:
                                            found_films = list(filter(lambda object: object._length ==
int(self._value), feature_films))
                                            elif self._slot == 'Длительность' and self._func == 'is greater
then':
                                                found_films = list(filter(lambda object: object._length >
int(self._value), feature_films))
                                                elif self._slot == 'Длительность' and self._func == 'is less
then':
                                                    found_films = list(filter(lambda object: object._length <
int(self._value), feature_films))
                                                    else:
                                                        found_films = list(filter(lambda object: self._func(self,
object), feature_films))
                                                        for film in found_films:
                                                            print(f'• {film} ({type(film).__name__})')
                                                            for film_director in film_directors:
                                                                if film_director.has_movie(film):
                                                                    print(f'        • {film_director}
({type(film_director).__name__})')
                                                        case 'Актёр':
                                                            found_actors = list(
filter(lambda object: self._func(self, object), actors))
                                                            for actor in found_actors:
                                                                print(f'• {actor} ({type(actor).__name__})')
```

```

        for film in serial_films + feature_films:
            if film.has_actor(actor):
                print(f'      • {film} ({type(film).__name__})')
                for film_director in film_directors:
                    if film_director.has_movie(film):
                        print(f'\t• {film_director}
({type(film_director).__name__})')
                    case 'Режиссёр':
                        if self._slot == 'ФИО':
                            self._result = list(
                                filter(lambda object: self._func(self, object),
film_directors))
                        elif self._func == __class__.contains:
                            self._result = list(
                                filter(lambda object: object.has_movie(self._value),
film_directors))
                        else:
                            self._result = list(
                                filter(lambda object: not object.has_movie(self._value),
film_directors))
                            for item in self._result:
                                print(f'• {str(item)} ({type(item).__name__})')

    def contains(self, object):
        return self._value in object

    def does_not_contains(self, object):
        return self._value not in object

    def equal(self, object):
        return self._value == object

    def not_equal(self, object):
        return self._value != object

    def startswith(self, object):
        return object.startswith(self._value)

    def endswith(self, object):
        return object.endswith(self._value)

    def make_query():
        '''Функция для написания запроса'''
        print("\033[4mВыберите класс:\033[0m")
        for num, cls in enumerate(QUERY, 1):
            print(num, '-', '\033[93m' + cls + '\033[0m')
        cls_num = input()
        if cls_num not in map(str, range(1, len(QUERY) + 1)):
            raise TypeError('Некорректный номер класса')
        cls = list(QUERY.keys())[int(cls_num) - 1]

        print("\033[4mВыберите слот:\033[0m")
        for num, slot in enumerate(QUERY[cls], 1):
            print(num, '-', '\033[94m' + slot + '\033[0m')
        slot_num = input()
        if slot_num not in map(str, range(1, len(QUERY[cls]) + 1)):
            raise TypeError('Некорректный номер слота')
        slot = QUERY[cls][int(slot_num) - 1]

        print("\033[4mВыберите функцию запроса:\033[0m")
        for num, func in enumerate(FUNCTION[slot], 1):

```

Продолжение Листинга А

```
        print(num, '-', '\033[93m' + func + '\033[0m')
    func_num = input()
    if func_num not in map(str, range(1, len(FUNCTION[slot]) + 1)):
        raise TypeError('Некорректный номер функции')
    func = FUNCTION[slot][int(func_num) - 1]

    value = input("\033[4mВведите значение запроса:\033[0m ")

    query_obj = Queries(cls, slot, func, value)
    query_obj.find()

actors = [Actor('Bryan Cranston'), Actor('Anna Gunn'),
          Actor('Aaron Paul'), Actor('Dean Norris'),
          Actor('Betsy Brandt'), Actor('RJ Mitte'),
          Actor('Bob Odenkirk'), Actor('Giancarlo Esposito'),
          Actor('Jonathan Banks'), Actor('Steven Michael Quezada'),
          Actor('Cillian Murphy'), Actor('Emily Blunt'),
          Actor('Matt Damon'), Actor('Robert Downey Jr.'),
          Actor('Florence Pugh'), Actor('Josh Hartnett'),
          Actor('David Krumholtz'), Actor('Benny Safdie'),
          Actor('Alden Ehrenreich'), Actor('Kenneth Branagh'),
          Actor('Leonardo DiCaprio'), Actor('Jonah Hill'),
          Actor('Margot Robbie'), Actor('Kyle Chandler'),
          Actor('Rob Reiner'), Actor('P.J. Byrne'),
          Actor('Jon Bernthal'), Actor('Cristin Milioti'),
          Actor('Jean Dujardin'), Actor('Matthew McConaughey'),
          Actor('Ryan Gosling'), Actor('America Ferrera'),
          Actor('Ariana Greenblatt'), Actor('Kate McKinnon'),
          Actor('Issa Rae'), Actor('Will Ferrell'),
          Actor('Michael Cera'), Actor('Simu Liu'),
          Actor('Alexandra Shipp'), Actor('Joseph Gordon-Levitt'),
          Actor('Elliot Page'), Actor('Tom Hardy'),
          Actor('Ken Watanabe'), Actor('Dileep Rao'),
          Actor('Tom Berenger'), Actor('Marion Cotillard'),
          Actor('Pete Postlethwaite'), Actor('Paul Anderson'),
          Actor('Sophie Rundle'), Actor('Helen McCrory'),
          Actor('Ned Dennehy'), Actor('Finn Cole'),
          Actor("Natasha O'Keefe"), Actor('Ian Peck'),
          Actor('Harry Kirton'), Actor('Packy Lee'),
          Actor('Matthew Fox'), Actor('Evangeline Lilly'),
          Actor('Josh Holloway'), Actor("Terry O'Quinn"),
          Actor('Naveen Andrews'), Actor('Jorge Garcia'),
          Actor('Michael Emerson'), Actor('Emilie de Ravin'),
          Actor('Kim Yoon-jin'), Actor('Daniel Dae Kim'),
          Actor('Henry Ian Cusick'), Actor('Dominic Monaghan')]

serial_films = [SerialFilm('Breaking Bad', actors[:10], 5, 62),
                SerialFilm('Peaky Blinders', [actors[10]] + actors[47:56], 6,
36),
                SerialFilm('Lost', actors[56:68], 6, 121)]

feature_films = [FeatureFilm('Oppenheimer', actors[10:20], 180),
                 FeatureFilm('The Wolf of Wall Street', actors[20:30], 172),
                 FeatureFilm('Barbie', [actors[22]] + actors[30:39], 104),
                 FeatureFilm('Inception', [actors[20]] + [actors[10]] +
actors[39:47], 148)]

film_directors = [FilmDirector('Michelle MacLaren', [serial_films[0]]),
                  FilmDirector('Adam Bernstein', [serial_films[0]]),
                  FilmDirector('Vince Gilligan', [serial_films[0]]),
                  FilmDirector('Christopher Nolan', [
```

Окончание Листинга А

```
        feature_films[0], feature_films[3]]),
    FilmDirector('Martin Scorsese', [feature_films[1]]),
    FilmDirector('Greta Gerwig', [feature_films[2]]),
    FilmDirector('Anthony Byrne', [serial_films[1]]),
    FilmDirector('Colm McCarthy', [serial_films[1]]),
    FilmDirector('Tim Mielants', [serial_films[1]]),
    FilmDirector('David Caffrey', [serial_films[1]]),
    FilmDirector('Otto Bathurst', [serial_films[1]]),
    FilmDirector('Tom Harper', [serial_films[1]]),
    FilmDirector('Jack Bender', [serial_films[2]]),
    ]

make_query()
```

Приложение Б.1

Код реализации метода имитации отжига для задачи коммивояжёра

Листинг Б.1 – 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 Б.1

```
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'
```



```

        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

```

```

        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:

```

```

        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 Б.1

```
        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',
                  'МФИ',
                  '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 Б.1

```
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 Б.1

```
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.2

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

Листинг Б.2 - Реализация метода имитации отжига Коши для задачи поиска глобального минимума функции

```
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.2

```
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.2

```
        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)
```

Приложение В.1

Код реализации глобального роевого алгоритма

Листинг В.1 - Реализация глобального роевого алгоритма

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

def goldstein_price(x: float, y: float) -> float:
    '''Функция Голдштейна-Прайса для оптимизации.'''
    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

class Particle:
    def __init__(self, bounds: List[Tuple[float, float]], fitness_function:
        Callable[..., float]):
        '''
        Инициализирует частицу с случайными позицией и скоростью.
        Параметры:
            bounds (List[Tuple[float, float]]): Ограничения для координат каждой
            частицы.
            fitness_function (Callable[..., float]): Целевая функция для
            оптимизации.
        '''
        self.position = [random.uniform(*bound) for bound in bounds]
        self.velocity = [random.uniform(-1, 1) for _ in bounds]
        self.best_position = self.position[:]
        self.fitness_function = fitness_function
        self.best_value = self.fitness_function(*self.position)

    def __str__(self) -> str:
        return f"(Координаты: {[f'{pos:.4f}' for pos in self.position]}; " + \
            f"Скорость: {[f'{vel:.4f}' for vel in self.velocity]}; " + \
            f"Лучшие координаты: {[f'{b_pos:.4f}' for b_pos in
            self.best_position]}; " + \
            f"Лучшее значение функции: {self.best_value:.4f})."

    def update_velocity(self, global_best_position: List[float], c1: float =
        2.0, c2: float = 2.0) -> None:
        '''
        Обновляет скорость частицы на основе её лучшей позиции и глобальной
        лучшей позиции роя.
        Параметры:
            global_best_position (List[float]): Лучшая позиция в рое.
            c1 (float): Коэффициент когнитивного компонента.
            c2 (float): Коэффициент социального компонента.
        '''
        for i in range(len(self.position)):
            r1, r2 = random.random(), random.random()
            cognitive = c1 * r1 * (self.best_position[i] - self.position[i])
            social = c2 * r2 * (global_best_position[i] - self.position[i])
            self.velocity[i] += cognitive + social

    def update_position(self, bounds: List[Tuple[float, float]]) -> None:
        '''
        Обновляет позицию частицы с учётом ограничений и обновляет её лучшую
        позицию.
        Параметры:
```

Окончание Листинга В.1

```
        bounds (List[Tuple[float, float]]): Ограничения для координат.
    """
    for i in range(len(self.position)):
        self.position[i] += self.velocity[i]
        self.position[i] = max(
            min(self.position[i], bounds[i][1]), bounds[i][0])
    current_value = self.fitness_function(*self.position)
    if current_value < self.best_value:
        self.best_position = self.position[:]
        self.best_value = current_value

class Swarm:
    def __init__(self, fitness_function: Callable[..., float], bounds:
List[Tuple[float, float]],
        num_particles: int, max_iterations: int):
    """
    Инициализирует рой частиц для оптимизации.
    Параметры:
        fitness_function (Callable[..., float]): Целевая функция для
оптимизации.
        bounds (List[Tuple[float, float]]): Ограничения для координат.
        num_particles (int): Количество частиц в рое.
        max_iterations (int): Максимальное количество итераций для
оптимизации.
    """
    self.particles = [Particle(bounds, fitness_function)
        for _ in range(num_particles)]
    self.global_best_position = min(
        self.particles, key=lambda p: p.best_value).best_position[:]
    self.global_best_value = fitness_function(*self.global_best_position)
    self.max_iterations = max_iterations

    def optimize(self) -> Tuple[List[float], float]:
    """
    Выполняет оптимизацию, обновляя позиции и скорости частиц.
    Возвращает:
        Tuple[List[float], float]: Лучшая позиция и значение целевой
функции.
    """
    for k in range(self.max_iterations):
        print(f"Итерация: {k + 1}/{self.max_iterations}")
        for particle in self.particles:
            particle.update_position(bounds)
            if particle.best_value < self.global_best_value:
                self.global_best_position = particle.best_position[:]
                self.global_best_value = particle.best_value
        for particle in self.particles:
            particle.update_velocity(self.global_best_position)
        print(f"Лучшая позиция: {self.global_best_position}")
        print(f"Лучшее значение функции: {self.global_best_value:.12f}")
        print('-' * 40)
    return self.global_best_position, self.global_best_value

bounds = [(-2, 2), (-2, 2)]
num_particles = 500 # Количество частиц
max_iterations = 500 # Максимальное количество итераций
swarm = Swarm(goldstein_price, bounds, num_particles, max_iterations)
best_position, best_value = swarm.optimize()
print("Лучшая позиция:", best_position)
print(f"Лучшее значение функции: {best_value:.12f}")
```

Приложение В.2

Код реализации локального роевого алгоритма

Листинг В.2 - Реализация локального роевого алгоритма

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

def goldstein_price(x: float, y: float) -> float:
    '''Функция Голдштейна-Прайса для оптимизации.'''
    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

class Particle:
    def __init__(self, bounds: List[Tuple[float, float]], fitness_function:
Callable[..., float]):
        '''
        Инициализирует частицу с случайными позицией и скоростью.
        Параметры:
            bounds (List[Tuple[float, float]]): Ограничения для координат каждой
частицы.
            fitness_function (Callable[..., float]): Целевая функция для
оптимизации.
        '''
        self.position = [random.uniform(*bound) for bound in bounds]
        self.velocity = [random.uniform(-1, 1) for _ in bounds]
        self.best_position = self.position[:]
        self.fitness_function = fitness_function
        self.best_value = self.fitness_function(*self.position)

    def __str__(self) -> str:
        '''Возвращает строковое представление текущего состояния частицы.'''
        return f"(Координаты: {[f'{pos:.4f}' for pos in self.position]}; " + \
            f"Скорость: {[f'{vel:.4f}' for vel in self.velocity]}; " + \
            f"Лучшие координаты: {[f'{b_pos:.4f}' for b_pos in
self.best_position]}; " + \
            f"Лучшее значение функции: {self.best_value:.4f})."

    def update_velocity(self, local_best_position: List[float], c1: float = 2.0,
c2: float = 2.0) -> None:
        '''
        Обновляет скорость частицы на основе её лучшей позиции и локальной
лучшей позиции.
        Параметры:
            local_best_position (List[float]): Локальная лучшая позиция.
            c1 (float): Коэффициент когнитивного компонента.
            c2 (float): Коэффициент социального компонента.
        '''
        for i in range(len(self.position)):
            r1, r2 = random.random(), random.random()
            cognitive = c1 * r1 * (self.best_position[i] - self.position[i])
            social = c2 * r2 * (local_best_position[i] - self.position[i])
            self.velocity[i] += cognitive + social

    def update_position(self, bounds: List[Tuple[float, float]]) -> None:
        '''
        Обновляет позицию частицы с учётом ограничений и обновляет её лучшую
позицию.
```

Продолжение Листинга В.2

```
        Параметры:
        bounds (List[Tuple[float, float]]): Ограничения для координат.
    '''
    for i in range(len(self.position)):
        self.position[i] += self.velocity[i]
        self.position[i] = max(min(self.position[i], bounds[i][1]),
bounds[i][0])
        current_value = self.fitness_function(*self.position)
        if current_value < self.best_value:
            self.best_position = self.position[:]
            self.best_value = current_value

class Swarm:
    def __init__(self, fitness_function: Callable[..., float], bounds:
List[Tuple[float, float]],
        num_particles: int, max_iterations: int):
        '''
        Инициализирует рой частиц для оптимизации.
        Параметры:
        fitness_function (Callable[..., float]): Целевая функция для
оптимизации.
        bounds (List[Tuple[float, float]]): Ограничения для координат.
        num_particles (int): Количество частиц в рое.
        max_iterations (int): Максимальное количество итераций для
оптимизации.
        '''
        self.particles = [Particle(bounds, fitness_function) for _ in
range(num_particles)]
        self.fitness_function = fitness_function
        self.max_iterations = max_iterations

    def get_local_best(self, index: int) -> List[float]:
        '''
        Находит лучшую позицию среди соседей данной частицы.
        Параметры:
        index (int): Индекс текущей частицы.
        Возвращает:
        List[float]: Лучшая позиция среди соседей.
        '''
        neighbors_indices = [(index - 1) % len(self.particles), index, (index +
1) % len(self.particles)]
        neighbors = [self.particles[i] for i in neighbors_indices]
        best_neighbor = min(neighbors, key=lambda p: p.best_value)
        return best_neighbor.best_position

    def optimize(self) -> Tuple[List[float], float]:
        '''
        Выполняет оптимизацию, обновляя позиции и скорости частиц.
        Возвращает:
        Tuple[List[float], float]: Лучшая позиция и значение целевой
функции.
        '''
        for _ in range(self.max_iterations):
            for i, particle in enumerate(self.particles):
                local_best_position = self.get_local_best(i)
                particle.update_velocity(local_best_position)
            for particle in self.particles:
                particle.update_position(bounds)
            best_particle = min(self.particles, key=lambda p: p.best_value)
            return best_particle.best_position, best_particle.best_value
```

Окончание Листинга В.2

```
bounds = [(-2, 2), (-2, 2)]
num_particles = 500 # Количество частиц
max_iterations = 500 # Максимальное количество итераций
swarm = Swarm(goldstein_price, bounds, num_particles, max_iterations)
best_position, best_value = swarm.optimize()
print("Лучшая позиция:", best_position)
print(f"Лучшее значение функции: {best_value:.12f}")
```

Приложение Г

Код реализации простого муравьиного алгоритма

Листинг Г – Реализация простого муравьиного алгоритма

```
import random
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(frozen=True)
class Vertex:
    '''Класс для представления узла графа, который включает название,
    сокращенное имя и адрес.'''
    name: str
    short_name: str = field(compare=False)
    address: str

    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 = [[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
    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 весов 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'
                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
    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 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:
            writer.writerow(row)

class SimpleAntColonyOptimization:
    def __init__(self, graph: Graph, k_max: int, num_ants: int, alpha: float, p:
float):

```

```

'''
Инициализирует алгоритм муравьиной колонии.
Параметры:
    graph (Graph): Граф, для которого выполняется оптимизация.
    k_max (int): Максимальное количество итераций.
    num_ants (int): Количество муравьев.
    alpha (float): Влияние феромонов.
    p (float): Коэффициент испарения феромонов.
'''
self.graph = graph
self.k_max = k_max
self.num_ants = num_ants
self.alpha = alpha
self.p = p
self.pheromones = [[0.1 if i != j else 0 for j in graph.vertices]
                    for i in graph.vertices]
self.best_path = None
self.best_cost = float('inf')

def optimize(self, output=True):
    '''
    Выполняет оптимизацию для поиска лучшего пути.
    Возвращает:
        Tuple[List[int], float]: Лучший путь и его стоимость.
    '''
    for k in range(self.k_max):
        if output:
            print('\033[92m' +
                  f"Итерация: {k + 1}/{self.k_max}" + '\033[0m')
        paths = []
        path_costs = []
        for ant in range(self.num_ants):
            path = self.construct_path()
            cost, calculation = self.graph.calculate_cost(
                [self.graph.vertices[i] for i in path])
            if output:
                print('\033[96m' +
                      f"Муравей: {ant + 1}/{self.num_ants}" + '\033[0m')
                print(f"Построенный путь: {
                    ' -> '.join(self.graph.vertices[node].short_name for
node in path)})")
                print(f"Стоимость построенного пути: {calculation}")
            paths.append(path)
            path_costs.append(cost)
            if cost < self.best_cost:
                self.best_cost = cost
                self.best_path = path
        self.evaporate_pheromones()
        self.deposit_pheromones(paths, path_costs)
        if output:
            print(f"Лучший путь: {
                ' -> '.join(self.graph.vertices[node].short_name for node in
self.best_path)})")
            print(f"Стоимость лучшего пути: {self.best_cost}")
        return self.best_path, self.best_cost

def construct_path(self) -> List[int]:
    '''
    Строит маршрут для одного муравья.
    Возвращает:
        List[int]: Последовательность индексов вершин.
    '''

```

```

'''
visited = set()
current_node = 0
path = [current_node]
visited.add(current_node)
while len(visited) < len(self.graph.vertices):
    next_node = self.choose_next_node(current_node, visited)
    path.append(next_node)
    visited.add(next_node)
    current_node = next_node
path.append(0)
return path

def choose_next_node(self, current_node: int, visited: set) -> int:
'''
Выбирает следующую вершину на основе вероятностей.
Параметры:
    current_node (int): Текущая вершина.
    visited (set): Набор уже посещенных вершин.
Возвращает:
    int: Индекс следующей вершины.
'''
probabilities = []
neighbors = range(len(self.graph.vertices))
for j in neighbors:
    if j not in visited:
        pheromone = self.pheromones[current_node][j]
        probabilities.append((j, pheromone ** self.alpha))
total = sum(prob[1] for prob in probabilities)
probabilities = [(node, prob / total) for node, prob in probabilities]
random_choice = random.uniform(0, 1)
cumulative = 0
for node, prob in probabilities:
    cumulative += prob
    if random_choice <= cumulative:
        return node
return neighbors[-1]

def evaporate_pheromones(self):
'''Испаряет феромоны на всех ребрах графа.'''
for i in range(len(self.pheromones)):
    for j in range(len(self.pheromones[i])):
        self.pheromones[i][j] *= (1 - self.p)

def deposit_pheromones(self, paths: List[List[int]], costs: List[float]):
'''
Обновляет феромоны на ребрах на основе пройденных путей.
Параметры:
    paths (List[List[int]]): Список всех пройденных путей.
    costs (List[float]): Список стоимости каждого пути.
'''
for i in range(len(self.best_path) - 1):
    self.pheromones[self.best_path[i]
                    ][self.best_path[i + 1]] += 1 / self.best_cost

for path, cost in zip(paths, costs):
    for i in range(len(path) - 1):
        self.pheromones[path[i]][path[i + 1]] += 1 / cost

vertex_0 = Vertex('Звезды Арбата',
                  'Отель',

```

Продолжение Листинга Г

```
'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')
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')
```

Окончание Листинга Г

```
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')
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()

saco = SimpleAntColonyOptimization(
    graph, k_max=1000, num_ants=24, alpha=1.190657414100356,
    p=0.4160457456608921)
best_path, best_cost = sacco.optimize()
print('-' * 40)
print("Лучший путь:",
      " -> ".join(graph.vertices[node].short_name for node in best_path))
print("Стоимость пути:", best_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()

# sacco = SimpleAntColonyOptimization(
#     test_graph, k_max=100, num_ants=7, alpha=1.1, p=0.25)
# best_path, best_cost = sacco.optimize()
# print('-' * 40)
# print("Лучший путь:",
#       " -> ".join(test_graph.vertices[node].short_name for node in best_path))
# print("Стоимость пути:", best_cost)
```

Приложение Д

Код реализации алгоритма пчелиной колонии

Листинг Д – Реализация алгоритма пчелиной колонии

```
import random
import math
import itertools
from typing import List, Tuple
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

class Point:
    def __init__(self, *args):
        self.coordinates = args

    def euclidean_distance(self, other_point):
        '''Вычисляет евклидово расстояние между текущей точкой и другой
        точкой.'''
        if not isinstance(other_point, __class__):
            raise ValueError('Евклидово расстояние может быть рассчитано только
            между экземплярами Point.')
        return math.sqrt(sum((x - y) ** 2 for x, y in zip(self, other_point)))

    def __repr__(self):
        return f"Point{self.coordinates}"

    def __str__(self):
        return str(self.coordinates)

    def __getitem__(self, index):
        return self.coordinates[index]

class BeeColony:
    def __init__(self, fitness_function, bounds: List[Tuple[float, float]],
        scouts: int, k: int,
        distance_threshold: float, l: float, max_iterations: int,
        maximize: bool = False):
        '''
        Инициализирует алгоритм пчелиной колонии для оптимизации.
        Параметры:
        fitness_function (Callable[..., float]): Целевая функция для
        оптимизации.
        bounds (List[Tuple[float, float]]): Ограничения для каждой
        координаты в виде [(min1, max1), ..., (minD, maxD)].
        scouts (int): Количество пчел-разведчиков.
        k (int): Количество итераций без улучшения для остановки алгоритма.
        distance_threshold (float): Максимальное евклидово расстояние для
        объединения точек.
        l (float): Размер области локального поиска.
        max_iterations (int): Максимальное количество итераций алгоритма.
        maximize (bool): Определяет, ищется максимум (True) или минимум
        функции (False).
        '''
        self.fitness_function = fitness_function
        self.bounds = bounds
        self.scouts = scouts
        self.k = k
        self.distance_threshold = distance_threshold
        self.l = l
        self.max_iterations = max_iterations
```

```

self.maximize = maximize
self.history = list()

def compare(self, a, b):
    '''Сравнение значений функции с учетом типа оптимизации.'''
    return a > b if self.maximize else a < b

def optimize(self) -> Tuple[Point, float]:
    '''
    Выполняет оптимизацию с использованием алгоритма пчелиной колонии.
    Возвращает:
        Tuple[Point, float]: Лучшая найденная точка и значение целевой
        функции в ней.
    '''
    scouts = [
        Point(*[random.uniform(bounds[0], bounds[1]) for bounds in
self.bounds])
        for _ in range(self.scouts)
    ]
    best_global_value = float("-inf") if self.maximize else float("inf")
    best_global_point = None
    num_iteration = 0
    stagnation_count = 0

    while num_iteration < self.max_iterations:
        print(f"Итерация №{num_iteration}")
        num_iteration += 1
        iteration_data = []

        values = [self.fitness_function(*point) for point in scouts]
        best_local_index = max(range(len(values)), key=lambda i: values[i]
\
        if self.maximize else min(range(len(values)), key=lambda i:
values[i]))
        best_local_point = scouts[best_local_index]
        best_local_value = values[best_local_index]
        if self.compare(best_local_value, best_global_value):
            best_global_value = best_local_value
            best_global_point = best_local_point
            stagnation_count = 0
        else:
            stagnation_count += 1

        combined_regions = []
        used_indices = set()
        for i, point_i in enumerate(scouts):
            if i in used_indices:
                continue
            region = [point_i]
            for j, point_j in enumerate(scouts):
                if j != i and j not in used_indices and \
                    point_i.euclidean_distance(point_j) <=
self.distance_threshold:
                region.append(point_j)
                used_indices.add(j)
            used_indices.add(i)
            combined_regions.append(region)
        print(f"Количество подобластей: {len(combined_regions)}")

        new_scouts = []
        for region in combined_regions:

```



```

        center = max(region, key=lambda p: self.fitness_function(*p)) \
            if self.maximize else min(region, key=lambda p:
self.fitness_function(*p))
        search_area = [
            (max(self.bounds[i][0], center[i] - self.l),
min(self.bounds[i][1], center[i] + self.l))
            for i in range(len(self.bounds))
        ]
        local_scouts = [center] + [
            Point(*[random.uniform(area[0], area[1]) for area in
search_area])
            for _ in range(self.scouts - 1)
        ]
        iteration_data.append(local_scouts)
        local_values = [self.fitness_function(*point) for point in
local_scouts]
        best_local_index = max(range(len(local_values)), key=lambda i:
local_values[i]) \
            if self.maximize else min(range(len(local_values)),
key=lambda i: local_values[i])
        new_scouts.append(local_scouts[best_local_index])

        self.history.append(iteration_data)
        scouts = new_scouts
        if stagnation_count >= self.k:
            break
        print(f"Лучшая позиция: {best_global_point}")
        print(f"Лучшее значение функции: {best_global_value:.6f}")
        return best_global_point, best_global_value

def visualize(self):
    '''Визуализирует работу пчел.'''
    fig, ax = plt.subplots()
    x_min, x_max = self.bounds[0]
    y_min, y_max = self.bounds[1]
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    colors = itertools.cycle(['b', 'g', 'r', 'c', 'm', 'y', 'k'])
    def update(frame):
        ax.clear()
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.set_title(f"Итерация {frame + 1}")
        iteration_data = self.history[frame]
        for region_points in iteration_data:
            x_coords = [p[0] for p in region_points]
            y_coords = [p[1] for p in region_points]
            color = next(colors)
            ax.scatter(x_coords, y_coords, c=color, s=10, label="Region
Points")
        ax.legend(loc='upper right')
    anim = FuncAnimation(fig, update, frames=len(self.history), blit=False,
interval=500, repeat=False)
    plt.show()

def goldstein_price(x: float, y: float) -> float:
    '''Функция Голдштейна-Прайса для оптимизации.'''
    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))

```

Окончание Листинга Д

```
        return term1 * term2

def sphere(x, y):
    '''Функция сферы для оптимизации.'''
    return x ** 2 + y ** 2

colony = BeeColony(
    fitness_function=goldstein_price,
    bounds=[(-20, 20), (-20, 20)],
    scouts=100,
    k=100,
    distance_threshold=1.5,
    l=0.5,
    max_iterations=1000,
    maximize=False
)

best_position, best_value = colony.optimize()
colony.visualize()
print("Лучшая позиция:", best_position)
print(f"Лучшее значение функции: {best_value:.6f}")
```

Приложение Е

Код реализации алгоритма роя светлячков

Листинг Е – Реализация алгоритма роя светлячков

```
import random
import math
from typing import List, Tuple, Callable
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def goldstein_price(x: float, y: float) -> float:
    """
    Функция Голдштейна-Прайса для оптимизации.
    Параметры:
        x (float): Координата по оси x.
        y (float): Координата по оси y.
    Возвращает:
        float: Значение функции Голдштейна-Прайса.
    """
    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

class Firefly:
    def __init__(self, position: List[float], luciferin: float = 0.0, radius:
float = 1.0):
        """
        Инициализирует светлячка.
        Параметры:
            position (List[float]): Начальная позиция светлячка.
            luciferin (float): Уровень люциферина.
            radius (float): Радиус окрестности светлячка.
        """
        self.position = position
        self.luciferin = luciferin
        self.radius = radius

    def update_luciferin(self, function_value: float, rho: float, gamma: float):
        """
        Обновляет уровень люциферина светлячка.
        Параметры:
            function_value (float): Значение целевой функции в текущей позиции.
            rho (float): Коэффициент уменьшения люциферина.
            gamma (float): Коэффициент привлекательности светлячка.
        """
        self.luciferin = (1 - rho) * self.luciferin + \
            gamma * (1 / function_value)

    def move_towards(self, other: 'Firefly', delta: float, bounds:
List[Tuple[float, float]]):
        """
        Перемещает светлячка в направлении другого более яркого светлячка.
        Параметры:
            other (Firefly): Другой светлячок, к которому перемещается текущий.
            delta (float): Коэффициент изменения позиции.
            bounds (List[Tuple[float, float]]): Границы пространства поиска.
        """
        direction = [other.position[i] - self.position[i]
```

```

        for i in range(len(self.position))
    distance = math.sqrt(sum(d ** 2 for d in direction))
    if distance > 0:
        normalized_direction = [d / distance for d in direction]
        self.position = [
            min(max(
                self.position[i] + delta * normalized_direction[i],
                bounds[i][0]), bounds[i][1])
            for i in range(len(self.position))
        ]

class FireflySwarm:
    def __init__(self, fitness_function: Callable[..., float], bounds:
List[Tuple[float, float]],
        num_fireflies: int, max_iterations: int, beta: float, rho:
float,
        delta: float, gamma: float, initial_radius: float):
        """
        Инициализирует алгоритм роя светлячков.
        Параметры:
            fitness_function (Callable[..., float]): Целевая функция для
оптимизации.
            bounds (List[Tuple[float, float]]): Границы пространства поиска
[(xmin, xmax), (ymin, ymax)].
            num_fireflies (int): Количество светлячков.
            max_iterations (int): Максимальное количество итераций.
            beta (float): Коэффициент изменения радиуса окрестности.
            rho (float): Коэффициент уменьшения уровня люциферина.
            delta (float): Коэффициент изменения позиции.
            gamma (float): Коэффициент увеличения люциферина.
            initial_radius (float): Начальный радиус окрестности.
        """
        self.fitness_function = fitness_function
        self.bounds = bounds
        self.num_fireflies = num_fireflies
        self.max_iterations = max_iterations
        self.beta = beta
        self.rho = rho
        self.delta = delta
        self.gamma = gamma
        self.initial_radius = initial_radius
        self.fireflies = []
        self.min_radius = 0.1
        self.max_radius = max(
            bounds[0][1] - bounds[0][0], bounds[1][1] - bounds[1][0])
        self.best_values = []
        self.positions_history = []

    def initialize_fireflies(self):
        """Инициализирует популяцию светлячков в случайных позициях."""
        self.fireflies = [
            Firefly(
                position=[random.uniform(bounds[0], bounds[1])
                    for bounds in self.bounds],
                radius=self.initial_radius
            )
            for _ in range(self.num_fireflies)
        ]

    def calculate_neighbors(self, firefly: Firefly) -> List[Firefly]:
        """

```

Продолжение Листинга E

```
        Вычисляет множество соседей светлячка.
        Параметры:
            firefly (Firefly): Текущий светлячок.
        Возвращает:
            List[Firefly]: Список соседей светлячка.
        """
        return [
            other for other in self.fireflies
            if other is not firefly
            and math.dist(firefly.position, other.position) < firefly.radius
            and firefly.luciferin < other.luciferin
        ]

    def calculate_probabilities(self, firefly: Firefly, neighbors:
List[Firefly]) -> List[float]:
        """
        Вычисляет вероятность перемещения к соседям.
        Параметры:
            firefly (Firefly): Текущий светлячок.
            neighbors (List[Firefly]): Список соседей.
        Возвращает:
            List[float]: Список вероятностей перемещения к каждому соседу.
        """
        total_difference = sum(
            other.luciferin - firefly.luciferin for other in neighbors)
        if total_difference == 0:
            return [1 / len(neighbors)] * len(neighbors)
        probabilities = [(other.luciferin - firefly.luciferin) /
            total_difference for other in neighbors]
        return probabilities

    def select_neighbor(self, neighbors: List[Firefly], probabilities:
List[float]) -> Firefly:
        """
        Выбирает соседа на основе вероятностей методом рулетки.
        Параметры:
            neighbors (List[Firefly]): Список соседей.
            probabilities (List[float]): Список вероятностей.
        Возвращает:
            Firefly: Выбранный сосед.
        """
        cumulative_probabilities = [
            sum(probabilities[:i + 1]) for i in range(len(probabilities))]
        rand = random.random()
        for i, prob in enumerate(cumulative_probabilities):
            if rand <= prob:
                return neighbors[i]

    def adjust_radius(self, firefly: Firefly, desired_neighbors: int):
        """
        Корректирует радиус окрестности светлячка.
        Параметры:
            firefly (Firefly): Текущий светлячок.
            desired_neighbors (int): Целевое количество соседей.
        """
        current_neighbors = len(self.calculate_neighbors(firefly))
        new_radius = firefly.radius + self.beta * \
            (desired_neighbors - current_neighbors)
        firefly.radius = min(self.max_radius, max(self.min_radius, new_radius))

    def optimize(self) -> Tuple[List[float], float]:
```

```

'''
Запускает процесс оптимизации.
Возвращает:
    Tuple[List[float], float]: Лучшая позиция и значение целевой
функции.
'''
self.initialize_fireflies()
best_position = None
best_value = float('inf')
desired_neighbors = 5

for iteration in range(self.max_iterations):
    for firefly in self.fireflies:
        function_value = self.fitness_function(*firefly.position)
        firefly.update_luciferin(function_value, self.rho, self.gamma)

    iteration_positions = []
    for firefly in self.fireflies:
        neighbors = self.calculate_neighbors(firefly)
        if neighbors:
            probabilities = self.calculate_probabilities(
                firefly, neighbors)
            selected_neighbor = self.select_neighbor(
                neighbors, probabilities)
            firefly.move_towards(
                selected_neighbor, self.delta, self.bounds)
        iteration_positions.append(firefly.position)
    self.positions_history.append(iteration_positions)

    for firefly in self.fireflies:
        self.adjust_radius(firefly, desired_neighbors)

    for firefly in self.fireflies:
        value = self.fitness_function(*firefly.position)
        if value < best_value:
            best_value = value
            best_position = firefly.position

    self.best_values.append(best_value)
    print(f"Итерация {iteration +
        1}: Лучший результат = {best_value:.6f}")

    return best_position, best_value

def plot_history(self):
    '''Отображает график сходимости.'''
    plt.plot(self.best_values)
    plt.title("Сходимость алгоритма роя светлячков")
    plt.xlabel("Итерации")
    plt.ylabel("Лучшее значение")
    plt.show()

def visualize(self):
    '''Анимация перемещения светлячков с учётом яркости.'''
    fig, ax = plt.subplots()
    x_min, x_max = self.bounds[0]
    y_min, y_max = self.bounds[1]
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)

    def update(frame):

```

```

        ax.clear()
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.set_title(f"Итерация {frame + 1}")
        positions = self.positions_history[frame]
        luciferin_values = [firefly.luciferin for firefly in self.fireflies]
        max_luciferin = max(luciferin_values)
        min_luciferin = min(luciferin_values)
        normalized_brightness = [
            (1 - min_luciferin) / (max_luciferin - min_luciferin + 1e-9)
            for l in luciferin_values
        ]
        x_coords, y_coords = zip(*positions)
        ax.scatter(
            x_coords, y_coords,
            c="green",
            s=10,
            alpha=normalized_brightness
        )

        anim = FuncAnimation(
            fig, update, frames=len(self.positions_history), blit=False,
            interval=500, repeat=False
        )
        plt.show()

if __name__ == "__main__":
    swarm = FireflySwarm(
        fitness_function=goldstein_price,
        bounds=[(-2, 2), (-2, 2)],
        num_fireflies=100,
        max_iterations=200,
        beta=0.6,
        rho=0.4,
        delta=0.25,
        gamma=1.0,
        initial_radius=0.5
    )
    best_position, best_value = swarm.optimize()
    print(f"Оптимальное решение: {best_position}, Значение: {best_value:.6f}")
    swarm.plot_history()
    swarm.visualize()

```