



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 «Онлайн курсы»

Студент группы ИКБО-15-22 Оганнисян Г.А.

(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 «____» _____ 2024г.

Допущен к защите «____» _____ 2024г.

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 _____

Подпись

Платонова О.В.

ФИО

«15» сентябрь 2024г.

ЗАДАНИЕ

на выполнение курсовой работы по дисциплине

«Системный анализ данных в системах поддержки принятия решений»

Студент Оганнисян Г.А. Группа ИКБО-15-22

Тема: «Онлайн курсы»

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

(Оганнисян Г.А.)

Ф.И.О. исполнителя

Москва 2024г.

ОТЗЫВ
на курсовую работу
по дисциплине «Системный анализ данных в системах поддержки принятия
решений»

Студент Оганнисян Г.А.
(ФИО студента)

ИКБО-15-22
(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:

Рекомендуемая оценка:

Подпись руководителя

Сорокин А.Б.

ФИО руководителя

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ОНТОЛОГИЯ	8
1.1 Понятие онтологии, модель, виды онтологий	8
1.2 Операции с онтологиями.....	9
1.3 Предметная область онтологии	11
1.4 Реализация онтологии в программе «Protégé»	12
1.5 Реализация онтологии на языке программирования GoLang	17
2 АЛГОРИТМ ОТЖИГ	19
2.1 Описание алгоритма отжига задача коммивояжера	20
2.2 Ручной расчет алгоритма отжига коммивояжера.....	20
2.3 Программная реализация алгоритма отжига коммивояжера.....	23
2.4 Описание алгоритма отжига Коши (Быстрый отжиг)	24
2.5 Принцип работы алгоритма отжига быстрого Коши.....	25
2.6 Ручной расчет алгоритма отжига Коши	26
2.7 Программная реализация алгоритма отжига Коши	27
3 ОСНОВНОЙ РОЕВОЙ АЛГОРИТМ	28
3.1 Постановка цели и задачи	29
3.2 Описание алгоритма	29
3.3 Ручной расчет роевого алгоритма	31
3.4 Программная реализация роевого алгоритма	33
4 МУРАВЬИНЫЙ АЛГОРИТМ	35
4.1 Постановка задачи	35
4.2 Описание муравьиного алгоритма	36
4.3 Ручной расчет муравьиного алгоритма	38
4.4 Программная реализация муравьиного алгоритма.....	43
5 ПЧЕЛИНЫЙ АЛГОРИТМ	45
5.1 Описание пчелиного алгоритма	46
5.2 Постановка задачи и цели	48

5.3 Ручной расчет пчелиного алгоритма.....	49
5.4 Программная реализация пчелиного алгоритма.....	51
6 ОТЧЕТ ПО ХАКАТОНУ: БОТ-ПОМОШНИК ДЛЯ УСКОРЕНИЕ АДАПТАЦИИ ПОЛЬЗОВАТЕЛЕЙ К ПО.....	52
6.1 Постановка цели и задачи	52
6.2 Архитектура системы	52
6.3 Описание функциональности	53
6.4 Результат работы.....	54
6.5 Заключение	54
ЗАКЛЮЧЕНИЕ	56
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ	57
ПРИЛОЖЕНИЯ.....	59

ВВЕДЕНИЕ

Системный анализ данных в системах поддержки принятия решений представляет собой методологию, направленную на изучение, обработку и использование данных для обоснованного выбора оптимальных решений. В условиях современного информационного общества, когда объем доступной информации постоянно увеличивается, системный подход позволяет организовать данные и применять эффективные алгоритмы для их анализа.

Целью данной курсовой работы является исследование различных алгоритмов, таких как алгоритмы отжига, основной роевой, муравьиный и пчелиный, которые применяются в задачах оптимизации и поиска оптимальных решений. Эти алгоритмы помогают находить лучшие решения в ситуациях, где необходимо учитывать множество факторов и ограничений. Каждый из представленных методов имеет свои особенности и может быть адаптирован к специфике задачи, что делает его полезным инструментом для систем поддержки принятия решений.

Для повышения эффективности анализа данных в рамках данной работы особое внимание уделено алгоритму отжига, который позволяет находить приближенные решения в задачах сложной многомерной оптимизации. Также подробно рассмотрены принципы работы муравьиного алгоритма, который использует идеи коллективного поведения для поиска оптимальных путей, и пчелиного алгоритма, имитирующего поведение пчел для решения задач многокритериальной оптимизации.

Важным элементом системного анализа является четкая постановка целей и задач, что позволяет определить критерии отбора решений и выбрать наиболее подходящие методы. В рамках данной работы рассматриваются подходы к разработке и реализации алгоритмов, которые способствуют повышению эффективности процессов принятия решений в условиях многокритериальности и неопределенности.

Таким образом, результаты данной работы могут быть полезны для разработки систем поддержки принятия решений в различных областях, где требуется оперативный и обоснованный выбор оптимального решения в условиях сложной информационной среды.

1 ОНТОЛОГИЯ

1.1 Понятие онтологии, модель, виды онтологий

Онтология – это формализованная модель представления знаний, концепций и их взаимосвязей в определенной предметной области. Она описывает понятия, их свойства и отношения между ними, чтобы обеспечить единое понимание для машины или человека. В информатике и искусственном интеллекте, онтологии используются для структурирования данных, улучшения поиска и фильтрации информации, а также для поддержки систем в принятии решений на основе семантического понимания данных.

Виды онтологий:

1. Онтологии представления помогают сформировать базовые идеи для представления знаний. Понятия и связи из других онтологий уточняют идеи этой онтологии и не зависят от конкретной предметной области. Эти онтологии могут поддерживать разные теории по одной и той же теме.

2. Общие онтологии охватывают базовые концепции, такие как «часть», «причина», «участие» и «представление». Они помогают понять основные отношения и идеи, которые применимы в разных контекстах.

3. Промежуточные онтологии, содержащие общие понятия и отношения, характерные для конкретной предметной области. В идеальном случае, они используются в качестве интерфейса между онтологиями предметных областей и общими онтологиями, но могут выступать как онтологии верхнего уровня для описания знаний конкретной.

4. Онтологии верхнего уровня, являющиеся конкретным назначением понятий общих и промежуточных онтологий. Это уникальный модуль, находящийся над онтологиями предметной области, или являющийся самостоятельной, независимой от предметной области теорией.

5. Онтологии предметных областей содержат понятия, термины, связанные с конкретной областью знаний.

6. Онтологии задач описывают конкретные задачи, основываясь на терминах из общих и предметных онтологий.

7. Онтологии-приложения, являющиеся специализацией онтологий предметных областей и онтологий задач, и опирающиеся на определения, характерные для конкретного приложения.

Формальная модель онтологии часто основана на логике, используя формальные языки для описания семантики, отношений и связей между концепциями.

1.2 Операции с онтологиями

1. Операции по редактированию:

Операции редактирования онтологий включают добавление, изменение и удаление элементов онтологии, таких как концепты, отношения, свойства и аксиомы. Это базовые операции, которые используются для изменения структуры или содержания онтологии:

- добавление: Включение новых классов, свойств, индивидов или аксиом в онтологию;
- изменение: Обновление существующих концептов, аксиом или отношений;
- удаление: Удаление концептов или аксиом из онтологии.

2. Алгебра онтологий:

Алгебра онтологий предполагает математические операции над онтологиями, позволяющие производить манипуляции над ними, как это делается с математическими объектами:

- объединение (Union): Создание новой онтологии на основе объединения нескольких существующих. Все классы, отношения и свойства объединяются в одну общую структуру;

- пересечение (Intersection): Операция, при которой создаётся новая онтология, включающая только те концепты и отношения, которые есть во всех исходных онтологиях;
- разность (Difference): Создание онтологии, содержащей элементы одной онтологии, которые отсутствуют в другой. Это может быть полезно для анализа различий между двумя моделями.

3. Операции по интеграции онтологий:

Интеграция онтологий необходима для объединения знаний из разных источников. Она включает такие операции, как сопоставление, объединение и согласование различных онтологий:

- сопоставление (Matching): Операция, в которой ищутся эквивалентные или схожие концепты и отношения в разных онтологиях. Сопоставление может выполняться на основе семантического анализа, анализа структуры или синтаксического анализа. Это часто используется при интеграции онтологий из разных источников данных;
- слияние (Merging): Это операция, при которой две онтологии объединяются в одну, с созданием новой структуры, которая содержит все уникальные и общие элементы. Важно, чтобы при слиянии не возникало противоречий между концептами и отношениями;
- разрешение конфликтов (Conflict Resolution): Это операция, применяемая при слиянии онтологий, если разные онтологии используют одни и те же концепты по-разному. Необходимо согласовать их определения и установить правила для разрешения таких конфликтов.

4. Операции по агрегированию и декомпозиции:

Операции агрегирования и декомпозиции позволяют управлять структурой онтологий на макро- и микроуровнях:

- агрегирование (Aggregation): Операция, при которой более мелкие концепты и отношения объединяются в более общие или абстрактные категории;

- **декомпозиция (Decomposition):** Обратная операция, которая предполагает разделение крупных понятий на более мелкие и конкретные концепты.

5. Операции по преобразованию:

Преобразование онтологий включает изменение их структуры или формы для оптимизации или соответствия определенным требованиям:

- **трансляция форматов:** Преобразование онтологии из одного формата в другой, например, из RDF в OWL. Это может понадобиться при переходе от одной платформы к другой;
- **абстрагирование:** Преобразование онтологии путем удаления деталей для создания более общей структуры. Операции по сравнению, проверке и оценке:
 - **сравнение онтологий:** Операция, при которой две или более онтологии анализируются для выявления их различий и сходств;
 - **проверка консистентности (Consistency Checking):** Операция, которая проверяет, не содержит ли онтология логических противоречий;
 - **оценка (Evaluation):** Процесс измерения качества онтологии на основе критериев, таких как полнота, точность, когерентность и полезность.

1.3 Предметная область онтологии

В качестве предметной области выбрана «Курсы по программированию». В рассматриваемой онтологии предполагается иерархия классов, в которой родительским классом является непосредственно абстрактный класс «Образовательные курсы». Образовательные курсы включают в себя курсы, студентов и преподавателей. В свою очередь, курсы подразделяются на очно-заочные курсы и онлайн курсы. Курсы также содержат такие элементы, как занятия, которые, в свою очередь, включают практические задания и тесты.. Иерархия классов показана на рисунке 1.1.

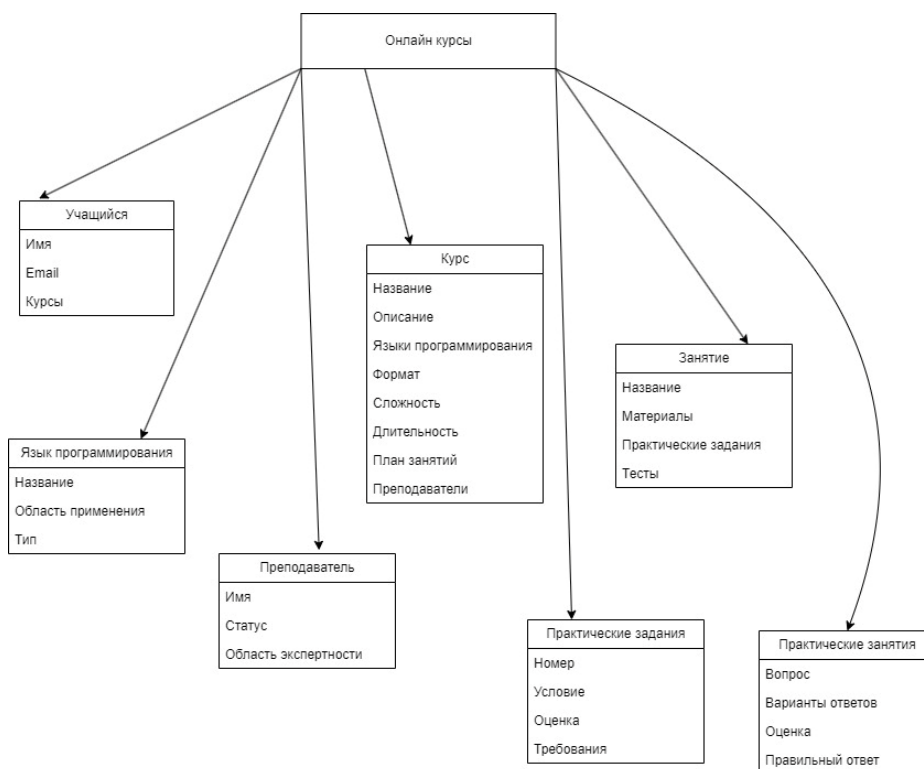


Рисунок 1.1 – Схема онтологии

1.4 Реализация онтологии в программе «Protégé»

Реализуем в программе Protégé онтологию согласно схеме. Создадим согласно схеме иерархию классов (рисунок 1.2).

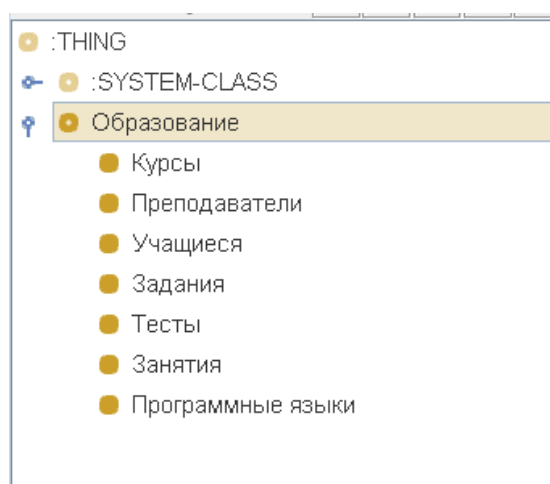


Рисунок 1.2 – Иерархия классов

Добавим необходимые слоты для класса «Программные языки» (рисунок 1.3).

The screenshot shows the 'Программные языки' class editor. The 'Name' field contains 'Программные языки' and the 'Documentation' field contains 'Языки программирования'. The 'Role' is set to 'Concrete'. The 'Template Slots' table is as follows:

Name	Cardinality	Type	Other Facets
Название	single	String	
область_применения	single	String	
тип_языка	single	String	

Рисунок 1.3 – Слоты класса «Программные языки»

Добавим необходимые слоты для класса «Задания» (рисунок 1.4).

The screenshot shows the 'Задания' class editor. The 'Name' field contains 'Задания' and the 'Documentation' field contains 'Задания'. The 'Role' is set to 'Concrete'. The 'Template Slots' table is as follows:

Name	Cardinality	Type	Other Facets
Номер задания	single	Integer	
оценка	single	Integer	
требования	multiple	String	
условие_задачи	single	String	

Рисунок 1.4 – Слоты класса «Задания»

Добавим необходимые слоты для класса «Тесты» (рисунок 1.5).

The screenshot shows the 'Тесты' class editor. The 'Name' field contains 'Тесты' and the 'Documentation' field contains 'Тесты'. The 'Role' is set to 'Concrete'. The 'Template Slots' table is as follows:

Name	Cardinality	Type	Other Facets
варианты_ответов	multiple	String	
вопросы	single	String	
оценка	single	Integer	
правильные_ответы	single	String	

Рисунок 1.5 – Слоты классов «Тесты»

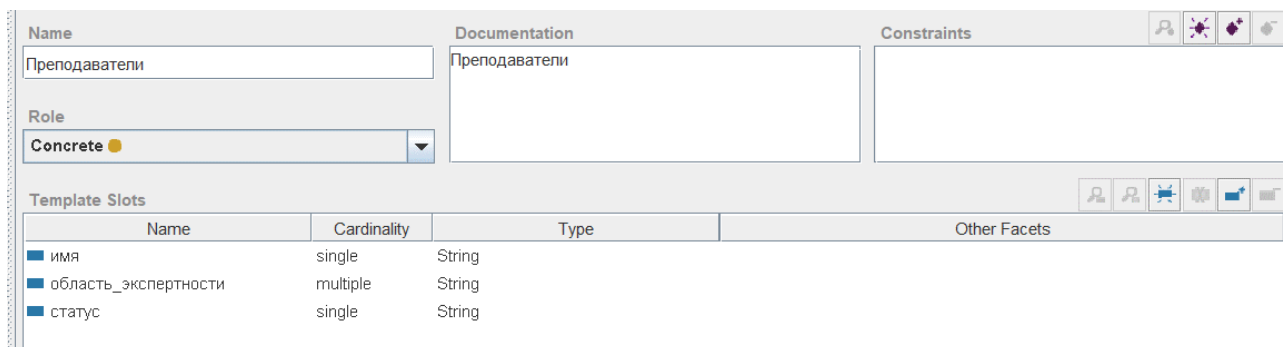
Добавим необходимые слоты для конкретного класса «Занятия» (рисунок 1.6).

The screenshot shows the 'Занятия' class editor. The 'Name' field contains 'Занятия' and the 'Documentation' field contains 'Занятие'. The 'Role' is set to 'Concrete'. The 'Template Slots' table is as follows:

Name	Cardinality	Type	Other Facets
материалы	single	String	
практические_задания	multiple	Instance of Задания	
тема_занятия	single	String	
тесты	single	Instance of Тесты	

Рисунок 1.6 – Слоты конкретного класса «Занятия»

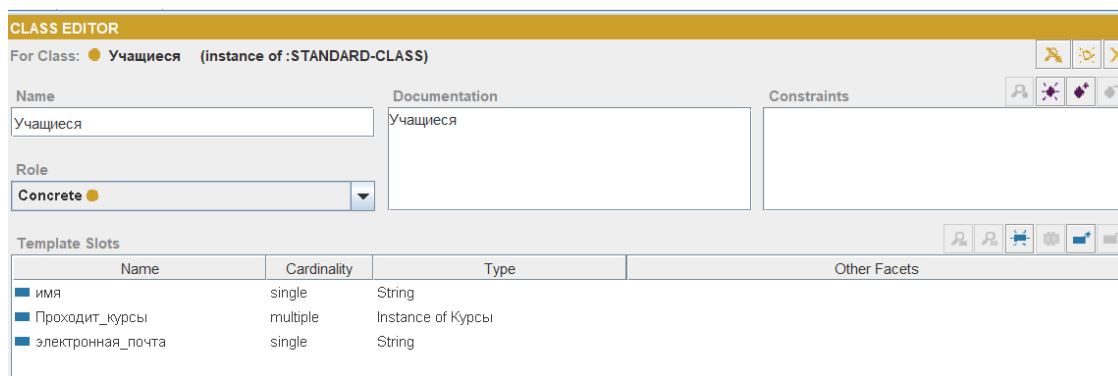
Класс «Занятия» содержит слот «практические задания», который ссылается на класс «Задания». Так же содержит слот «тесты», который ссылается на класс «Тесты». Созданы слоты для класса «Преподаватель» (Рисунок 1.7)



Name	Cardinality	Type	Other Facets
имя	single	String	
область_экспертности	multiple	String	
статус	single	String	

Рисунок 1.7 – Слоты конкретного класса «Преподаватель»

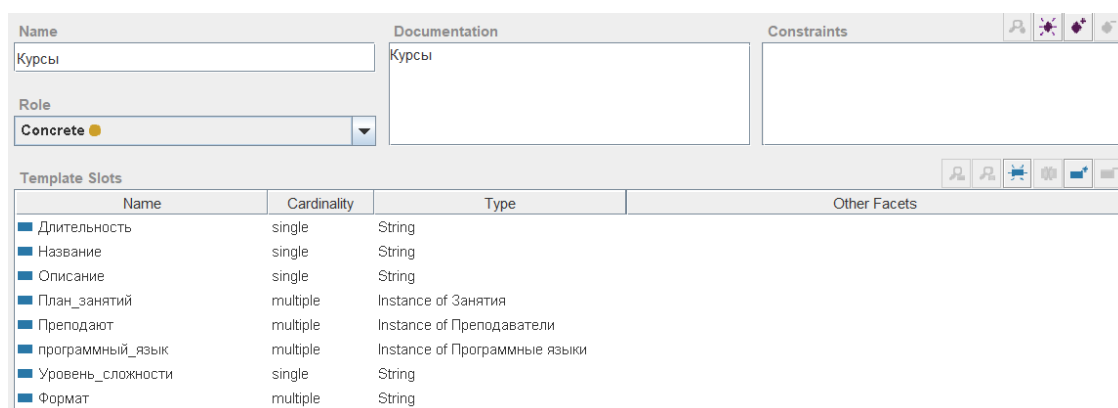
Созданы слоты для класса «Учащиеся» (Рисунок 1.8)



Name	Cardinality	Type	Other Facets
имя	single	String	
Проходит_курсы	multiple	Instance of Курсы	
электронная_почта	single	String	

Рисунок 1.7 – Слоты для класса «Учащиеся»

Класс «Учащиеся» содержит слот «Проходит_курсы», который указывается на курсы, которые проходит учащийся. Созданы слоты для класса «Курсы» (Рисунок 1.8)



Name	Cardinality	Type	Other Facets
Длительность	single	String	
Название	single	String	
Описание	single	String	
План_занятий	multiple	Instance of Занятия	
Преподают	multiple	Instance of Преподаватели	
программный_язык	multiple	Instance of Программные языки	
Уровень_сложности	single	String	
Формат	multiple	String	

Рисунок 1.9 – Слоты для класса «Курсы»

Класс «Курсы» имеет слоты, которые ссылаются на классы «Занятия», «Преподаватели», «Программные языки».

Теперь добавим объекты всех не абстрактных классов и заполним поля этих классов, тем самым в том числе организовав связи между ними.

Добавим и заполним объекты класса «Занятие» (рисунок 1.9).

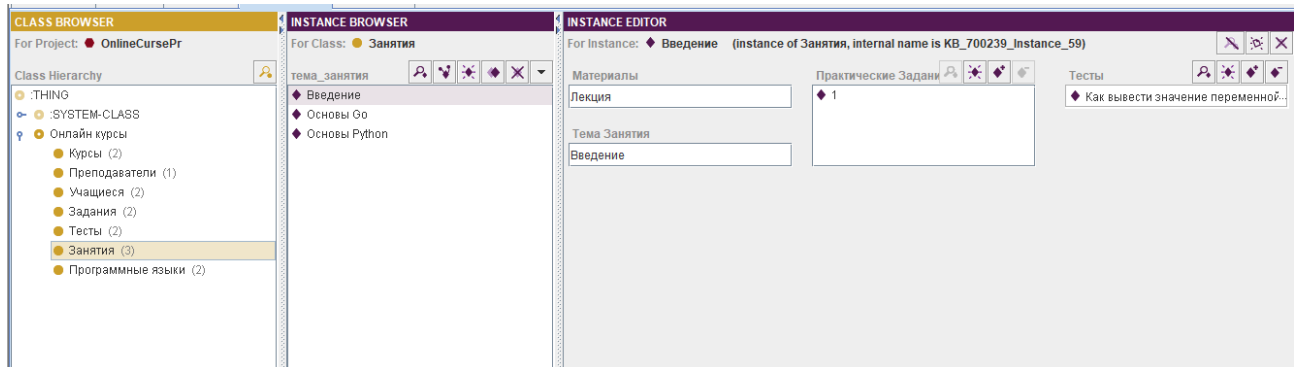


Рисунок 1.9 – Объекты класса «Занятие»

Добавим и заполним объекты класса «Программные языки» (рисунок 1.10).

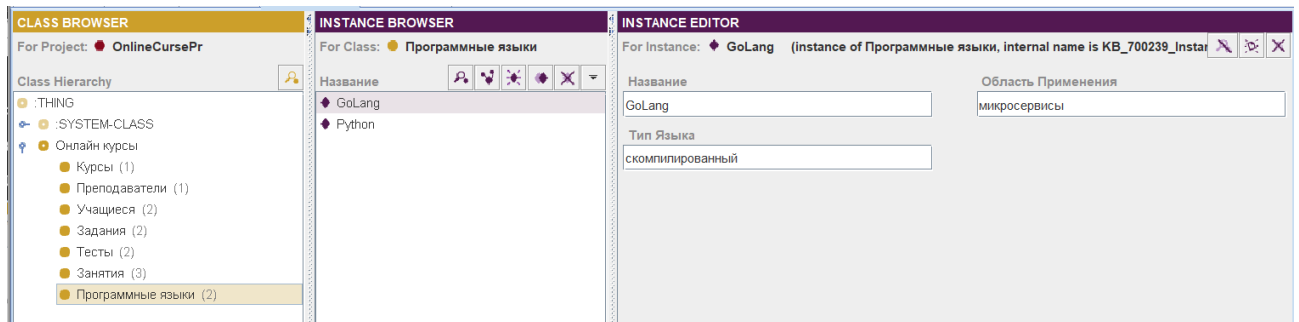


Рисунок 1.10 – Объекты класса «Программные языки»

Добавим и заполним объекты класса «Тесты» (рисунок 1.11).

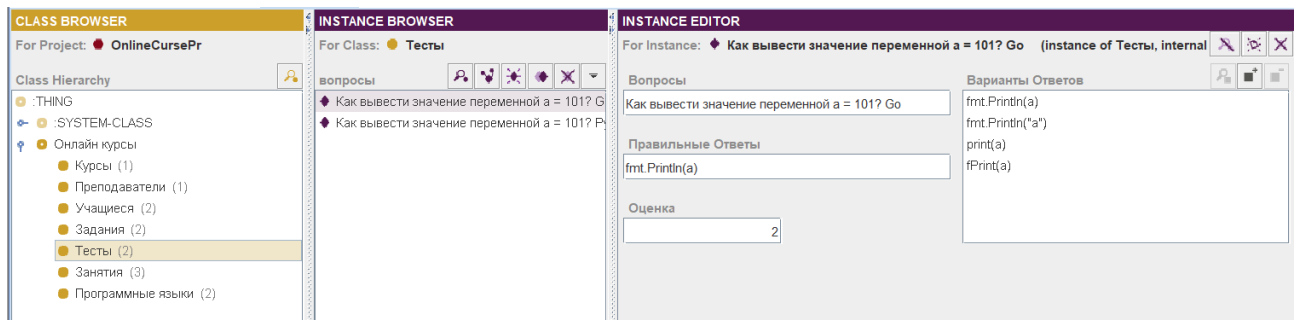


Рисунок 1.11 – Объекты класса «Тесты»

Добавим и заполним объекты класса «Задания» (рисунок 1.12).

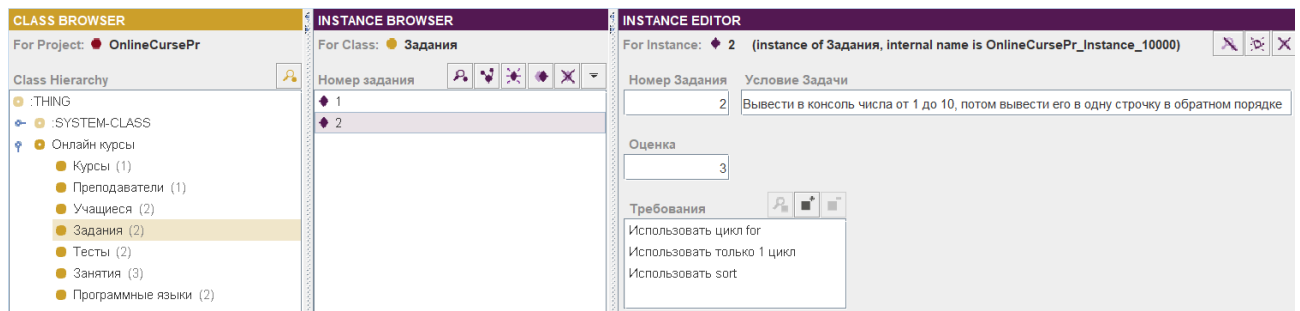


Рисунок 1.12 – Объекты класса «Задания»

Добавим и заполним объекты класса «Учащиеся» (рисунок 1.13).

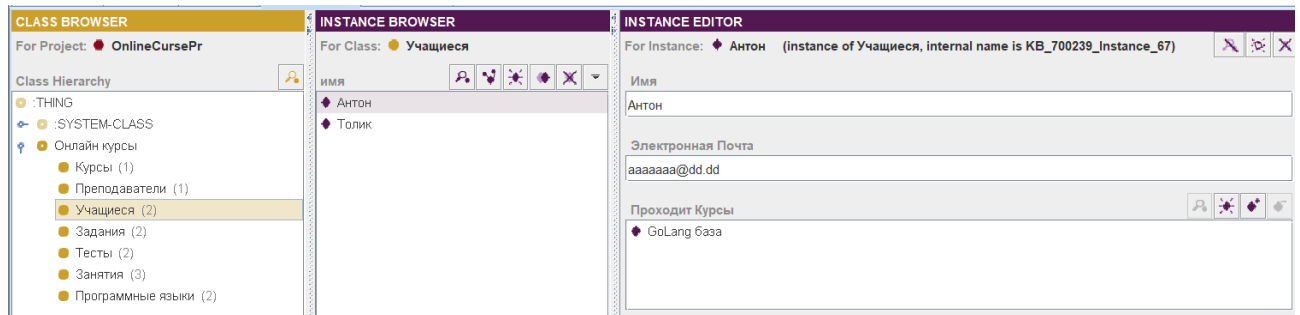


Рисунок 1.13 – Объекты класса «Учащиеся»

Добавим и заполним объекты класса «Преподаватели» (рисунок 1.14).

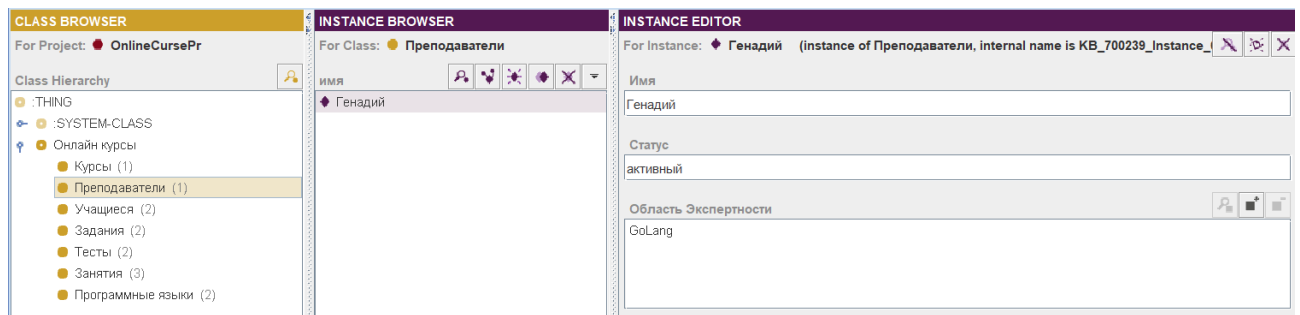


Рисунок 1.14 – Объекты класса «Преподаватели»

Добавим и заполним объекты класса «Курсы» (рисунок 1.15).

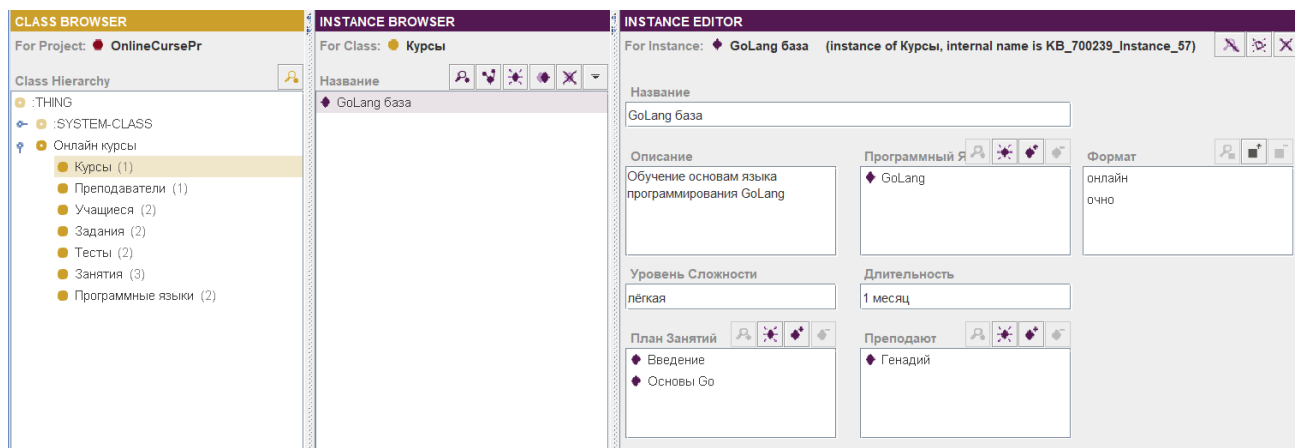


Рисунок 1.15 – Объекты класса «Курсы»

Теперь сформируем несколько запросов чтобы протестировать работу созданной нами онтологии.

Запросим объекты из класса «Курсы», которые содержат в себе язык программирования “GoLang” (рисунок 1.16).

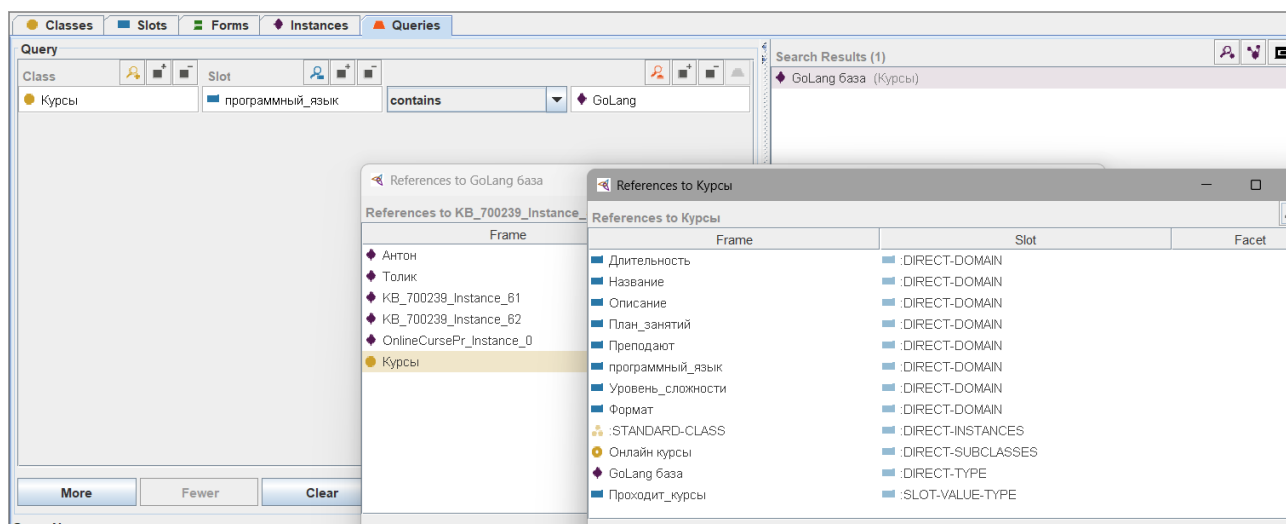


Рисунок 1.16 – Результаты первого запроса

Запросим объект класса «Курсы», которые по слоту «Преподают» Геннадий. (рисунок 1.17).

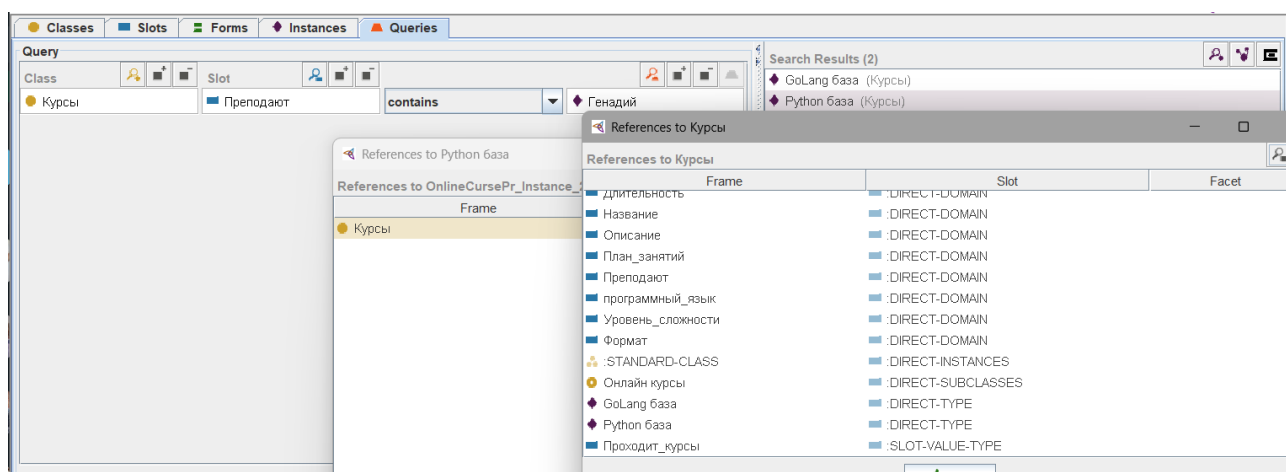


Рисунок 1.17 – Результаты второго запроса

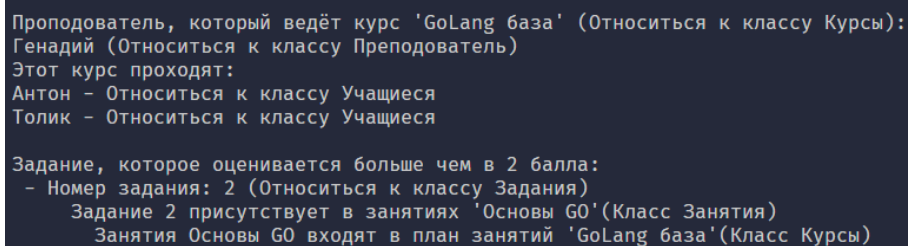
1.5 Реализация онтологии на языке программирования GoLang

Реализуем описанную модель антологии с соответствующими классами их полями, обеспечив необходимые связи между ними.

Реализуем на языке программирования GoLang программу, отражающую онтологию, составляющую её объекты и связи между ними (Приложение А).

Протестируем работу программы, сделав несколько запросов и сравнив ответы на эти запросы с ответами, полученными на аналогичные запросы в программе Protégé.

Запросим преподавателей, которые ведут курс “GoLang база”. Так же запросим задания, которые оцениваются больше чем в 2 балла (рисунок 1.19).



```
Преподаватель, который ведёт курс 'GoLang база' (Относиться к классу Курсы):  
Генадий (Относиться к классу Преподаватель)  
Этот курс проходят:  
Антон - Относиться к классу Учащиеся  
Толик - Относиться к классу Учащиеся  
  
Задание, которое оценивается больше чем в 2 балла:  
- Номер задания: 2 (Относиться к классу Задания)  
  Задание 2 присутствует в занятиях 'Основы GO'(Класс Занятия)  
  Занятия Основы GO входят в план занятий 'GoLang база'(Класс Курсы)
```

Рисунок 1.19 – Результаты первого запроса

2 АЛГОРИТМ ОТЖИГ

Метод имитации отжига является стохастическим алгоритмом оптимизации, который широко применяется для поиска глобального минимума сложных функций на дискретных или непрерывных пространствах. Основная идея алгоритма заимствована из процесса физического отжига – термической обработки материалов, при которой система постепенно охлаждается, что позволяет ей стабилизироваться в состоянии с минимальной энергией.

В задачах оптимизации элементы пространства решений S можно рассматривать как состояния системы, а функция $f(x)$ – как энергию системы, соответствующую состоянию x . Алгоритм последовательно перемещается по пространству решений, пытаясь улучшить текущее состояние системы, но допускает временные ухудшения, чтобы избежать локальных минимумов. Это достигается за счет управляющего параметра температуры T , которая постепенно снижается, ограничивая вероятность принятия худших решений на поздних этапах.

Алгоритм начинается с выбора начального пути V_0 и вычисления его длины. На каждой итерации генерируется новый путь с использованием вероятностного распределения, зависящего от текущей температуры. Принятие нового состояния определяется вероятностью, зависящей от разности энергий текущего и нового состояний, а также от температуры. Таким образом, при высокой температуре алгоритм допускает большее количество переходов к менее эффективным решениям, что помогает исследовать пространство решений более глобально. По мере снижения температуры система постепенно «замораживается» на решении, близком к глобальному минимуму.

Метод отжига особенно эффективен для задач с большими пространствами поиска, где прямые методы оптимизации могут застревать в локальных минимумах.

2.1 Описание алгоритма отжига задача коммивояжера

Алгоритм имитации отжига для решения задачи коммивояжера базируется на стохастическом подходе к поиску глобального экстремума в пространстве возможных решений. В данной задаче цель состоит в нахождении наилучшего маршрута, который минимизирует общую длину пути, при условии, что коммивояжер должен посетить все отделы супермаркета ровно один раз и вернуться в исходную точку. Основная идея алгоритма заключается в том, чтобы позволить системе «охлаждаться» постепенно, уменьшая вероятность принятия менее удачных решений на поздних этапах поиска, что помогает избегать локальных минимумов и способствует нахождению глобального решения.

После нахождения нового маршрута вычисляется его длина и приращение длины (ΔS). Если длина нового маршрута меньше длины текущего маршрута ($\Delta S < 0$), то новое решение принимается. В противном случае решение принимается с некоторой вероятностью, которая зависит от приращения длины и текущей температуры.

Когда новый маршрут хуже текущего ($\Delta S > 0$), решение принимается с вероятностью по формуле (2.1):

$$P_* = T e^{-\frac{\Delta S}{T}}. \quad (2.1)$$

2.2 Ручной расчет алгоритма отжига коммивояжера

Для начала зададим длины ребер

Таблица 2.1 – Длины ребер

Ребро	$L_{i,j}$	Ребро	$L_{i,j}$
1 – 2	25	2 – 6	12
1 – 3	31	3 – 4	23
1 – 4	22	3 – 5	40
1 – 5	15	3 – 6	33
1 – 6	21	4 – 5	24
2 – 3	45	4 – 6	19

Продолжение таблицы 2.1

2 – 4	27	5 – 6	18
2 – 5	25		

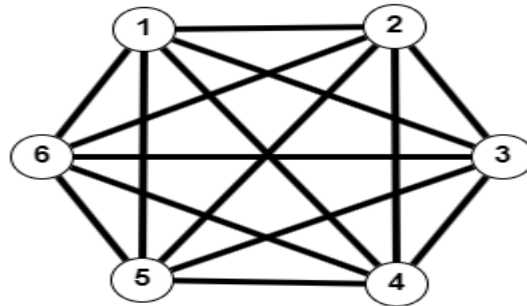


Рисунок 2.1 – Полный граф

1. Находим длину первоначального маршрута $V_0 = [1, 3, 4, 5, 6, 2, 1]$ (рисунок 2.2).

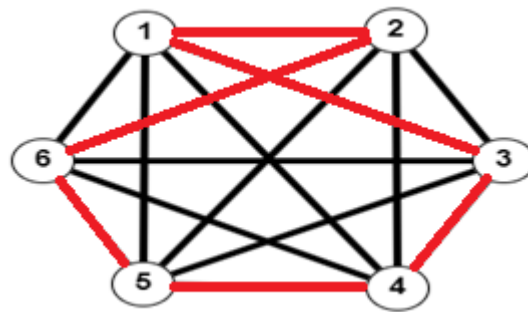


Рисунок 2.2 – Первоначальный маршрут V_0

$$\begin{aligned}
 S_0 &= L_{1,3} + L_{3,4} + L_{4,5} + L_{5,6} + L_{6,2} + L_{2,1} = \\
 &= 31 + 23 + 24 + 18 + 12 + 25 = 133.
 \end{aligned}$$

Меняем порядок прохождения вершин. По условию задачи меняем местами четвертый и шестой элемент списка прохождения, т.е. вершины 5 и 2. Получаем новый список $V_1 = [1, 3, 4, 2, 6, 5, 1]$ (рисунок 2.3).

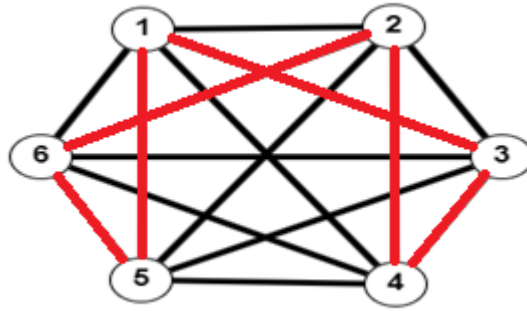


Рисунок 2.3 – Маршрут V_1

Находим длину соответствующего маршрута:

$$\begin{aligned} S_1 &= L_{1,3} + L_{3,4} + L_{4,2} + L_{2,6} + L_{6,5} + L_{5,1} = \\ &= 31 + 23 + 27 + 12 + 18 + 15 = 126. \end{aligned}$$

Вычисляем приращение длины $\Delta S_1 = S_1 - S_0 = 126 - 133 = -7$. Так как приращение отрицательное, т.е. длина уменьшилась (решение улучшилось), то маршрут V_1 принимается. Одновременно уменьшается температура $T_2 = 0,5 * T_1 = 50$.

2. Выполняем очередную замену. Меняем местами пятый и шестой элемент списка порождения вершин. Получаем последовательность $V_2 = [1, 3, 4, 2, 5, 6, 1]$ (рисунок 2.4).

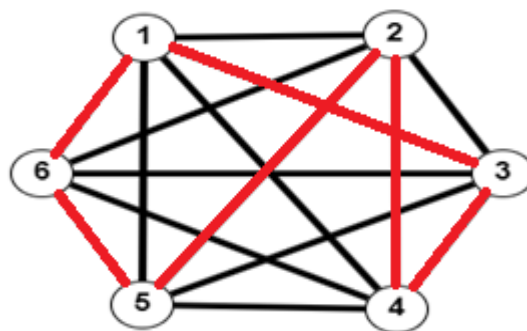


Рисунок 2.4 – Маршрут V_2

Находим длину маршрута V_2 :

$$S_2 = L_{1,3} + L_{3,4} + L_{4,2} + L_{2,5} + L_{5,6} + L_{6,1} = \\ = 31 + 23 + 27 + 25 + 18 + 21 = 145.$$

Вычисляем приращение длины $\Delta S_2 = S_2 - S_1 = 145 - 126 = 19$. Полученный маршрут оказался длинее. Брать этот маршрут в качестве очередного в процедуре поиска минимума или не брать, зависит от вероятности перехода, вычисленной по формуле (2.1).

$$P_* = T_1 e^{-\frac{\Delta S_2}{T_2}}.$$

От выпавшего в генераторе случайных чисел значения P_2 (значения P_1 оказалось в решении не задействовано, на первом этапе полученный маршрут уменьшился). Так как после первого этапа температура уменьшилась, поэтому $T_2 = 50$, то:

$$P_* = 100e^{-\frac{19}{50}} = 68.38 > 59 = P_2.$$

Выпавшее в генераторе случайных чисел P_2 попало в требуемый диапазон $[0, 68.38]$, следовательно, маршрут V_2 принимается в качестве очередного, несмотря на то, что длина больше. Одновременно уменьшается температура $T_3 = 0,5T_2 = 25$.

2.3 Программная реализация алгоритма отжига коммивояжера

Выполним программную реализацию на языке GoLang (Приложение Б). В результате работы программы нам необходимо получить оптимальный маршрут движения (обхода графа), а также его длину. Запустим написанную программу и убедимся в правильности её работы:

```

Итерация 88: Текущее расстояние 571.211731, Лучшее расстояние 554.246131, Температура 41.29496711338884
Итерация 89: Текущее расстояние 571.211731, Лучшее расстояние 543.526991, Температура 40.882017442254956
Итерация 90: Текущее расстояние 543.526991, Лучшее расстояние 543.526991, Температура 40.473197267832404
Итерация 91: Текущее расстояние 578.008104, Лучшее расстояние 543.526991, Температура 40.06846529515408
Итерация 92: Текущее расстояние 616.914728, Лучшее расстояние 543.526991, Температура 39.66778064220254
Итерация 93: Текущее расстояние 616.914728, Лучшее расстояние 543.526991, Температура 39.27110283578051
Итерация 94: Текущее расстояние 616.914728, Лучшее расстояние 543.526991, Температура 38.8783918074227
Итерация 95: Текущее расстояние 616.914728, Лучшее расстояние 543.526991, Температура 38.48960788934848
Итерация 96: Текущее расстояние 616.914728, Лучшее расстояние 543.526991, Температура 38.104711810454994
Итерация 97: Текущее расстояние 616.929351, Лучшее расстояние 543.526991, Температура 37.72366469235045
Итерация 98: Текущее расстояние 560.527907, Лучшее расстояние 543.526991, Температура 37.34642804542694
Итерация 99: Текущее расстояние 560.527907, Лучшее расстояние 543.526991, Температура 36.97296376497267
Итерация 100: Текущее расстояние 560.770376, Лучшее расстояние 543.526991, Температура 36.60323412732294
Лучший найденный маршрут:
Нью-Йорк (40.71, -74.01)
Лондон (51.51, -0.13)
Париж (48.86, 2.35)
Москва (55.76, 37.62)
Сидней (-33.87, 151.21)
Токио (35.69, 139.69)
Пекин (39.90, 116.41)
Рим (41.90, 12.50)
Общее расстояние: 543.526991

```

Рисунок 2.5 – Результат работы программы алгоритма отжига коммивояжера

2.4 Описание алгоритма отжига Коши (Быстрый отжиг)

Алгоритм отжига Коши, также известный как быстрый отжиг, представляет собой модификацию классического алгоритма имитации отжига, в котором процесс охлаждения системы происходит быстрее благодаря использованию специального закона изменения температуры. В отличие от традиционного метода, где температура снижается по геометрической или экспоненциальной схеме, в алгоритме Коши температура уменьшается по гиперболическому закону, что делает процесс поиска оптимального решения более интенсивным на ранних этапах. Этот метод может быть особенно полезен

при решении задач с большим пространством поиска и высокой вычислительной сложностью, таких как задача коммивояжера.

Изменение температуры в алгоритме отжига Коши вычисляется по формуле (2.2), где k – номер текущей итерации.

$$T(k) = \frac{T_{k-1}}{k}; k > 0. \quad (2.2)$$

Алгоритм продолжается до тех, пор пока температура не опуститься ниже заданного минимального значения T_{min} .

2.5 Принцип работы алгоритма отжига быстрого Коши

Алгоритм отжига Коши можно адаптировать для оптимизации многомерных функций, включая функции с переменными x и y . Основная идея остается прежней – использовать стохастический подход для нахождения минимума функции, но с учетом многомерности пространства решений.

Выбранная функция: Функция Розенброка.

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2,$$

глобальный минимум:

$$f(1, 1 \dots) = 0,$$

метод поиска:

$$\begin{aligned} -\infty &\leq x_i \leq \infty, \\ 1 &\leq i \leq n \end{aligned}$$

2.6 Ручной расчет алгоритма отжига Коши

Для решения задачи были выбраны следующие точки: $x_0 = -1, y_0 = 1$.

$$f_0 = (1 - (-1))^2 + 100(1 - (-1)^2)^2 = 4$$

Выбираем новые точки для расчета изменения энергии: $x_1 = -0.2, y_1 = 0.6$.

$$f_1 = (1 - (-0.2))^2 + 100(0.6 - (-0.2)^2)^2 = 32.8$$

Так как новое состояние больше предыдущего рассчитываем разницу энергии. $\Delta f_1 = f_1 - f_0 = 4 - 32.8 = 28.8$. Рассчитаем температуру после изменения по формуле (1.2).

$$T_1 = \frac{T_0}{k} = \frac{100}{1} = 100.$$

Будем ли мы брать данное состояние зависит от вероятности перехода, вычисленной по формуле (2.3), где k – номер текущей итерации.

$$P_* = e^{-\frac{\Delta f_k}{T_k}}, \quad (2.3)$$

подставим значения в формулу (2.3)

$$P_1 = e^{-\frac{28.8}{100}} = 0.756 < 0.9.$$

Выпавшее в генераторе случайных чисел P_1 не попало в требуемый диапазон, следовательно, решение отклонено.

Выбираем очередные новые точки: $x_2 = 2, y_2 = 1$.

$$f_2 = (1 - (-1.5))^2 + 100(1.3 - (-1.5)^2)^2 = 96.5$$

Вычисляем разницу энергии $\Delta f_2 = f_2 - f_0 = 96.5 - 4 = 92.5$.

Одновременно уменьшается температура $T_2 = \frac{T_1}{k} = \frac{100}{2} = 50$.

2.7 Программная реализация алгоритма отжига Коши

Также, ссылаясь на Приложение В, реализован алгоритм отжига Коши в виде программы. На рисунке 2.6 приведен результат работы программы.

```
Оптимальные параметры для стабилизации полета дрона:  
Угол наклона (theta): 1.0019  
Ускорение (a): 1.0006  
Минимальная энергия (функция Розенброка): 0.0001
```

Рисунок 2.6 – Результат работы программы алгоритма отжига Коши

3 ОСНОВНОЙ РОЕВОЙ АЛГОРИТМ

Метод роя, также известный как метод частиц или метод оптимизации роя частиц, является эффективным алгоритмом оптимизации, который использует идеи из природы для решения сложных задач оптимизации. Он основан на поведении роя птиц или рыб, которые в природе демонстрируют коллективную интеллектуальную активность для достижения общей цели. Метод роя является одним из наиболее популярных алгоритмов оптимизации и широко применяется в различных областях, таких как машинное обучение, искусственный интеллект, экономика и финансы. Он позволяет эффективно находить оптимальное решение в сложных и многомерных пространствах, не требуя больших вычислительных ресурсов. В этом методе каждая частица представляет собой потенциальное решение задачи, а их движение в пространстве определяется лучшими решениями, найденными ранее.

Дополнительно, метод роя обладает рядом преимуществ, которые способствуют его популярности среди исследователей и практиков. Во-первых, алгоритм PSO прост в реализации и не требует сложных математических преобразований, что облегчает его применение в широком спектре задач. Во-вторых, PSO обладает высокой гибкостью и может быть легко адаптирован под специфические требования различных приложений путем настройки параметров, таких как коэффициенты инерции и ускорения. Кроме того, метод роя демонстрирует хорошую сходимость и устойчивость к попаданию в локальные минимумы, что делает его особенно ценным при решении задач с нелинейными и многомерными функциями. Также, благодаря своей параллельной природе, PSO эффективно использует современные вычислительные ресурсы, что позволяет значительно ускорить процесс оптимизации при обработке больших объемов данных. Все эти характеристики делают метод роя частиц одним из наиболее универсальных и надежных инструментов в арсенале алгоритмов оптимизации.

3.1 Постановка цели и задачи

Найти минимум функции Розенброка, данная функция определена на интервале $[-\infty; \infty]$.

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2,$$

3.2 Описание алгоритма

Роевой алгоритм (РА) 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 фитнес-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. Каждая частица сохраняет значения координат своей траектории с соответствующими лучшими значениями целевой функции, которые обозначим y_i , которая отражает когнитивную компоненту. Аналогично значение глобального оптимума, достигнутого частицами роя, будем обозначать \hat{y}_J , которое отражает социальную компоненту. Таким образом, каждая частица роя подчиняется достаточно простым правилам поведения (изложенным ниже формально), которые

учитывают локальный успех каждой особи и глобальный оптимум всех особей (или некоторого множества соседей) роя.

Каждая i -я частица характеризуется в момент времени t своей позицией $x_i(t)$ в гиперпространстве и скоростью движения $v_i(t)$. Позиция частицы изменяется в соответствии с формулой (1.1):

$$x_i(t + 1) = x_i(t) + v_i(t + 1), \quad (3.1)$$

где $x_{i_0} \sim (x_{min}, x_{max})$.

Вектор скорости $v_i(t + 1)$ управляет процессом поиска решения и его компоненты определяются с учетом когнитивной и социальной составляющей по формуле:

$$v_{ij}(t + 1) = wv_i(t) + c_1r_1[y_{ij}(t) - x_{ij}(t)] + c_2r_2[\hat{y}_j(t) - x_{ij}(t)]. \quad (3.2)$$

Здесь $v_i(t) - j$ -я компонента скорости ($j = 1, \dots, n_x$) i -ой частицы в момент времени t , $x_{ij}(t) - j$ -я координата позиции i -ой частицы, c_1 и c_2 – положительные коэффициенты ускорения (часто полагаемые 2), регулирующие вклад когнитивной и социальной компонент, r_1 и r_2 – случайные числа из диапазона $[0,1]$, которые генерируются в соответствии с нормальным распределением и вносят элемент случайности в процесс поиска. Кроме этого $y_{ij}(t)$ – персональная лучшая позиция по j -ой координате i -ой частицы, а $\hat{y}_j(t)$ – лучшая глобальная позиция роя, где целевая функция имеет экстремальное значение.

При решении задач минимизации персональная лучшая позиция в следующий момент времени $(t + 1)$ определяется по формуле (3.3):

$$y_i(t + 1) = \begin{cases} y_i(t), & \text{если } f(x_i(t + 1)) \geq f(y_i(t)) \\ x_i(t + 1), & \text{если } f(x_i(t + 1)) < f(y_i(t)), \end{cases} \quad (3.3)$$

где $f: R^{n\infty} \rightarrow R$ фитнесс-функция. Как и в эволюционных алгоритмах, фитнесс-функция измеряет близость текущего решения к оптимуму.

Существует два основных подхода в оптимизации роя частиц, под названиями “lbest” и “gbest”, отличающиеся топологией соседства, используемой для обмена опытом между частицами. Для модели “gbest” лучшая частица определяется из всего роя. Глобальная лучшая позиция “gbest” $\hat{y}_J(t)$ в момент t определяется в соответствии с формулой (3.4):

$$\hat{y}_J(t) \in \{y_0(t), \dots, y_{n_s}(t)\}, f(\hat{y}_J(t)) = \min \{f(y_0(t)), \dots, f(y_{n_s}(t))\}, \quad (3.4)$$

где n_s – общее число частиц в рое.

В процессе поиска решения описанные действия выполняются для каждой частицы роя.

3.3 Ручной расчет роевого алгоритма

Заданная целевая функция (функция трехгорбового верблюда):

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2,$$

На примере трех частиц покажем работу алгоритма в течение одной итерации. Начальные условия:

1. Количество частиц: $M = 3$.
2. Количество итераций: $L = 1$.
3. Инерция: $w = 0.7$.
4. Параметры в изменении скорости: $c_1 = 2, c_2 = 2$. Интервал поиска решений: $x \in [-5; 5], y \in [-5; 5]$.
5. Начальная скорость $v_i = 0$

Случайным образом проинициализируем стартовые положения частиц так, чтобы они принадлежали данному диапазону:

$$x_{1_0} = 1; x_{2_0} = -1; x_{3_0} = 0,$$

$$y_{1_0} = 1; y_{2_0} = 2; y_{3_0} = -1,$$

$$f(x_{1_0}, y_{1_0}) = 0; f(x_{2_0}, y_{2_0}) = 104; f(x_{3_0}, y_{3_0}) = 101.$$

Лучшее значение целевой функции $f(x_{1_0}, y_{1_0}) = 0$.

1. Обновим скорость первой частицы по x : $v_{x_1} = 0.7 \cdot 0 + 2 \cdot 0.4 \cdot (1 - 1) + 2 \cdot (1 - 1) = 0$.

Обновим скорость первой частицы по y : $v_{y_1} = 0.7 \cdot 0 + 2 \cdot 0.4 \cdot (1 - 1) + 2 \cdot (1 - 1) = 0$.

Обновим позицию первой частицы по x : $x_{1_1} = x_{1_0} + v_{x_1} = 1 + 0 = 1$.

Обновим позицию первой частицы по y : $y_{1_1} = y_{1_0} + v_{y_1} = 1 + 0 = 1$.

Оценка фитнеса: $f(1; 1) = 0$.

После работы с первой частицей не обновляем личный лучший, так как, $0 \geq 0$.

2. Обновим скорость второй частицы по x : $v_{x_2} = 0.7 \cdot 0 + 2 \cdot 0.3 \cdot (-1 - (-1)) + 2 \cdot 0.5 \cdot (1 - (-1)) = 2$.

Обновим скорость второй частицы по y : $v_{y_2} = 0.7 \cdot 0 + 2 \cdot 0.3 \cdot (2 - 2) + 2 \cdot 0.5 \cdot (1 - 2) = -1$.

Обновим позицию второй частицы по x : $x_{2_1} = x_{2_0} + v_{x_2} = -1 + 2 = 1$.

Обновим позицию второй частицы по y : $y_{2_1} = y_{2_0} + v_{y_2} = 2 - 1 = 1$.

Оценка фитнеса: $f(1; 1) = 0$.

После работы со второй частицей обновляем личный лучший, так как, $0 < 104$.

3. Обновим скорость третьей частицы по x : $v_{x_3} = 0.7 \cdot 0 + 2 \cdot 0.2 \cdot (0 - 0) + 2 \cdot 0.7 \cdot (1 - 0) = 1.4$.

Обновим скорость третьей частицы по y : $v_{y_3} = 0.7 \cdot 0 + 2 \cdot 0.2 \cdot (-1 - (-1)) + 2 \cdot 0.7 \cdot (1 - (-1)) = 2.8$.

Обновим позицию третьей частицы по x : $x_{3_1} = x_{3_0} + v_{x_3} = 0 + 1.4 = 1.4$.

Обновим позицию третьей частицы по y : $y_{3_1} = y_{3_0} + v_{y_3} = -1 + 2.8 = 1.8$.

Оценка фитнеса: $f(1.4; 1.8) = 2.56$.

После работы с третьей частицей обновляем личный лучший, так как, $2.56 < 101$.

После расчета первой итерации, глобальное лучшее значение остается неизменным и равным 0, глобальное лучшее положение тоже остается неизменным – (1; 1).

3.4 Программная реализация роевого алгоритма

Выполним программную реализацию алгоритма роя частиц (PSO) на языке GoLang (Приложение Г). Результат программы представлен на рисунках 3.1. В результате работы программы необходимо найти оптимальное значение функции трёхгорбого верблюда, а также соответствующие координаты точки минимума. Запустим написанную программу и убедимся в правильности её работы:

```

Итерация 76: Лучшее значение функции = 0.000224 в позиции [0.9895324340614572 0.9802441060914888]
Итерация 77: Лучшее значение функции = 0.000206 в позиции [0.9891372229052839 0.9774519473526091]
Итерация 78: Лучшее значение функции = 0.000206 в позиции [0.9891372229052839 0.9774519473526091]
Итерация 79: Лучшее значение функции = 0.000206 в позиции [0.9891372229052839 0.9774519473526091]
Итерация 80: Лучшее значение функции = 0.000206 в позиции [0.9891372229052839 0.9774519473526091]
Итерация 81: Лучшее значение функции = 0.000206 в позиции [0.9891372229052839 0.9774519473526091]
Итерация 82: Лучшее значение функции = 0.000206 в позиции [0.9891372229052839 0.9774519473526091]
Итерация 83: Лучшее значение функции = 0.000048 в позиции [0.9930795220509285 0.9862264787466368]
Итерация 84: Лучшее значение функции = 0.000048 в позиции [0.9930795220509285 0.9862264787466368]
Итерация 85: Лучшее значение функции = 0.000016 в позиции [1.0032143457205431 1.0062044240873558]
Итерация 86: Лучшее значение функции = 0.000016 в позиции [1.0032143457205431 1.0062044240873558]
Итерация 87: Лучшее значение функции = 0.000016 в позиции [1.0032143457205431 1.0062044240873558]
Итерация 88: Лучшее значение функции = 0.000016 в позиции [1.0032143457205431 1.0062044240873558]
Итерация 89: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 90: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 91: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 92: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 93: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 94: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 95: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 96: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 97: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 98: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Итерация 99: Лучшее значение функции = 0.000001 в позиции [0.9988202286798821 0.9976397573672612]
Оптимизация завершена
Лучшая позиция: [0.9988202286798821 0.9976397573672612]
Лучшее значение функции: 0.000001

```

Рисунок 3.1 – Результат работы программы основного роевого алгоритма

4 МУРАВЬИНЫЙ АЛГОРИТМ

Муравьиный алгоритм – это метаэвристический метод оптимизации, вдохновленный поведением реальных муравьев при поиске оптимальных путей к источнику пищи. Основная идея алгоритма заключается в том, что муравьи, перемещаясь по окружающей среде, оставляют на своем пути особые химические вещества – феромоны, которые служат сигнальными метками для других муравьев.

Когда муравей сталкивается с выбором между несколькими путями, вероятность выбора того или иного пути зависит от количества феромонов: чем больше феромонов на пути, тем он привлекательнее. По мере того, как больше муравьев выбирают определенные маршруты и откладывают на них феромоны, эти пути становятся все более предпочтительными. Важно, что феромоны со временем испаряются, что предотвращает застревание алгоритма на неэффективных маршрутах и позволяет исследовать новые возможности.

Благодаря этому механизму взаимодействия и адаптации, муравьиный алгоритм способен эффективно решать сложные задачи оптимизации, такие как задача коммивояжера, планирование маршрутов и оптимизация расписания. Алгоритм обладает способностью к самоорганизации и нахождению близких к оптимальным решениям, даже в условиях сложного и большого пространства решений.

4.1 Постановка задачи

Необходимо найти оптимальный путь обхода графа используя муравьиный алгоритм. Граф изображен на рисунке 4.1.

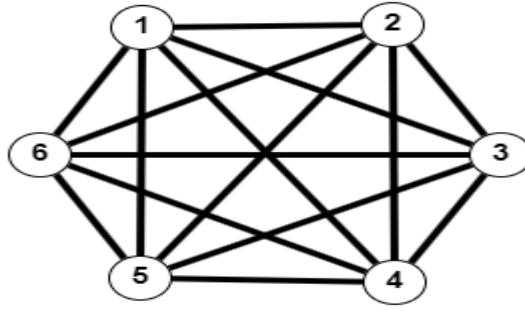


Рисунок 4.1 – Полный граф

4.2 Описание муравьиного алгоритма

В начале работы алгоритма каждой дуге графа присваивается небольшое начальное значение феромона τ_{ij} . Параметры α и β определяют степень влияния феромонов и эвристической информации, соответственно. Муравьи размещаются в вершинах графа, и каждый муравей начинает строить маршрут, стремясь посетить каждую вершину ровно один раз, прежде чем вернуться в начальную.

Каждый муравей последовательно выбирает следующую вершину, основываясь на вероятности, которая определяется значением феромонов и эвристической информацией. Если муравей находится в вершине i , вероятность выбора следующей вершины j рассчитывается следующим образом:

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{k \in N_i} \tau_{ik}^{\alpha} \eta_{ik}^{\beta}}, \quad (4.1)$$

где:

- τ_{ij} – концентрация феромона на дуге;
- $\eta_{ij} = \frac{1}{d_{ij}}$ – эвристическая информация, обратная расстоянию d_{ij} ;
- α – параметр, определяющий влияние феромонов;
- β – параметр, определяющий влияние эвристической информации;

- N_i – множество вершин, которые муравей еще не посетил.

Муравей продолжает движение, пока не пройдет через все вершины и не вернется в исходную вершину, формируя замкнутый маршрут.

После того как все муравьи завершили свои маршруты, выполняется обновление феромонов по следующим правилам:

1. Испарение феромонов: Чтобы предотвратить бесконечное накопление феромонов и дать возможность исследовать новые маршруты, феромоны на всех дугах уменьшаются:

$$\tau_{ij} \leftarrow (1 - \rho) * \tau_{ij}, \quad (4.2)$$

где:

- τ_{ij} – текущее значение феромона на дуге (i, j) ;
- ρ – коэффициент испарения ($0 < \rho < 1$).

2. Добавление феромонов: Муравьи оставляют феромоны на пройденных маршрутах. Количество добавляемого феромона обратно пропорционально длине маршрута:

$$\Delta\tau_{ij} = \frac{1}{L_k}, \quad (4.3)$$

где:

- $\Delta\tau_{ij}$ – количество феромона, добавляемого на дугу (i, j) ;
- L_k – длина маршрута, пройденного k -м муравьем.

Полная формула обновления феромонов выглядит таким образом:

$$\tau_{ij}(t + 1) = (1 - \rho) * \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k. \quad (4.4)$$

Алгоритм включает механизм испарения феромонов, чтобы обеспечить баланс между исследованием новых маршрутов и использованием уже найденных хороших решений. Быстрое испарение (ρ близкое к 1) делает поиск

более случайным, а медленное испарение (ρ близкое к 0) усиливает ранее найденные пути.

Алгоритм может завершиться по одному из следующих условий:

- достигнуто заданное количество итераций;
- найдено решение с приемлемой длиной маршрута;
- все муравьи начинают использовать один и тот же путь, что свидетельствует о стабилизации решения.

Таким образом, муравьиный алгоритм сочетает в себе локальное поведение (выбор следующей вершины) и коллективное взаимодействие (использование феромонов), чтобы находить приближенные решения задач оптимизации.

4.3 Ручной расчет муравьиного алгоритма

Данные для расчета:

- количество муравьев: 6;
- значения параметров:
 - $\alpha = 0.04$ – параметр, определяющий влияние феромонов;
 - $\beta = 1.2$ – параметр, определяющий влияние эвристической информации;
 - $\rho = 0.05$ – коэффициент испарения феромонов.
- начальная концентрация феромонов: $\tau_{ij} = 1$ для всех дуг.

Таблица 4.1 – Матрица расстояний

№ дуги \ № дуги	1	2	3	4	5	6
1	—	4	7	8	3	9
2	4	—	5	6	4	5
3	7	5	—	1	7	6
4	8	6	1	—	2	3
5	3	4	7	2	—	8
6	9	5	6	3	8	—

Возьмем начальный маршрут: $1 \rightarrow 6 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$, длина данного маршрута: $L = 43$.

Взвешенная эвристическая информация для каждой дуги вычисляется с учетом параметра β по формуле:

$$\tau_{ij}^{\alpha} * \eta_{ij}^{\beta} = \tau_{ij}^{\alpha} * \left(\frac{1}{d_{ij}}\right)^{\beta}, \quad (4.5)$$

1. Переход из вершины 1 в вершину 2:

- вычисление взвешенной эвристической информации для всех дуг в соответствии с формулой 4.5:

$$\circ \tau_{12}^{\alpha} * \eta_{12}^{\beta} = 1^{0.04} * \left(\frac{1}{4}\right)^{1.2} \approx 0.189;$$

$$\circ \tau_{13}^{\alpha} * \eta_{13}^{\beta} = 1^{0.04} * \left(\frac{1}{7}\right)^{1.2} \approx 0.096;$$

$$\circ \tau_{14}^{\alpha} * \eta_{14}^{\beta} = 1^{0.04} * \left(\frac{1}{8}\right)^{1.2} \approx 0.082;$$

$$\circ \tau_{15}^{\alpha} * \eta_{15}^{\beta} = 1^{0.04} * \left(\frac{1}{3}\right)^{1.2} \approx 0.267;$$

$$\circ \tau_{16}^{\alpha} * \eta_{16}^{\beta} = 1^{0.04} * \left(\frac{1}{9}\right)^{1.2} \approx 0.071.$$

- расчет суммы всех взвешенных эвристик:

$$\sum_{k \in N_1} \tau_{1k}^{\alpha} * \eta_{1k}^{\beta} = 0.189 + 0.096 + 0.082 + 0.267 + 0.071 = 0.705.$$

- вычисление вероятности перехода в соответствии с формулой 4.1:

$$P_{12} = \frac{0.189}{0.705} \approx 0.268.$$

- длина маршрута после перехода:

$$L_1 = 0 + 4 = 4.$$

- обновление феромонов в соответствии с формулой 4.4:

$$\tau_{12}(t+1) = (1 - 0.05) * 1 + 0.25 = 1.2.$$

2. Переход из вершины 2 в вершину 5:

- вычисление взвешенной эвристической информации для всех дуг в соответствии с формулой 4.5:

$$\circ \tau_{21}^{\alpha} * \eta_{21}^{\beta} = 1^{0.04} * \left(\frac{1}{4}\right)^{1.2} \approx 0.189;$$

$$\circ \tau_{23}^{\alpha} * \eta_{23}^{\beta} = 1^{0.04} * \left(\frac{1}{5}\right)^{1.2} \approx 0.145;$$

$$\circ \tau_{24}^{\alpha} * \eta_{24}^{\beta} = 1^{0.04} * \left(\frac{1}{6}\right)^{1.2} \approx 0.116;$$

$$\circ \tau_{25}^{\alpha} * \eta_{25}^{\beta} = 1^{0.04} * \left(\frac{1}{4}\right)^{1.2} \approx 0.189;$$

$$\circ \tau_{26}^{\alpha} * \eta_{26}^{\beta} = 1^{0.04} * \left(\frac{1}{5}\right)^{1.2} \approx 0.145.$$

- расчет суммы всех взвешенных эвристик:

$$\sum_{k \in N_2} \tau_{2k}^{\alpha} * \eta_{2k}^{\beta} = 0.189 + 0.145 + 0.116 + 0.189 + 0.145 = 0.784.$$

- вычисление вероятности перехода в соответствии с формулой 4.1:

$$P_{25} = \frac{0.189}{0.784} \approx 0.241.$$

- длина маршрута после перехода:

$$L_2 = 4 + 4 = 8.$$

- обновление феромонов в соответствии с формулой 4.4:

$$\tau_{25}(t+1) = (1 - 0.05) * 1 + 0.125 = 1.075.$$

3. Переход из вершины 5 в вершину 4:

- вычисление взвешенной эвристической информации для всех дуг в соответствии с формулой 4.5:

$$\circ \tau_{51}^{\alpha} * \eta_{51}^{\beta} = 1^{0.04} * \left(\frac{1}{3}\right)^{1.2} \approx 0.268;$$

$$\circ \tau_{52}^{\alpha} * \eta_{52}^{\beta} = 1^{0.04} * \left(\frac{1}{4}\right)^{1.2} \approx 0.189;$$

$$\circ \tau_{53}^{\alpha} * \eta_{53}^{\beta} = 1^{0.04} * \left(\frac{1}{7}\right)^{1.2} \approx 0.096;$$

$$\circ \tau_{54}^{\alpha} * \eta_{54}^{\beta} = 1^{0.04} * \left(\frac{1}{2}\right)^{1.2} \approx 0.435;$$

$$\circ \tau_{56}^{\alpha} * \eta_{56}^{\beta} = 1^{0.04} * \left(\frac{1}{8}\right)^{1.2} \approx 0.082.$$

- расчет суммы всех взвешенных эвристик:

$$\sum_{k \in N_5} \tau_{5k}^{\alpha} * \eta_{5k}^{\beta} = 0.268 + 0.189 + 0.096 + 0.435 + 0.082 = 1.07.$$

- вычисление вероятности перехода в соответствии с формулой 4.1:

$$P_{54} = \frac{0.435}{1.07} \approx 0.407.$$

- длина маршрута после перехода:

$$L_3 = 8 + 2 = 10.$$

- обновление феромонов в соответствии с формулой 4.4:

$$\tau_{54}(t+1) = (1 - 0.05) * 1 + 0.1 = 1.05.$$

4. Переход из вершины 4 в вершину 6:

- вычисление взвешенной эвристической информации для всех дуг в соответствии с формулой 4.5:

$$\circ \tau_{41}^{\alpha} * \eta_{41}^{\beta} = 1^{0.04} * \left(\frac{1}{8}\right)^{1.2} \approx 0.082;$$

$$\circ \tau_{42}^{\alpha} * \eta_{42}^{\beta} = 1^{0.04} * \left(\frac{1}{6}\right)^{1.2} \approx 0.116;$$

$$\circ \tau_{43}^{\alpha} * \eta_{43}^{\beta} = 1^{0.04} * \left(\frac{1}{1}\right)^{1.2} = 1;$$

$$\circ \tau_{45}^{\alpha} * \eta_{45}^{\beta} = 1^{0.04} * \left(\frac{1}{2}\right)^{1.2} \approx 0.435;$$

$$\circ \tau_{46}^{\alpha} * \eta_{46}^{\beta} = 1^{0.04} * \left(\frac{1}{3}\right)^{1.2} \approx 0.268.$$

- расчет суммы всех взвешенных эвристик:

$$\sum_{k \in N_4} \tau_{4k}^{\alpha} * \eta_{4k}^{\beta} = 0.082 + 0.116 + 1 + 0.435 + 0.268 = 1.901.$$

- вычисление вероятности перехода в соответствии с формулой 4.1:

$$P_{46} = \frac{0.268}{1.901} \approx 0.141.$$

- длина маршрута после перехода:

$$L_4 = 10 + 3 = 13.$$

- обновление феромонов в соответствии с формулой 4.4:

$$\tau_{46}(t + 1) = (1 - 0.05) * 1 + 0.08 = 1.03.$$

5. Переход из вершины 6 в вершину 3:

- вычисление взвешенной эвристической информации для всех дуг в соответствии с формулой 4.5:

$$\circ \tau_{61}^{\alpha} * \eta_{61}^{\beta} = 1^{0.04} * \left(\frac{1}{9}\right)^{1.2} \approx 0.072;$$

$$\circ \tau_{62}^{\alpha} * \eta_{62}^{\beta} = 1^{0.04} * \left(\frac{1}{5}\right)^{1.2} \approx 0.145;$$

$$\circ \tau_{63}^{\alpha} * \eta_{63}^{\beta} = 1^{0.04} * \left(\frac{1}{6}\right)^{1.2} \approx 0.116;$$

$$\circ \tau_{64}^{\alpha} * \eta_{64}^{\beta} = 1^{0.04} * \left(\frac{1}{3}\right)^{1.2} \approx 0.268;$$

$$\circ \tau_{65}^{\alpha} * \eta_{65}^{\beta} = 1^{0.04} * \left(\frac{1}{8}\right)^{1.2} \approx 0.082.$$

- расчет суммы всех взвешенных эвристик:

$$\sum_{k \in N_6} \tau_{6k}^{\alpha} * \eta_{6k}^{\beta} = 0.072 + 0.145 + 0.116 + 0.268 + 0.082 = 0.683.$$

- вычисление вероятности перехода в соответствии с формулой 4.1:

$$P_{63} = \frac{0.116}{0.683} \approx 0.17.$$

- длина маршрута после перехода:

$$L_5 = 13 + 6 = 19.$$

- обновление феромонов в соответствии с формулой 4.4:

$$\tau_{63}(t + 1) = (1 - 0.05) * 1 + 0.05 = 1.$$

6. Переход из вершины 3 в вершину 1:

- вычисление взвешенной эвристической информации для всех дуг в соответствии с формулой 4.5:

$$\circ \tau_{31}^{\alpha} * \eta_{31}^{\beta} = 1^{0.04} * \left(\frac{1}{7}\right)^{1.2} \approx 0.096;$$

$$\circ \tau_{32}^{\alpha} * \eta_{32}^{\beta} = 1^{0.04} * \left(\frac{1}{5}\right)^{1.2} \approx 0.145;$$

$$\circ \tau_{34}^{\alpha} * \eta_{34}^{\beta} = 1^{0.04} * \left(\frac{1}{1}\right)^{1.2} = 1;$$

$$\circ \tau_{35}^{\alpha} * \eta_{35}^{\beta} = 1^{0.04} * \left(\frac{1}{7}\right)^{1.2} \approx 0.096;$$

$$\circ \tau_{36}^{\alpha} * \eta_{36}^{\beta} = 1^{0.04} * \left(\frac{1}{6}\right)^{1.2} \approx 0.116.$$

- расчет суммы всех взвешенных эвристик:

$$\sum_{k \in N_3} \tau_{3k}^{\alpha} * \eta_{3k}^{\beta} = 0.096 + 0.145 + 1 + 0.096 + 0.116 = 1.453.$$

- вычисление вероятности перехода в соответствии с формулой 4.1:

$$P_{31} = \frac{0.096}{1.453} \approx 0.066.$$

- длина маршрута после перехода:

$$L_6 = 19 + 7 = 26.$$

- обновление феромонов в соответствии с формулой 4.4:

$$\tau_{31}(t+1) = (1 - 0.05) * 1 + 0.04 = 0.99.$$

Полученный маршрут для первого муравья: $1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 6 \rightarrow 3 \rightarrow 1$, изображенный на рисунке 4.2.

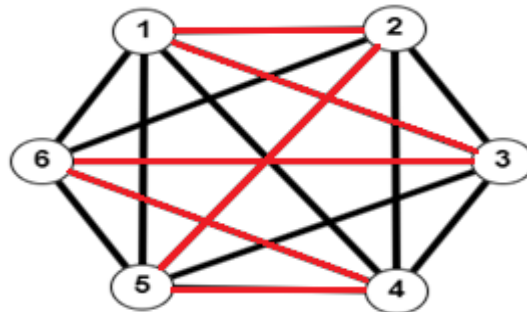


Рисунок 4.2 – Выбранный маршрут

После расчета первой итерации, была определена общая длина маршрута для первого муравья: $L = 26$. Значение лучшей длины маршрута было обновлено, так как новая длина оказалась меньше предыдущей.

4.4 Программная реализация муравьиного алгоритма

Выполним программную реализацию муравьиного алгоритма на языке GoLang (Приложение Д). Результаты работы программы представлены на

рисунке 4.3. В результате работы программы необходимо найти оптимальный путь в графе, а также длину этого пути. Запуск программы позволяет найти оптимальный маршрут с минимальной длиной, что подтверждает эффективность муравьиного алгоритма в решении задачи поиска кратчайшего пути. Запустим написанную программу и убедимся в правильности её работы:

```
Муравей 2: Путь = [0 4 3 5 2 1 0], Длина = 23.00
Муравей 3: Путь = [0 4 3 5 2 1 0], Длина = 23.00
Муравей 4: Путь = [0 4 3 5 2 1 0], Длина = 23.00
Муравей 5: Путь = [0 4 3 5 2 1 0], Длина = 23.00
Муравей 6: Путь = [0 2 1 5 3 4 0], Длина = 25.00

Итерация 998:
Муравей 1: Путь = [0 1 5 2 4 3 0], Длина = 32.00
Муравей 2: Путь = [0 4 2 1 3 5 0], Длина = 33.00
Муравей 3: Путь = [0 1 2 5 4 3 0], Длина = 33.00
Муравей 4: Путь = [0 4 1 2 5 3 0], Длина = 29.00
Муравей 5: Путь = [0 1 2 5 3 4 0], Длина = 23.00
Муравей 6: Путь = [0 4 3 5 2 1 0], Длина = 23.00

Итерация 999:
Муравей 1: Путь = [0 1 2 5 4 3 0], Длина = 33.00
Муравей 2: Путь = [0 4 3 5 2 1 0], Длина = 23.00
Муравей 3: Путь = [0 1 3 4 2 5 0], Длина = 34.00
Муравей 4: Путь = [0 5 3 4 2 1 0], Длина = 30.00
Муравей 5: Путь = [0 5 2 1 3 4 0], Длина = 31.00
Муравей 6: Путь = [0 4 3 5 2 1 0], Длина = 23.00

Итерация 1000:
Муравей 1: Путь = [0 1 2 3 4 5 0], Длина = 29.00
Муравей 2: Путь = [0 1 2 5 3 4 0], Длина = 23.00
Муравей 3: Путь = [0 1 2 5 3 4 0], Длина = 23.00
Муравей 4: Путь = [0 1 2 5 3 4 0], Длина = 23.00
Муравей 5: Путь = [0 1 4 3 5 2 0], Длина = 26.00
Муравей 6: Путь = [0 4 3 5 2 1 0], Длина = 23.00

Лучший путь: [0 4 3 2 5 1 0]
Лучшая длина: 21.00
```

Рисунок 4.3 – Результат работы программы муравьиного алгоритма

5 ПЧЕЛИНЫЙ АЛГОРИТМ

Пчелиный алгоритм (Bee Algorithm) – это эволюционный алгоритм оптимизации, основанный на моделировании поведения пчел в природе при поиске источников пищи. Он черпает вдохновение из того, как пчелы координируют усилия для нахождения лучших участков с пищей и оптимизируют свои поиски, взаимодействуя друг с другом и распределяя ресурсы. В этом алгоритме используется популяция пчел, каждая из которых представляет собой потенциальное решение задачи оптимизации. Позиция пчелы в пространстве поиска соответствует точке, где вычисляется значение целевой функции, и это значение определяет «приспособленность» или качество решения.

Цель алгоритма – найти оптимальное решение, минимизируя (или максимизируя) значение целевой функции.

Процесс поиска состоит из чередования этапов локального и глобального поиска. На этапе глобального поиска разведчики исследуют пространство, а лучшие найденные области становятся фокусом локального поиска, где происходит распределение рабочих пчел. Алгоритм постепенно концентрирует усилия на тех участках пространства, которые показывают лучшие значения функции, чтобы достичь глобального минимума (или максимума). Это сочетание локального и глобального поиска помогает избежать застревания в локальных минимумах, так как пчелы-разведчики постоянно исследуют новые области пространства.

Алгоритм пчел эффективно применим для поиска глобального минимума в задачах оптимизации благодаря его способности сосредотачиваться на лучших участках и одновременно искать новые потенциальные решения.

Такой подход делает пчелиный алгоритм полезным инструментом для решения задач в различных областях, где требуется найти глобальный минимум функции.

5.1 Описание пчелиного алгоритма

Пчелиный алгоритм представляет собой метод поиска экстремумов, вдохновленный поведением пчел при поиске источников нектара. В этом алгоритме каждое потенциальное решение представлено в виде пчелы, которая «знает» свое текущее местоположение в пространстве поиска и хранит параметры многомерной функции.

Этапы работы алгоритма

1. Инициализация и размещение пчел-разведчиков: На первом этапе задается количество пчел-разведчиков S , которые случайным образом распределяются в пространстве поиска D . Каждая пчела-разведчик $X_{\beta,0}$ имеет свои случайные координаты, где β – номер пчелы, а 0 обозначает номер текущей итерации. Для каждой из этих точек вычисляется значение целевой функции $F(X)$, которое будет использоваться для определения перспективности областей.

2. Выделение лучших и перспективных участков: На основе полученных значений целевой функции выделяются два типа участков:

- Лучшие участки: n областей с наибольшими значениями целевой функции. Они представляют наиболее перспективные зоны для дальнейшего поиска.
- Перспективные участки: m областей, значения функции в которых близки к наибольшим. Эти участки также заслуживают внимания, так как могут содержать локальные максимумы.

Каждый участок d_{β} представляет собой область локального поиска в виде гиперкуба размерности R^k с центром в точке $X_{\beta,0}$. Длина сторон гиперкуба равна 2Δ , где Δ – параметр, определяющий размер локальной области поиска.

3. Проверка расстояния между агентами и объединение участков: Вычисляется Евклидово расстояние между парами пчел-разведчиков – $\|X_{\beta,0} - X_{\gamma,0}\|$. Для точек $A = (x_1, x_2, \dots, x_n)$ и $B = (y_1, y_2, \dots, y_n)$ евклидово расстояние считается в соответствии с формулой 5.1.

$$d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \quad (5.1)$$

Если расстояние между двумя пчелами меньше заданного порогового значения, то применяется один из следующих подходов:

- назначение двух пересекающихся участков для каждого агента (лучших и/или перспективных);
- назначение одного общего участка, центр которого расположен в точке пчелы с наибольшим значением целевой функции. В данной реализации используется второй вариант, при котором объединяется только один участок.

4. Обработка пересекающихся участков: При выполнении алгоритма проверяется, находятся ли два агента (пчелы) на расстоянии друг от друга, меньшем, чем заданный порог. Для этого вычисляется Евклидово расстояние между точками агентов. Если расстояние меньше порога, считается, что участки пересекаются, и предпринимается объединение участков.

Для объединения участков используется следующий подход: создается один общий участок, центр которого находится в средней точке между двумя пересекающимися агентами. Эта точка рассчитывается по формуле 5.2:

$$x_{new} = \frac{x_1 + x_2}{2}, y_{new} = \frac{y_1 + y_2}{2}, \quad (5.2)$$

где (x_1, y_1) и (x_2, y_2) – координаты агентов. После вычисления новой точки оба агента перемещаются в эту позицию. Значение функции в новой точке вычисляется и используется для определения качества объединенного участка.

5. Отправка дополнительных агентов на выбранные участки: В каждый из выделенных участков направляются дополнительные агенты для более тщательного поиска. На лучшие участки отправляется по N агентов, а на перспективные – по M агентов. Координаты новых агентов определяются случайным образом в пределах заданных участков.

6. Вычисление целевой функции и обновление центров участков: В новых точках вычисляется значение целевой функции $F(X)$. Среди всех полученных значений выбирается точка с наибольшим значением функции, и она становится новым центром соответствующей области. Это позволяет алгоритму постепенно приближаться к глобальному максимуму.

7. Повторение процесса до достижения критерия остановки: Шаги 4 и 5 повторяются, пока не будет достигнут искомый результат или пока значения функции и координаты максимумов не перестанут изменяться в течение нескольких итераций (параметр остановки τ). Алгоритм завершает свою работу, если значения функции стабилизировались или было достигнуто максимальное количество итераций.

5.2 Постановка задачи и цели

Целью данной работы реализовать нахождение глобального минимума функции, определенной в ограниченном пространстве поиска. Для этого используется пчелиный алгоритм, который моделирует поведение пчел при поиске оптимальных источников нектара и позволяет эффективно исследовать как локальные, так и глобальные экстремумы.

Целевая функция для оптимизации – это функция трёхгорбого верблюда, заданная следующим выражением:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2,$$

глобальный минимум:

$$f(1, 1 \dots) = 0,$$

метод поиска:

$$-\infty \leq x_i \leq \infty,$$

$$1 \leq i \leq n$$

5.3 Ручной расчет пчелиного алгоритма

1. Параметры алгоритма:

- количество пчел-разведчиков (S): 3;
- количество пчел, отправляемых на лучшие участки (N): 5;
- количество пчел, отправляемых на перспективные участки (M): 3;
- количество лучших участков (n): 1;
- количество перспективных участков (m): 1;
- пороговое значение для Евклидова расстояния: 0.7;
- размер области для каждого участка (Δ): 0.5.

2. Начальные позиции пчел и значения функции:

Пусть пчелы-разведчики оказались в следующих точках пространства:

$$A_1: f(1, 1) = 0.$$

$$A_2: f(-2, 3) = 109.$$

$$A_3: f(4, -2) = 32409.$$

Согласно, установленным параметрам, выбирается лучшие точки:

$$f(1, 1) = 0.$$

Затем определяется перспективный участок:

$$f(-2, 3) = 109$$

3. Вычисление Евклидова расстояния между пчелами при помощи формулы 5.1:

Расстояние между A_1 и A_2 :

$$d_{12} = \sqrt{(1 - (-2))^2 + (1 - 3)^2} = \sqrt{13} \approx 3.6056.$$

Так как $3.6056 > 0.7$, участки не пересекаются.

Расстояние между A_1 и A_3 :

$$d_{13} = \sqrt{(1 - 4)^2 + (1 - (-2))^2} = \sqrt{18} \approx 4.2426.$$

Так как $4.2426 > 0.7$, участки не пересекаются.

Расстояние между A_2 и A_3 :

$$d_{23} = \sqrt{(-2 - 4)^2 + (3 - (-2))^2} = \sqrt{61} \approx 7.8102.$$

Так как $7.8102 > 0.7$, участки не пересекаются.

В окрестности лучших точек будут отправлены по 2 пчелы:

Для лучшей точки значение координат, которыми ограничивается участок будет:

$x \in [1 - 0.5 = 0.5; 1 + 0.5 = 1.5]$ – для первой координаты.

$y \in [1 - 0.5 = 0.5; 1 + 0.5 = 1.5]$ – для второй координаты.

Аналогично рассчитываются интервалы для перспективных участков.

В лучший интервал отправляем по 2 пчелы, а на перспективные участки по 1 пчеле. Причем, мы не будем менять положение пчел, нашедших лучшие и перспективные участки, иначе есть вероятность того, что на следующей итерации максимальное значение целевой функции будет хуже, чем на предыдущем шаге.

Теперь пусть на первом лучшем участке имеются следующие пчелы:

$$f(0.5, 0.5) = (1 - 0.5)^2 + 100(0.5 - 0.5^2)^2 = 6.5.$$

$$f(1.5, 1) = 156.5.$$

$$f(1.2, 1.1) = 11.6.$$

$$f(0.8, 0.9) = 6.8.$$

$$f(1, 1) = 0.$$

Как видно, из расчета, уже среди этих новых точек есть точки, которые лучше, чем предыдущее решение. Следовательно, данное улучшение фиксируется, и соответствующие координаты обновляются.

Также поступаем с перспективным участком. После чего среди всех новых точек снова отмечаются лучшая и перспективная, а процесс повторяется заново. Если значение функции в перспективной точке лучше, чем предыдущее, ее координаты также обновляются.

5.4 Программная реализация пчелиного алгоритма

Выполним программную реализацию пчелиного алгоритма на языке GoLang (Приложение Е). Результаты работы программы представлены на рисунках 5.1, 5.2. В результате работы программы необходимо выполнить поиск глобального экстремума целевой функции, значения которой зависят от параметров x и y в пределах заданного диапазона. Пчелиный алгоритм моделирует поведение пчел, распределяющих свои усилия между разведкой новых участков и локальным поиском в наиболее перспективных зонах. На каждом шаге алгоритма пчелы-разведчики случайным образом распределяются в пространстве, после чего на основе значений целевой функции определяются лучшие и перспективные участки. В лучшие и перспективные зоны отправляются дополнительные агенты для более глубокого локального поиска, а если участки пересекаются, используется объединение, что помогает избежать дублирования.

В результате, значение функции постепенно улучшается, а координаты пчел сходятся к области, содержащей глобальный минимум. Итоговые результаты включают как числовые значения для найденных экстремумов, так и визуализацию процесса сходимости, что позволяет оценить распределение агентов в пространстве поиска на разных этапах работы алгоритма.

```
Итерация 995: Лучшее решение: X = 1.1305, Y = 1.2387, F(X,Y) = 0.1715
Итерация 996: Лучшее решение: X = 1.0742, Y = 1.1302, F(X,Y) = 0.0618
Итерация 997: Лучшее решение: X = 0.8702, Y = 0.7742, F(X,Y) = 0.0460
Итерация 998: Лучшее решение: X = 1.1762, Y = 1.3790, F(X,Y) = 0.0331
Итерация 999: Лучшее решение: X = 1.0034, Y = 1.0208, F(X,Y) = 0.0198
Итерация 1000: Лучшее решение: X = 1.2383, Y = 1.5004, F(X,Y) = 0.1649

Финальное решение: X = 1.0032, Y = 1.0097, F(X,Y) = 0.0011
```

Рисунок 5.1 – Результат работы программы пчелиного алгоритма

6 ОТЧЕТ ПО ХАКАТОНУ: БОТ-ПОМОШНИК ДЛЯ УСКОРЕНИЯ АДАПТАЦИИ ПОЛЬЗОВАТЕЛЕЙ К ПО

6.1 Постановка цели и задачи

Современные пользователи программного обеспечения (ПО) часто сталкиваются с проблемой адаптации к новым инструментам. Основные трудности включают:

- сложность изучения новой среды;
- необходимость поиска ответов в обширной документации;
- недостаток времени на изучение функционала.

Цель проекта: разработать инновационное решение в виде чат-бота для ускорения адаптации пользователей к ПО.

Основные задачи:

1. Автоматизация поиска ответов в документации.
2. Обеспечение интуитивно понятного интерфейса для взаимодействия с пользователем.
3. Интеграция с AI для генерации релевантных и персонализированных ответов.

6.2 Архитектура системы

Технологический стек:

- Backend: язык Go с использованием фреймворка Gin для обработки HTTP-запросов.
- AI: интеграция с YandexGPT для генерации ответов.
- Frontend: React для создания пользовательского интерфейса.
- Хранилище: сохранение документации и истории запросов.

Ключевые элементы архитектуры:

1. API обработка:
 - Реализация CORS для взаимодействия с фронтендом.
 - Обработка запросов пользователя через REST API.
2. Обработка документов:
 - Импорт и анализ файлов форматов .docx и JSON для извлечения релевантной информации.
3. AI интеграция:
 - Генерация текстовых ответов на вопросы пользователя с помощью YandexGPT.
4. Frontend:
 - Простой и интуитивно понятный интерфейс, включающий поле для ввода запросов, область для отображения ответов, и историю запросов.

6.3 Описание функциональности

Процесс работы бота:

1. Пользователь вводит вопрос в текстовое поле.
2. Бэкэнд обрабатывает запрос:
 - ищет ответ в загруженной документации;
 - отправляет запрос в YandexGPT для получения более точного ответа.
3. Сформированный ответ отображается в интерфейсе пользователя.
4. Если требуется, бот предоставляет изображения и дополнительные материалы из документации.

Преимущества решения:

- Ускорение адаптации к ПО за счёт мгновенных ответов.
- Снижение нагрузки на службу поддержки.
- Доступность сервиса 24/7.

- Постоянное обучение и улучшение модели на основе пользовательских запросов.

6.4 Результат работы

1. Тестирование UX:

- Бот показал высокую точность в предоставлении релевантных ответов.
- Простота интерфейса позволила пользователям без опыта быстро освоить работу с ПО.

2. Скорость:

- Среднее время отклика составило менее 1 секунды.

3. Продуктивность:

- Уменьшение времени, необходимого на обучение пользователей, на 30%.

4. Демонстрация:

- Визуальные примеры интерфейса, работы бота и типичных сценариев взаимодействия.

6.5 Заключение

Проект успешно реализован в рамках хакатона, предоставив инструмент, способный:

- улучшить пользовательский опыт за счёт быстрого доступа к информации;
- снизить затраты на поддержку пользователей;
- повысить продуктивность благодаря автоматизации задач обучения.

Дальнейшие шаги:

1. Расширение функционала, включая поддержку новых форматов документации.

2. Улучшение моделей машинного обучения для повышения точности ответов.

3. Проведение дополнительных тестирований на реальных пользователях.

Проект доказал свою эффективность и готов к дальнейшему развитию.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы исследованы и реализованы различные алгоритмы для решения задач оптимизации в рамках системного анализа данных. Рассмотренные алгоритмы, такие как алгоритм отжига, основной роевой, муравьиный и пчелиный, продемонстрировали свою эффективность в условиях многокритериальности и ограничений, характерных для систем поддержки принятия решений.

Алгоритм отжига позволил находить решения в сложных задачах оптимизации, где важна способность алгоритма избегать локальных экстремумов и достигать глобально оптимальных результатов. Муравьиный алгоритм показал свою полезность в задачах поиска путей и маршрутов, имитируя коллективное поведение для нахождения наилучших решений. Пчелиный алгоритм, ориентированный на многокритериальную оптимизацию, показал хорошие результаты при решении задач, требующих учета различных критериев одновременно. Алгоритм имеет сходства с градиентным спуском, а также показал наилучший результат при работе с целевой функцией.

В процессе работы также разработаны подходы к программной реализации этих алгоритмов, что позволило не только глубже понять их структуру, но и оценить их практическую применимость для решения реальных задач. Анализ эффективности и ограничений каждого алгоритма показал, что выбор подхода должен быть основан на специфике задачи и требований к точности и скорости принятия решений.

Таким образом, результаты работы подтверждают, что использование алгоритмов оптимизации в системах поддержки принятия решений способствует повышению качества и обоснованности выбора решений в условиях неопределенности и множества факторов. Выполненные исследования и программные реализации могут служить основой для дальнейшего совершенствования систем поддержки принятия решений и адаптации их под конкретные прикладные задачи.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Программа для реализации онтологий – protégé – URL: <https://protege.stanford.edu/>. Дата обращения – 10.09.2024.
2. Пантелеев, А. В. Методы глобальной оптимизации. Метаэвристические стратегии и алгоритмы / А.В. Пантелеев, Д.В. Метлицкая, Е.А. Алешина / Вузовская книга. 2020, 244 с.
3. Шматов, Г. П. Нейронные сети и генетический алгоритм: учебное пособие / Г. П. Шматов / Лань: электронно-библиотечная система 2019
4. Тестовые функции для оптимизации – URL: https://ru.wikipedia.org/wiki/Тестовые_функции_для_оптимизации. Дата обращения – 01.10.2024.
5. Иванов, А. В. Применение алгоритма пчелиного роя для оптимизации задач распределения ресурсов / А.В. Иванов, Б.С. Петров / Вестник Московского университета 2020, с. 45-58.
6. Петрова, М.В. Гибридизация алгоритма пчелиного роя с генетическими алгоритмами для оптимизации сложных систем / М.В. Петрова, К.С. Лебедев / Журнал системного анализа и математического моделирования 2023, с. 34-47.
7. Лукина, О.П. Алгоритм пчелиного роя в задачах планирования и управления / О.П. Лукина, Ю.И. Васильев / Известия высших учебных заведений 2022, с. 301-315.
8. Соколов, Н.В. Применение алгоритма пчелиного роя для оптимизации нейронных сетей / Н.В. Соколов, Т.Е. Иванова / Вестник Санкт-Петербургского университета 2021, с. 134-150.
9. Наседкин, И.В. Нахождение кратчайшего пути с помощью эвристических алгоритмов / И.В. Наседкин, И.Н. Репьев, М.М. Бычковский, Н.Н. Зайкин, Е.В. Фатьянова, Д.А. Нестеров / Известия ТузГУ. Технические науки 2023.

10. Дюlicheва, Ю.Ю. Алгоритмы роевого интеллекта и их применение для анализа образовательных данных. / Ю.Ю. Дюlicheва / Открытое образование. 2019, с. 33-43.
11. Документация к YandexGPT – URL: <https://yandex.cloud/ru/docs>
12. Документации к библиотекам GoLang – URL: <https://pkg.go.dev/>

ПРИЛОЖЕНИЯ

Приложение А – Код реализации онтологии на языке GoLang.

Приложение Б – Код реализации алгоритма отжига коммивояжера на языке GoLang.

Приложение В – Код реализации алгоритма отжига Коши на языке GoLang.

Приложение Г – Код реализации основного роевого алгоритма на языке GoLang.

Приложение Д – Код реализации муравьиного алгоритма на языке GoLang.

Приложение Е – Код реализации пчелиного алгоритма на языке GoLang.

Приложение Ж – Ссылка на репозиторий с кодом проекта для хакатона.

Приложение И – Сертификат участника в окружном хакатоне «Цифровой прорыв. Сезон: Искусственный интеллект»

Приложение А

Код реализации онтологии на языке GoLang.

Листинг А – Код реализации онтологии

```
package main

import "fmt"

// Класс "Курсы"
type Course struct {
    Title      string
    Description string
    Languages   []Program_Languages // Привязка: курс содержит программные
языки
    Format      []string // очный, заочный
    Difficulty  string
    Duration    string
    Lesson_plan []Lesson
    Teaches     []Teacher
}

type Program_Languages struct {
    Title      string
    Scope_of_application string // область применения
    Type       string
}

// Класс "Занятия"
type Lesson struct {
    Title      string
    Materials  string
    Practical  []Practical_exercises
    Test       []Test
}

type Practical_exercises struct {
    Number      int
    Conditions  string
    Evaluation   int //оценка
    Requirements []string //требования
}

type Test struct {
    Question      string
    Answer_options []string //варианты ответы
    Evaluation     int
    Right_answer  string // правильный ответ
}

// Класс "Учащиеся"
type Student struct {
    Name      string
    Email     string
    Takes_courses []Course // Привязка: учащийся зарегистрирован на
несколько курсов
}

// Класс "Преподаватели"
type Teacher struct {
```

Продолжение листинга А

```
Name    string
Status  string
Area    []string //область экспертизы
}

func main() {
    plang1 := Program_Languages{
        Title:      "GoLang",
        Scope_of_application: "микросервисы",
        Type:        "скомпилированный",
    }
    test1 := Test{
        Question: "Как вывести значение переменной a = 101? Go",
        Answer_options: []string{
            "fmt.Println(a)",
            "fmt.Println('a')",
            "print(a)",
            "fPrint(a)",
        },
        Evaluation: 2,
        Right_answer: "fmt.Println(a)",
    }
    pr1 := Practical_exercises{
        Number: 1,
        Conditions: "Вывести в консоль числа от 1 до 10",
        Evaluation: 1,
        Requirements: []string{
            "Использовать цикл for",
            "Использовать только 1 цикл",
        },
    }
    pr2 := Practical_exercises{
        Number: 2,
        Conditions: "Вывести в консоль числа от 1 до 10, потом вывести его
в одну строчку в обратном порядке",
        Evaluation: 3,
        Requirements: []string{
            "Использовать цикл for",
            "Использовать только 1 цикл",
            "Использовать sort",
        },
    }

    lesson1 := Lesson{
        Title:      "Введение",
        Materials: "Лекция",
        Practical: []Practical_exercises{pr1},
        Test:      []Test{test1},
    }

    lesson2 := Lesson{
        Title:      "Основы GO",
        Materials: "Лекция",
        Practical: []Practical_exercises{pr1, pr2},
        Test: []Test{ // добавим тест
            {
                Question: "Как объявить переменную в Go?",
                Answer_options: []string{
                    "var a int",
                    "int a",
                    "a := int",
                    "let a = int",
                },
            }
        },
    }
}
```

```

        },
        Evaluation: 2,
        Right_answer: "var a int",
    },
},
}

teacher1 := Teacher{
    "Геннадий",
    "активный",
    []string{"golang"},
}

course1 := Course{
    Title: "GoLang база",
    Description: "Обучение основам языка программирования GoLang",
    Languages: []Program_Languages{plang1},
    Format: []string{"очно", "заочно"},
    Difficulty: "лёгкая",
    Duration: "1 месяц",
    Lesson_plan: []Lesson{lesson1, lesson2},
    Teaches: []Teacher{teacher1},
}

student1 := Student{
    "Антон",
    "aaaaa@dd.dd",
    []Course{course1},
}

student2 := Student{
    Name: "Толик",
    Email: "ttttt@dd.dd",
    Takes_courses: []Course{course1},
}

courses := []Course{course1}
students := []Student{student1, student2}
pracs := []Practical_exercises{pr1, pr2}
lessons := []Lesson{lesson1, lesson2}

fmt.Println("Проподователь, который ведёт курс 'GoLang база' (Относиться к классу Курсы):")
for _, i := range courses[0].Teaches {
    fmt.Printf("%s (Относиться к классу Преподователь)\n", i.Name)
    fmt.Print("Этот курс проходят:")
    for _, j := range students {
        for _, c := range j.Takes_courses {
            if c.Title == "GoLang база" {
                fmt.Printf("\n%s - Относиться к классу
Учащиеся", j.Name)
            }
        }
    }
}

fmt.Println()
fmt.Println("Задание, которое оценивается больше чем в 2 балла: ")
for _, i := range pracs {
    if i.Evaluation > 2 {
        fmt.Printf(" - Номер задания: %v (Относиться к классу
Задания) \n", i.Number)
        for _, les := range lessons {
            for _, prac := range les.Practical {
                if prac.Number == i.Number {
                    fmt.Printf("Задание %v присутствует в
занятиях '%s' (Класс Занятия)", i.Number, les.Title)

```

Продолжение листинга А

```

                                for _, c := range courses {
                                    for _, l := range c.Lesson_plan {
                                        if l.Title == les.Title {
                                            fmt.Printf("\n
Занятия %s входят в план занятий '%s' (Класс Курсы)", l.Title, c.Title)
                                                }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Приложение Б

Код реализации алгоритма отжига коммивояжера на языке GoLang.

Листинг Б – Код реализации алгоритма отжига коммивояжера

```
package main

import (
    "fmt"
    "math"
    "math/rand"
    "time"
)

// Координаты городов
type City struct {
    name string
    x, y float64
}

// Расстояние между двумя городами
func distance(a, b City) float64 {
    return math.Sqrt(math.Pow(a.x-b.x, 2) + math.Pow(a.y-b.y, 2))
}

// Подсчет общего расстояния по маршруту
func totalDistance(route []City) float64 {
    total := 0.0
    for i := 0; i < len(route)-1; i++ {
        total += distance(route[i], route[i+1])
    }
    total += distance(route[len(route)-1], route[0]) // замыкаем маршрут
    return total
}

// Перемешивание для получения нового маршрута
func swap(route []City) []City {
    newRoute := make([]City, len(route))
    copy(newRoute, route)

    i, j := rand.Intn(len(route)), rand.Intn(len(route))
    for i == j {
        j = rand.Intn(len(route))
    }

    newRoute[i], newRoute[j] = newRoute[j], newRoute[i]
    return newRoute
}

// Имитация отжига
func simulatedAnnealing(cities []City, initialTemp, coolingRate float64,
    maxIter int) []City {
    rand.Seed(time.Now().UnixNano())
    currentRoute := cities
    bestRoute := cities
    currentTemp := initialTemp

    for i := 0; i < maxIter; i++ {
        newRoute := swap(currentRoute)
        currentDistance := totalDistance(currentRoute)
```


Продолжение листинга Б

```
        newDistance := totalDistance(newRoute)

        // Проверка, принимаем ли новый маршрут
        if newDistance < currentDistance || math.Exp((currentDistance-
newDistance)/currentTemp) > rand.Float64() {
            currentRoute = newRoute
            if newDistance < totalDistance(bestRoute) {
                bestRoute = newRoute
            }
        }

        currentTemp *= coolingRate

        fmt.Printf("Итерация %d: Текущее расстояние %f, Лучшее расстояние
%f, Температура %v\n", i+1, currentDistance, totalDistance(bestRoute),
currentTemp)
    }

    return bestRoute
}

func main() {
    // Список городов с координатами (широта, долгота)
    cities := []City{
        {"Лондон", 51.5074, -0.1278},
        {"Париж", 48.8566, 2.3522},
        {"Рим", 41.9028, 12.4964},
        {"Нью-Йорк", 40.7128, -74.0060},
        {"Токио", 35.6895, 139.6917},
        {"Москва", 55.7558, 37.6173},
        {"Пекин", 39.9042, 116.4074},
        {"Сидней", -33.8688, 151.2093},
    }

    initialTemp := 100.0
    coolingRate := 0.99
    maxIter := 100

    bestRoute := simulatedAnnealing(cities, initialTemp, coolingRate,
maxIter)
    fmt.Println("Лучший найденный маршрут:")
    for _, city := range bestRoute {
        fmt.Printf("%s (%.2f, %.2f)\n", city.name, city.x, city.y)
    }
    fmt.Printf("Общее расстояние: %f\n", totalDistance(bestRoute))
}
```

Приложение В

Код реализации алгоритма отжига Коши на языке GoLang.

Листинг В – Код реализации алгоритма отжига Коши

```
package main

import (
    "fmt"
    "math"
    "math/rand"
    "time"
)

// Функция Розенброка для оценки стабильности полета дрона
func rosenbrock(theta, a float64) float64 {
    return math.Pow((a-1), 2) + 100*math.Pow((theta-a*a), 2)
}

// Генерация случайного числа с распределением Коши
func cauchyRandom() float64 {
    return math.Tan(math.Pi * (rand.Float64() - 0.5))
}

// Алгоритм имитации отжига по Коши
func simulatedAnnealingCauchy(maxIterations int, initialTemp float64,
    coolingRate float64) (float64, float64, float64) {
    // Инициализация начальных параметров (угол наклона theta и ускорение a)
    theta := rand.Float64() * 10
    a := rand.Float64() * 10
    currentEnergy := rosenbrock(theta, a)

    for i := 0; i < maxIterations; i++ {
        // Вычисление новой температуры
        temperature := initialTemp / float64(i+1)

        // Генерация новых значений theta и a с помощью распределения Коши
        newTheta := theta + cauchyRandom()
        newA := a + cauchyRandom()

        // Вычисление энергии для нового состояния
        newEnergy := rosenbrock(newTheta, newA)

        // Принятие нового состояния с определённой вероятностью
        if newEnergy < currentEnergy || math.Exp((currentEnergy-
            newEnergy)/temperature) > rand.Float64() {
            theta = newTheta
            a = newA
            currentEnergy = newEnergy
        }

        // Снижение температуры
        initialTemp *= coolingRate
    }

    return theta, a, currentEnergy
}

func main() {
    rand.Seed(time.Now().UnixNano())
```

Продолжение листинга В

```
// Настройки алгоритма отжига
maxIterations := 10000
initialTemp := 100.0
coolingRate := 0.99

// Запуск алгоритма
optimalTheta, optimalA, minEnergy :=
simulatedAnnealingCauchy(maxIterations, initialTemp, coolingRate)

fmt.Printf("Оптимальные параметры для стабилизации полета дрона:\n")
fmt.Printf("Угол наклона (theta): %.4f\n", optimalTheta)
fmt.Printf("Ускорение (a): %.4f\n", optimalA)
fmt.Printf("Минимальная энергия (функция Розенброка): %.4f\n",
minEnergy)
}
```

Приложение Г

Код реализации основного роевого алгоритма на языке GoLang.

Листинг Г – Код реализации основного роевого алгоритма

```
package main

import (
    "fmt"
    "math"
    "math/rand"
    "time"
)

const (
    numParticles    = 30 // Количество частиц в рое
    numDimensions   = 2  // Размерность задачи (функция Розенброка)
    maxIterations   = 100 // Максимальное количество итераций
    inertiaWeight   = 0.7 // Вес инерции
    cognitiveCoeff  = 1.5 // Коэффициент когнитивной составляющей
    socialCoeff     = 1.5 // Коэффициент социальной составляющей
)

// Функция Розенброка
func rosenbrock(x []float64) float64 {
    sum := 0.0
    for i := 0; i < len(x)-1; i++ {
        sum += 100*math.Pow(x[i+1]-x[i]*x[i], 2) + math.Pow(1-x[i], 2)
    }
    return sum
}

// Частица
type Particle struct {
    position []float64 // Текущее положение
    velocity []float64 // Текущая скорость
    bestPos  []float64 // Лучшая позиция частицы
    bestFitness float64 // Лучшее значение функции для частицы
}

// Создание новой частицы
func newParticle(dimensions int) Particle {
    position := make([]float64, dimensions)
    velocity := make([]float64, dimensions)
    bestPos := make([]float64, dimensions)

    for i := 0; i < dimensions; i++ {
        position[i] = rand.Float64()*10 - 5 // Генерация начального
        // положения в пределах [-5, 5]
        velocity[i] = rand.Float64()*2 - 1 // Генерация начальной
        // скорости в пределах [-1, 1]
        bestPos[i] = position[i]
    }

    bestFitness := rosenbrock(position)

    return Particle{
        position: position,
        velocity: velocity,
        bestPos: bestPos,
    }
}
```

```

        bestFitness: bestFitness,
    }
}

func main() {
    rand.Seed(time.Now().UnixNano())

    // Инициализация частиц
    particles := make([]Particle, numParticles)
    for i := range particles {
        particles[i] = newParticle(numDimensions)
    }

    // Глобальные лучшие позиция и значение функции
    globalBestPos := make([]float64, numDimensions)
    globalBestFitness := math.Inf(1)

    // Поиск глобального минимума
    for iteration := 0; iteration < maxIterations; iteration++ {
        for i := range particles {
            fitness := rosenbrock(particles[i].position)

            // Обновление личного лучшего
            if fitness < particles[i].bestFitness {
                particles[i].bestFitness = fitness
                copy(particles[i].bestPos, particles[i].position)
            }

            // Обновление глобального лучшего
            if fitness < globalBestFitness {
                globalBestFitness = fitness
                copy(globalBestPos, particles[i].position)
            }
        }
        // Обновление скоростей и положений
        for i := range particles {
            for d := 0; d < numDimensions; d++ {
                r1 := rand.Float64()
                r2 := rand.Float64()

                cognitive := cognitiveCoeff * r1 *
(particles[i].bestPos[d] - particles[i].position[d])
                social := socialCoeff * r2 * (globalBestPos[d] -
particles[i].position[d])

                particles[i].velocity[d] =
inertiaWeight*particles[i].velocity[d] + cognitive + social
                particles[i].position[d] += particles[i].velocity[d]
            }
        }

        // Вывод результатов текущей итерации на русском
        fmt.Printf("Итерация %d: Лучшее значение функции = %.6f в позиции
%v\n",
            iteration, globalBestFitness, globalBestPos)
    }

    // Итоговый вывод на русском
    fmt.Println("Оптимизация завершена")
    fmt.Printf("Лучшая позиция: %v\n", globalBestPos)
    fmt.Printf("Лучшее значение функции: %.6f\n", globalBestFitness)
}

```

Приложение Д

Код реализации муравьиного алгоритма на языке GoLang.

Листинг Д – Код реализации муравьиного алгоритма

```
package main

import (
    "fmt"
    "math"
    "math/rand"
    "time"
)

// Параметры алгоритма
const (
    alpha = 2.0
    beta  = 0.00001
    rho   = 0.000005
)

// Матрица расстояний
var distances = [][]float64{
    {math.Inf(1), 4, 7, 8, 3, 9},
    {4, math.Inf(1), 5, 6, 4, 5},
    {7, 5, math.Inf(1), 1, 7, 6},
    {8, 6, 1, math.Inf(1), 2, 3},
    {3, 4, 7, 2, math.Inf(1), 8},
    {9, 5, 6, 3, 8, math.Inf(1)},
}

var (
    numVertices = len(distances)
    pheromones   = make([][]float64, numVertices)
    eta          = make([][]float64, numVertices)
)

func init() {
    rand.Seed(time.Now().UnixNano())

    // Инициализация матрицы феромонов и эвристической информации
    for i := range pheromones {
        pheromones[i] = make([]float64, numVertices)
        eta[i] = make([]float64, numVertices)
        for j := range pheromones[i] {
            pheromones[i][j] = 1.0
            if distances[i][j] != math.Inf(1) {
                eta[i][j] = 1 / distances[i][j]
            } else {
                eta[i][j] = 0
            }
        }
    }
}

// Расчет вероятностей перехода
func calculateProbabilities(currentVertex int, unvisited []int) []float64 {
    numerator := make([]float64, len(unvisited))
    denominator := 0.0
```

Продолжение листинга Д

```
        for i, v := range unvisited {
            numerator[i] = math.Pow(pheromones[currentVertex][v], alpha) *
math.Pow(eta[currentVertex][v], beta)
            denominator += numerator[i]
        }

        probabilities := make([]float64, len(unvisited))
        for i := range probabilities {
            probabilities[i] = numerator[i] / denominator
        }
        return probabilities
    }

// Выбор следующей вершины на основе вероятностей
func chooseNextVertex(probabilities []float64, unvisited []int) int {
    r := rand.Float64()
    cumulative := 0.0
    for i, p := range probabilities {
        cumulative += p
        if r <= cumulative {
            return unvisited[i]
        }
    }
    return unvisited[len(unvisited)-1]
}

// Маршрут муравья
func antTravel(startVertex int) ([]int, float64) {
    currentVertex := startVertex
    unvisited := make([]int, 0, numVertices-1)
    for i := 0; i < numVertices; i++ {
        if i != currentVertex {
            unvisited = append(unvisited, i)
        }
    }

    path := []int{currentVertex}
    totalLength := 0.0

    for len(unvisited) > 0 {
        probabilities := calculateProbabilities(currentVertex, unvisited)
        nextVertex := chooseNextVertex(probabilities, unvisited)

        totalLength += distances[currentVertex][nextVertex]
        path = append(path, nextVertex)
        currentVertex = nextVertex

        // Удаляем посещенную вершину из списка
        for i, v := range unvisited {
            if v == currentVertex {
                unvisited = append(unvisited[:i], unvisited[i+1:]...)
                break
            }
        }
    }

    // Возврат к стартовой вершине
    path = append(path, startVertex)
    totalLength += distances[currentVertex][startVertex]

    return path, totalLength
}
```

Продолжение листинга Д

```
// Обновление феромонов
func updatePheromones(paths [][]int, lengths []float64) {
    for i := range pheromones {
        for j := range pheromones[i] {
            pheromones[i][j] *= (1 - rho)
        }
    }

    for k, path := range paths {
        deltaTau := 1 / lengths[k]
        for i := 0; i < len(path)-1; i++ {
            from, to := path[i], path[i+1]
            pheromones[from][to] += deltaTau
            pheromones[to][from] += deltaTau
        }
    }
}

// Основная функция муравьиного алгоритма
func antColonyOptimization(numAnts, numIterations, startVertex int) ([]int,
float64) {
    var bestPath []int
    bestLength := math.Inf(1)

    for iteration := 0; iteration < numIterations; iteration++ {
        paths := make([][]int, 0, numAnts)
        lengths := make([]float64, 0, numAnts)

        fmt.Printf("\nИтерация %d:\n", iteration+1)

        for ant := 0; ant < numAnts; ant++ {
            path, length := antTravel(startVertex)
            paths = append(paths, path)
            lengths = append(lengths, length)

            fmt.Printf("  Муравей %d: Путь = %v, Длина = %.2f\n", ant+1,
path, length)
        }

        updatePheromones(paths, lengths)

        // Обновление глобального лучшего пути
        for i, length := range lengths {
            if length < bestLength {
                bestLength = length
                bestPath = paths[i]
            }
        }
    }

    return bestPath, bestLength
}

func main() {
    numAnts := 6
    numIterations := 1000
    startVertex := 0

    bestPath, bestLength := antColonyOptimization(numAnts, numIterations,
startVertex)
    fmt.Printf("\nЛучший путь: %v\nЛучшая длина: %.2f\n", bestPath,
bestLength)
}
```


Приложение Е

Код реализации пчелиного алгоритма на языке GoLang.

Листинг Е – Код реализации пчелиного алгоритма

```
package main

import (
    "fmt"
    "math"
    "math/rand"
    "time"
)

// Параметры алгоритма
const (
    a, b          = 1.0, 100.0 // параметры функции Розенброка
    S             = 30         // количество пчел-разведчиков
    n             = 8          // количество лучших участков
    m             = 10         // количество перспективных участков
    N             = 15         // пчелы в лучших участках
    M             = 10         // пчелы в перспективных участках
    Delta         = 0.5        // размер локальной области поиска
    StopIterations = 1000      // критерий останова
    SearchMin     = -2.0       // минимальная граница поиска
    SearchMax     = 2.0        // максимальная граница поиска
    DistanceThresh = 0.2       // порог для объединения участков
)

// Пчела
type Bee struct {
    X, Y float64 // координаты
    F     float64 // значение функции
}

// Функция Розенброка
func rosenbrock(x, y float64) float64 {
    return math.Pow(a-x, 2) + b*math.Pow(y-x*x, 2)
}

// Генерация случайной точки
func randomPoint() (float64, float64) {
    return SearchMin + rand.Float64()*(SearchMax-SearchMin),
        SearchMin + rand.Float64()*(SearchMax-SearchMin)
}

// Оценка пчелы
func evaluateBee(bee *Bee) {
    bee.F = rosenbrock(bee.X, bee.Y)
}

// Генерация разведчиков
func generateScoutBees() []Bee {
    bees := make([]Bee, S)
    for i := 0; i < S; i++ {
        bees[i].X, bees[i].Y = randomPoint()
        evaluateBee(&bees[i])
    }
    return bees
}
```

Продолжение листинга E

```
// Сортировка пчел по значению функции
func sortBees(bees []Bee) {
    for i := 0; i < len(bees)-1; i++ {
        for j := 0; j < len(bees)-i-1; j++ {
            if bees[j].F > bees[j+1].F {
                bees[j], bees[j+1] = bees[j+1], bees[j]
            }
        }
    }
}

// Вычисление Евклидова расстояния
func euclideanDistance(b1, b2 Bee) float64 {
    return math.Sqrt(math.Pow(b1.X-b2.X, 2) + math.Pow(b1.Y-b2.Y, 2))
}

// Удаление близких точек
func mergeCloseAreas(bees []Bee, threshold float64) []Bee {
    merged := []Bee{bees[0]} // Начинаем с первой точки
    for i := 1; i < len(bees); i++ {
        tooClose := false
        for _, b := range merged {
            if euclideanDistance(bees[i], b) < threshold {
                tooClose = true
                break
            }
        }
        if !tooClose {
            merged = append(merged, bees[i])
        }
    }
    return merged
}

// Поиск в окрестности
func searchNeighborhood(center Bee, numBees int, delta float64) []Bee {
    neighborhood := make([]Bee, numBees)
    for i := 0; i < numBees; i++ {
        neighborhood[i].X = center.X + (rand.Float64()*2-1)*delta
        neighborhood[i].Y = center.Y + (rand.Float64()*2-1)*delta
        evaluateBee(&neighborhood[i])
    }
    return neighborhood
}

// Основной алгоритм
func beeAlgorithm() Bee {
    rand.Seed(time.Now().UnixNano())
    scoutBees := generateScoutBees()

    for iteration := 0; iteration < StopIterations; iteration++ {
        // Сортируем разведчиков по значению функции
        sortBees(scoutBees)

        // Удаляем слишком близкие участки
        scoutBees = mergeCloseAreas(scoutBees, DistanceThresh)

        // Вывод текущего состояния
        bestBee := scoutBees[0]
        fmt.Printf("Итерация %d: Лучшее решение: X = %.4f, Y = %.4f, F(X,Y) = %.4f\n",
            iteration+1, bestBee.X, bestBee.Y, bestBee.F)
    }
}
```

Продолжение листинга E

```
// Лучшие и перспективные участки
bestBees := scoutBees[:min(len(scoutBees), n)]
promisingBees := scoutBees[n:min(len(scoutBees), n+m)]

// Локальный поиск
newBees := []Bee{}
for _, bee := range bestBees {
    newBees = append(newBees, searchNeighborhood(bee, N,
Delta)...)
}
for _, bee := range promisingBees {
    newBees = append(newBees, searchNeighborhood(bee, M,
Delta)...)
}

// Обновляем популяцию
scoutBees = append(newBees, scoutBees[n+m:]...)
sortBees(scoutBees)
}

// Лучшее решение
return scoutBees[0]
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}

func main() {
    bestSolution := beeAlgorithm()
    fmt.Printf("\nФинальное решение: X = %.4f, Y = %.4f, F(X,Y) = %.4f\n",
        bestSolution.X, bestSolution.Y, bestSolution.F)
}
```

Приложение Ж

Ссылка на репозиторий с кодом проекта для хакатона.

https://github.com/Lamadj0/assistant_bot/tree/main

Приложение И

Приложение И – Сертификат участника окружного хакатона «Цифровой прорыв. Сезон: Искусственный интеллект»

