Министерство науки и высшего образования РФ

федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Прикладная математика и фундаментальная информатика»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **КУРСОВОЙ ПРОЕКТ** | | | | |
|  | | | | |
| по дисциплине | Проектная деятельность | | | |
|  |  |  |  |  |
| на тему | Решение задачи коммивояжера алгоритмом муравьиной колонии | | | |
|  |  |  |  |  |
| **Пояснительная записка** | | | | |
|  | | | | |
| шифр проекта | 056-КП-02.03.03-20-ПЗ | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студента | Лещёв Дмитррий Евгеньевич | | | |
|  | фамилия, имя, отчество полностью | | | |
| Курс | 2 | Группа | ФИТ-221 | |
| Направление | 02.03.02 Фундаментальная информатика и информационные технологии | | | |
|  | код, наименование | | | |
| Руководитель | канд.физ.-мат. наук | | | |
|  | ученая степень, звание | | | |
| Девятерикова М.В | | | |
|  | фамилия, инициалы | | | |
| Выполнил |  | | | |
|  | дата, подпись студента | | | |
| К защите |  | | | |
|  | дата, подпись руководителя | | | |
| Выполнение и  подготовка к защите КП | | Защита КП | | Итоговый рейтинг |
|  | |  | |  |
| Проект защищен с оценкой | | | | |

Омск 2024

**СПИСОК ИСПОЛНИТЕЛЕЙ**

Ответственный исполнитель

(лидер команды) 20.05.2024 Чепурко А.И.

подпись, дата

Курс 2 Группа ФИТ-221

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Исполнитель 20.05.2024 Сагалбаев Д.А.

подпись, дата

Курс 2 Группа ФИТ-221

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Исполнитель 20.05.2024 Зензин Е.Н.

подпись, дата

Курс 2 Группа МО-221

Направление 02.03.02 Математическое обеспечение и администрирование информационных систем

Исполнитель 20.05.2024 Чуакбаев Е.А.

подпись, дата

Курс 2 Группа ФИТ-221

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Исполнитель 20.05.2024 Лещёв Д.Е.

подпись, дата

**РЕФЕРАТ**

Пояснительная записка 29 c., 8 рис., 3 источника.

ОПТИМИЗАЦИЯ, ЗАДАЧА КОММИВОЯЖЕРА, АЛГОРИТМ МУРАВЬИНОЙ КОЛОНИИ

Объект проекта –алгоритм муравьиной колонии.

Цель проекта – решение задачи о коммивояжера с помощью алгоритма муравьиной колонии.

В процессе работы над проектом наша команда изучила методическую литературу, прочла различные статьи.

В результате выполнения проекта была разработана реализация алгоритма муравьиной колонии для задачи коммивояжера.

Содержание

[ВВЕДЕНИЕ 3](#_Toc168244150)

[1 Постановка задачи 4](#_Toc168244151)

[1.1 Общие сведения 4](#_Toc168244152)

[1.2 Практические приложения 5](#_Toc168244153)

[2 Изучение теоретических основ и программного инструментария для разработки программного решения 7](#_Toc168244154)

[2.1 Постановки задачи о коммивояжера 7](#_Toc168244155)

[2.2 NP - полнота задачи 8](#_Toc168244156)

[2.3 Алгоритм муравьиного колонии 9](#_Toc168244157)

[2.4 Инструментарий для проектного решения 10](#_Toc168244158)

[3 Разработка решения 11](#_Toc168244159)

[3.1 Разработка прототипа/макета 11](#_Toc168244160)

[3.2 Версионный контроль 13](#_Toc168244161)

[3.3 Демонстрация решения 14](#_Toc168244162)

[ЗАКЛЮЧЕНИЕ 20](#_Toc168244163)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc168244164)

**ВВЕДЕНИЕ**

**Задача коммивояжера** (TSP) и её различные модификации широко применяются на практике в прикладной математике, логистике, экономике и управлении для нахождения оптимальных маршрутов доставки, планирования и распределения ресурсов. Эта задача заключается в нахождении кратчайшего пути, проходящего через заданное количество городов и возвращающегося в исходный пункт.

**Актуальность темы** исследования обусловлена тем, что задача коммивояжера и её различные модификации имеют широкое применение в различных областях, таких как транспортировка, управление запасами, планирование маршрутов и распределение ресурсов. Решение этой задачи позволяет минимизировать затраты на транспортировку, время доставки и повысить общую эффективность логистических процессов.

**Область применения** алгоритма муравьиной колонии включает различные области, такие как оптимизация маршрутов доставки, планирование задач в проектах и распределение ресурсов. Этот метод может быть использован для нахождения оптимальных или близких к оптимальным решений в сложных комбинаторных задачах, что позволяет значительно сократить затраты или увеличить прибыль в различных сценариях. Таким образом, исследование алгоритма муравьиной колонии для решения задачи коммивояжера имеет большое значение для практического применения в различных отраслях экономики и управления.

**Цель проекта** – разработка и исследование метода решения задачи коммивояжера с использованием алгоритма муравьиной колонии, а также оценка его эффективности по сравнению с другими методами оптимизации.

Для достижения цели необходимо выполнить следующие **задачи**:

1. Изучить постановку задачи коммивояжера и её особенности;
2. Разработать алгоритм муравьиной колонии для задачи коммивояжера;
3. Провести экспериментальное исследование реализации алгоритма;
4. Провести анализ алгоритма, выявив его слабые и сильные стороны.

**1 Постановка задачи**

**1.1 Общие сведения**

Проектная идея заключается в применении метода муравьиной колонии для решения задачи коммивояжера. Основная цель проекта — исследовать и реализовать алгоритм муравьиной колонии (в англоязычных статьях также встречаются названия Ant Colony Optimization, ACO), который эффективно находит приближённые решения для задачи коммивояжера.

Необходимые шаги для исследования, реализации и применения метода муравьиной колонии:

1. Формализация задачи коммивояжера: определить параметры задачи, такие как количество городов и расстояния между ними;
2. Разработка алгоритма муравьиной колонии: создание и настройка основных компонентов алгоритма, включая моделирование поведения муравьёв и механизм обновления феромонов;
3. Имплементация алгоритма: написание кода для алгоритма муравьиной колонии на выбранном языке программирования;
4. Тестирование и валидация: проведение экспериментов для оценки производительности алгоритма на различных наборах данных. Сравнение полученных результатов с известными методами оптимизации;
5. Анализ и интерпретация результатов: анализ полученных результатов, включая оценку качества решений, время выполнения и устойчивость алгоритма к изменению параметров.

Проблема заключается в сложности создания эффективного алгоритма, который способен находить хорошие приближённые решения для задачи коммивояжера за разумное время. Это включает в себя настройку параметров алгоритма, таких как количество муравьёв, скорость испарения феромонов, и выбор подходящей стратегии обновления решений.

**1.2 Практические приложения**

Задача коммивояжера находит широкое применение в различных областях благодаря своей универсальности и релевантности к множеству реальных проблем. Вот несколько ключевых областей, где применяется задача коммивояжера:

**Логистика и транспортировка:**

1. **Маршрутизация транспортных средств**: Оптимизация маршрутов для грузовых автомобилей или контейнеров с целью минимизации затрат на топливо и времени доставки, при этом учитывая ограничения по времени и дорожным условиям;
2. **Управление парком транспортных средств**: Эффективное распределение и планирование использования автопарка для выполнения различных задач, таких как доставка товаров или обслуживание клиентов.

**Финансовый сектор:**

1. **Маршруты инкассаторов**: Оптимизация маршрутов для инкассации денежных средств из различных точек для минимизации времени и повышения безопасности;
2. **Аудиторские проверки**: Планирование маршрутов для проведения аудиторских проверок, чтобы минимизировать затраты на перемещения между объектами.

**Производство и управление ресурсами:**

1. **Техническое обслуживание**: Оптимизация маршрутов для технического обслуживания оборудования на производственных площадках, чтобы минимизировать время простоя и затраты на обслуживание;
2. **Управление запасами**: Оптимизация маршрутов для перемещения материалов и продукции на складах и в производственных цепочках.

**Информационные технологии:**

1. **Маршрутизация данных в сетях**: Оптимизация путей передачи данных в компьютерных сетях для максимальной пропускной способности и минимизации задержек;
2. **Облачные вычисления**: Оптимизация распределения задач и маршрутов передачи данных в облачных системах для эффективного использования вычислительных ресурсов.

**Проектное управление:**

1. Инспекционные маршруты: Оптимизация маршрутов для инспекторов, проводящих проверки на различных объектах, чтобы минимизировать время на перемещения и повысить эффективность работы;
2. Планирование проектов: Оптимизация графиков выполнения задач и маршрутов для работников, участвующих в проектах, чтобы достичь максимальной эффективности при ограниченных ресурсах.

**Медицина и здравоохранение:**

1. **Мобильные медицинские бригады**: Оптимизация маршрутов для медицинских бригад, оказывающих помощь на дому, чтобы минимизировать время на перемещения и повысить качество обслуживания пациентов;
2. **Анализ геномных данных**: Оптимизация последовательностей обработки геномных данных для минимизации времени анализа и повышения точности результатов.

**Электронная коммерция:**

1. **Маршруты курьеров**: Оптимизация маршрутов доставки заказов клиентам, чтобы минимизировать время доставки и повысить удовлетворенность клиентов;
2. **Управление складом**: Оптимизация маршрутов для сбора товаров на складе для выполнения заказов, что позволяет минимизировать время на сборку и повысить производительность.

Эти примеры демонстрируют, насколько широко и разнообразно может быть применена задача коммивояжера. В каждом случае основная идея заключается в оптимизации маршрутов при ограниченных ресурсах, что делает задачу коммивояжера универсальным инструментом для решения множества реальных проблем.

**2 Изучение теоретических основ и программного инструментария для разработки программного решения**

**2.1** **Постановки задачи о коммивояжера**

Задача коммивояжера – одна из задач комбинаторной оптимизации. Классическая задача коммивояжера известна очень давно. Вот её постановка: имеется набор из 𝑁 городов, необходимо найти кратчайший маршрут, который проходит через все эти города ровно один раз и возвращается в исходный город.

Формально, задача коммивояжера может быть представлена следующим образом:

1. Пусть задано множество городов ,
2. Для каждой пары городов определено расстояние .

Требуется найти такой порядок посещения городов, чтобы минимизировать общий путь, пройденный коммивояжером.

Минимизировать

Где – расстояние между городами *i* и *i +* 1, а – расстояние от последнего города к первому.

Проблема коммивояжера является NP-трудной, что означает, что для неё не существует известного полиномиального времени решения. Тем не менее, существуют множество эвристических и метаэвристических алгоритмов, которые могут найти приближенные решения за разумное время. Одним из таких методов является алгоритм муравьиной колонии.

Вариации задачи коммивояжера:

1. Симметричная задача коммивояжера: расстояния между городами одинаковы в обоих направлениях, то есть ;
2. Асимметричная задача коммивояжера: расстояния между городами могут отличаться в зависимости от направления, то есть ;
3. Многомерная задача коммивояжера: включает дополнительные параметры, такие как временные окна для посещения городов или ограничение на ресурсы.

**2.2 NP - полнота задачи**

Большинство используемых алгоритмов имеют полиномиальное время работы, то есть, если размер входных данных – n, то время выполнения алгоритма в худшем случае оценивается как , где k – константа. Однако существуют задачи, которые не могут быть решены за полиномиальное время. Эти задачи относятся к классу NP-полных. Некоторые задачи этого класса могут показаться похожими на те, что решаются за полиномиальное время, но это не так. Задача считается NP-полной, если для неё не существует полиномиального алгоритма. Алгоритм называется полиномиальным, если его сложность в худшем случае ограничена сверху многочленом от n. NP-полные задачи часто возникают в различных областях: в булевой логике, теории графов, теории множеств, кодировании информации, алгебре, биологии, физике, экономике, теории автоматов и языков. Считается, что NP-полные задачи очень трудноразрешимы, и если для одной из них удастся найти полиномиальный алгоритм, такой алгоритм будет существовать для любой задачи из этого класса. Несмотря на усилия многих ученых, до сих пор не найден полиномиальный алгоритм для любой из NP-полных задач. Из этого следует, что при известной NP-полноте задачи лучше сосредоточиться на разработке приближенного алгоритма, чем пытаться найти полиномиальный, или же, если это возможно, использовать алгоритмы с экспоненциальной сложностью работы.

**2.3 Алгоритм муравьиного колонии**

Алгоритм муравьиной колонии (Ant Colony Optimization, ACO) — это метаэвристический метод, который используется для решения комбинаторных оптимизационных задач. Он вдохновлен поведением реальных муравьев в природе, которые используют феромоны для поиска кратчайших путей к источникам пищи. В ACO искусственные муравьи взаимодействуют друг с другом через феромонные следы, чтобы найти хорошие решения для задачи оптимизации.

**Основные этапы алгоритма**

1. Инициализация:

* Параметры алгоритма: коэффициенты влияния феромонов (α) и расстояния (β), коэффициент испарения феромонов (ρ), количество феромона, выделяемого муравьем (Q), количество элитных муравьев (e), и другие параметры;
* Матрицы: матрица расстояний между городами (L) и матрица феромонов (τ).

1. Запуск муравьев:

* Начальная позиция: муравьи могут начинать из случайных городов или из одного фиксированного города;
* Проход по маршруту: каждый муравей строит путь, последовательно выбирая следующий город на основе вероятности, которая зависит от феромона и расстояния до города. Вероятность выбора города *j* для муравья, находящегося в городе *i*, определяется формулой:

,

Где - количество феромона на пути от года i к городу j, –

обратное расстояние между городами i и j.

1. Обновление феромонов:

* Испарение феромонов: на каждом шаге все феромоны частично испаряются:
* Обновление феромонов: каждый муравей откладывает феромоны на пройденных маршрутах:

, где – длина маршрута муравья.

1. Выбор лучшего решения:

* В каждом поколении алгоритм сохраняет лучший найденный маршрут. Если этот маршрут лучше глобального лучшего, то он обновляет глобальный лучший маршрут.

**2.4 Инструментарий для проектного решения**

1. *GitHub Desktop* — это набор инструментов, который предоставляет интуитивно понятный пользовательский интерфейс (UI) для управления репозиториями Git и совместной работы над ними. Он создан для упрощения процесса работы с контролем версий, что позволяет разработчикам сосредоточиться на написании кода.

**Основные возможности *GitHub Desktop*:**

* Простое клонирование, создание и управление репозиториями без необходимости использовать командную строку;
* Удобная навигация по репозиториям, просмотр истории коммитов и управление ветвями через визуальный интерфейс;
* Возможность получать обновления, публиковать изменения и объединять ветки несколькими щелчками мыши;
* Удобные инструменты для разрешения конфликтов слияния с помощью визуального сравнения и слияния.

2. *Visual Studio Code* (VS Code) – это бесплатный и легковесный редактор кода, разработанный Microsoft. Он предоставляет широкий спектр возможностей для разработки приложений на различных языках программирования.

**Основные возможности и функции Visual Studio Code:**

* VS Code основан на архитектуре с поддержкой расширений и плагинов, что позволяет изменять и настраивать редактор под свои нужды;
* Сообщество разработчиков создало множество расширений, добавляющих дополнительные функции и возможности;
* Возможность отладки кода непосредственно в редакторе;
* Установка точек останова, выполнение шагов отладки и анализ значений переменных, что значительно упрощает процесс исправления ошибок.

**3 Разработка решения**

**3.1 Разработка прототипа/макета**

Класс AntAlgorithm принимает в качестве входных значений следующие параметры:

1. a – коэффициент запаха;
2. b - коэффициент расстояния;
3. rho - коэффициент высыхания;
4. Q - количество выпускаемого феромона;
5. e - количество элитных муравьев;
6. local\_refresh - локальное обновление феромона;
7. where\_to\_start - место начала движения муравьев;
8. as\_modif - модификация обновления феромона.

В классе реализованы следующие функции:

* + fit;
  + get\_best\_route;
  + get\_best\_dist;
  + get\_best\_dists;
  + get\_average\_dists.

Функция **fit**, которая принимает следующие параметры: L (Матрица расстояний между городами), AGES(Количество поколений (итераций) алгоритма), ANTS(Количество муравьев в каждом поколении), ph (Начальное значение феромона).

**Основной цикл по поколениям** (for age in range(self.AGES)):

**Инициализация маршрутов и расстояний муравьев**:

* antROUTE: Маршруты муравьев.
* antDIST: Длины маршрутов муравьев.

**Цикл по муравьям** (for k in range(self.ANTS)):

* **Начальное расположение муравья** в зависимости от self.where\_to\_start.
* **Построение маршрута**: Для каждого города вычисляется вероятность посещения следующего города на основе текущего уровня феромона и обратного расстояния.
* **Локальное обновление феромона**: Обновление феромона на каждом шаге в зависимости от модификации системы муравьев (self.as\_modif).
* **Вычисление длины маршрута**: После посещения всех городов вычисляется длина маршрута.
* **Сравнение и обновление лучшего маршрута**: Если найден новый лучший маршрут, он сохраняется.

**Обновление феромонов**:

* **Высыхание феромонов**: Снижение уровня феромонов на всех маршрутах.
* **Добавление феромонов на пройденные маршруты**: В зависимости от модификации системы муравьев (self.as\_modif).
* **Элитные муравьи**: Дополнительное усиление феромонов на лучших маршрутах элитными муравьями.

**Возврат:** Функция возвращает обновленную матрицу феромонов.

Функция **get\_best\_route** возвращает лучший найденный маршрут, хранящийся в атрибуте self.BEST\_ROUTE.

Функция **get\_best\_dist** возвращает длину лучшего найденного маршрута, хранящуюся в атрибуте self.BEST\_DIST.

Функция **get\_best\_dists** возвращает:

1. Список номеров поколений;
2. Вектор длины лучших маршрутов для каждого поколения;
3. self.antBEST\_DIST: Массив, содержащий длины лучших маршрутов для каждого поколения (итерации) алгоритма.

**3.2 Версионный контроль**

Совместная работа над проектом осуществлялась при помощи системы контроля версий *Git*. Каждый из участников команды осуществлял разработку, предварительно скопировав созданный удаленный репозиторий на локальный компьютер. Изменения командой *gitpush* отправлялись на отдельные ветки, после чего объединялись с данными общей ветки *main*.

**3.3 Демонстрация решения**

Был разработан алгоритм для решения задачи коммивояжера, код находится в приложении A. Для проверки реализации были проведены тесты алгоритма. Входные данные представлены на рисунке.

1. *a ­–* коэффициент запаха;
2. *b –* коэффициент расстояния;
3. *rho –* коэффициент высыхания;
4. *Q –* количество выпускаемого феромона;
5. *e -* количество элитных муравьев;
6. *L –* Матрица расстояний между городами.

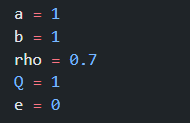


Рисунок 1 – Входные данные

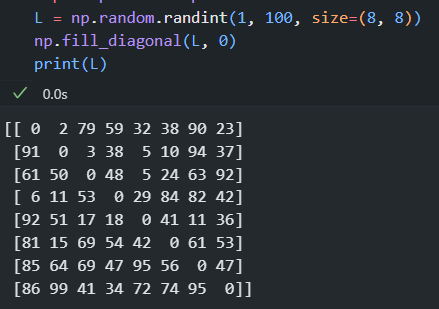


Рисунок 2 - Входная матрица

На рисунке 3 изображен результат работы программы – самый короткий путь получится таковым: 3 -> 5 -> 7 -> 8 -> 6 -> 4 -> 1 -> 2 -> 3 длиною 170.

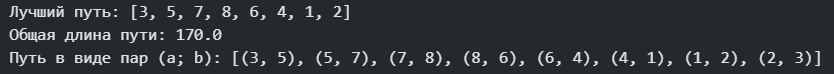


Рисунок 3 – Результат работы программы

Выше был описан пример работоспособности нашей программы. Теперь проведем эксперименты с разными размерностями матрицы и с разными параметрами нашего алгоритма и посмотрим, как изменится конечный результат работы нашей программы. В эксперименте были использованы следующие параметры:

1. A = [1, 2] - коэффициент запаха;
2. B = [1, 2] - коэффициент расстояния;
3. E = [2, 5, 10] - количество элитных муравьев;
4. RHO = [0.5, 0.7, 0.9] - коэффициент высыхания;
5. L = 8x8 - матрица расстояний между городами.

В процессе перебора параметров были получены следующие результаты, изображенные на рисунках 4 и 5:

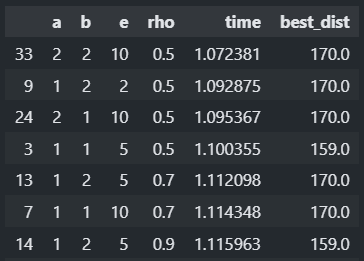


Рисунок 4 - Результат перебора параметров



Рисунок 5 – Результат перебора параметров

В результате, после эксперимента мы получили, что лучшие параметры для данной задачи являются таковыми:

1. Коэффициент запаха (a) = 1;
2. Коэффициент расстояния (b) = 1 ;
3. Количество элитных муравьев (e) = 5;
4. Коэффициент высыхания (rho) = 0.5.

Самый короткий путь равен 159. В новом эксперименте у нас останется такой перебор параметров, но изменится размер матрицы расстояний (L = 3x3).

Результат перебора представлен на рисунке 6:

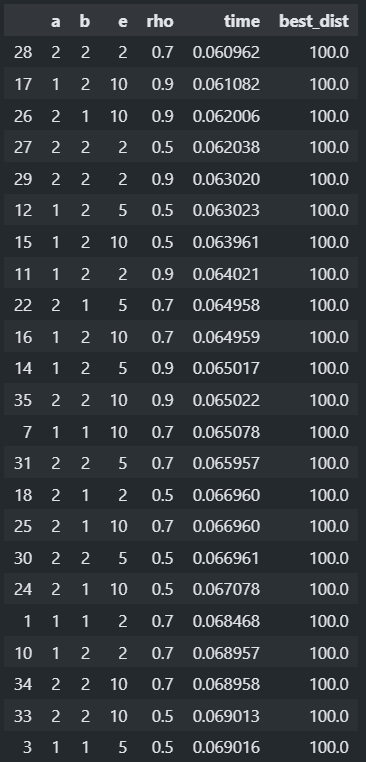


Рисунок 6 – Результат перебора параметров для матрицы 3x3

В результате, после эксперимента мы получили, что лучшие параметры для данной задачи являются таковыми:

1. Коэффициент запаха (a) = 2;
2. Коэффициент расстояния (b) = 2;
3. Количество элитных муравьев (e) = 2;
4. Коэффициент высыхания (rho) = 0.7.

Для достижения оптимальной производительности муравьиного алгоритма, были выбраны следующие параметры. Эти настройки позволяют алгоритму работать быстрее, сохраняя высокую точность решений. В результате экспериментов был найден самый короткий путь длиной 100.

Таким образом, для решения задачи коммивояжера с помощью муравьиного алгоритма важно тщательно подбирать параметры и учитывать размерность задачи, чтобы обеспечить баланс между скоростью и точностью работы алгоритма.

Также был добавлен графический интерфейс для работы с алгоритмом с помощью фреймворка «Streamlit». Пример работы приложения изображен на рисунках 7 и 8.

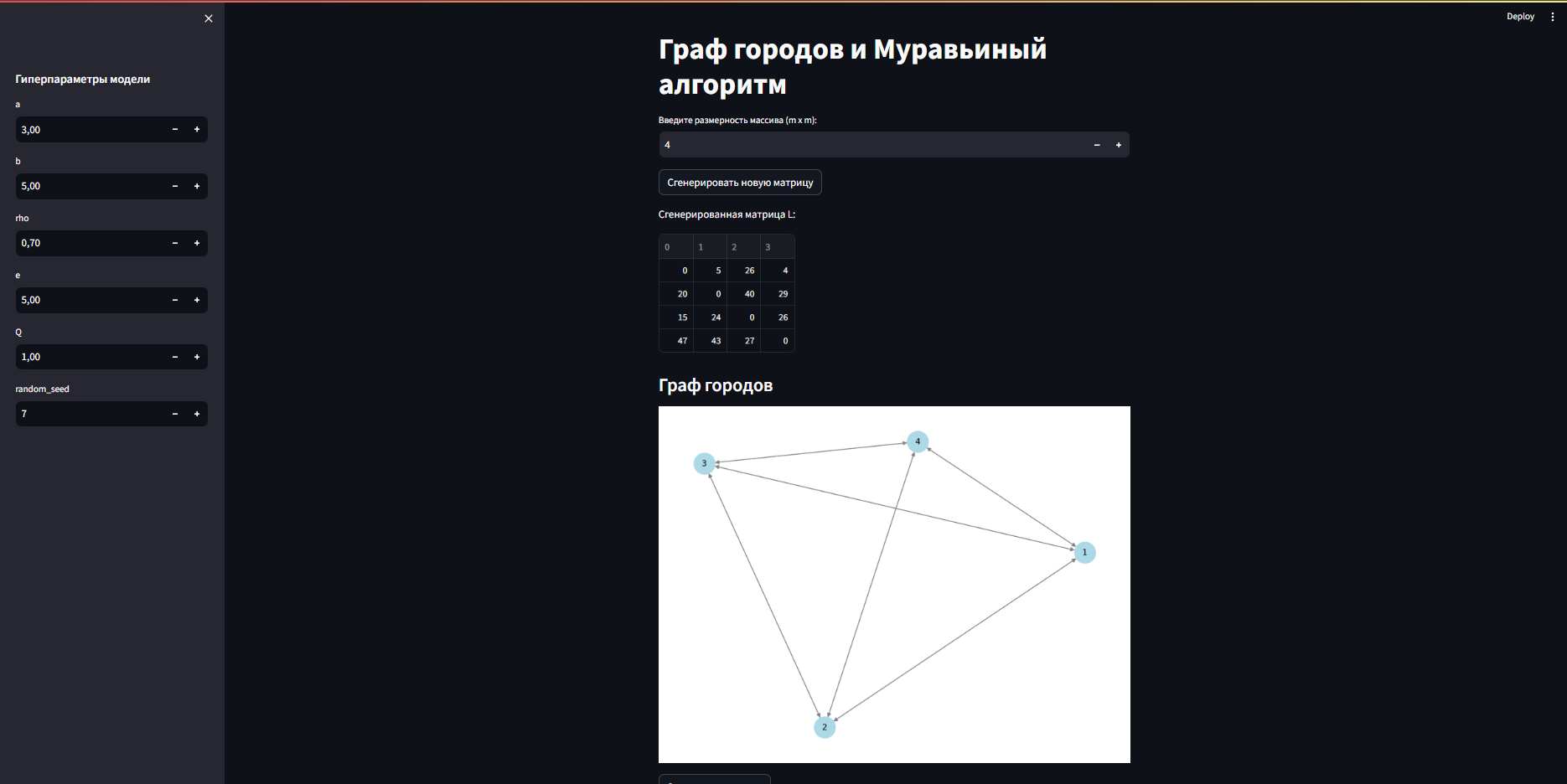


Рисунок 7 – Графический интерфейс

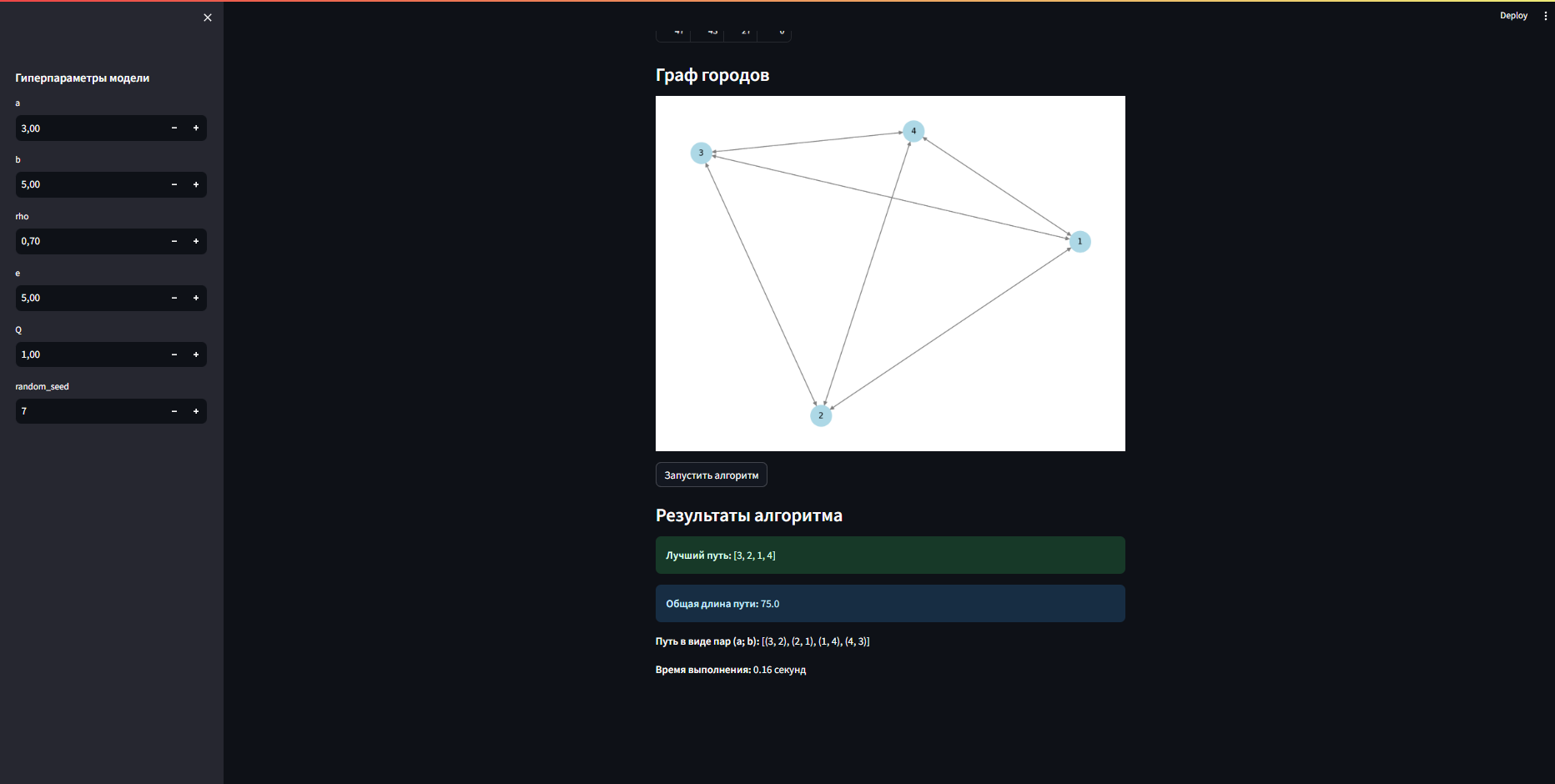


Рисунок 8 – Графический интерфейс

# ЗАКЛЮЧЕНИЕ

В данном проекте был разработан алгоритм муравьиной колонии для решения задачи коммивояжера. Программа позволяет пользователю вводить данные, такие как количество городов, матрица расстояний между ними, и затем выполнять моделирование для нахождения оптимального маршрута.

Был изучен алгоритм муравьиной колонии.

Был разработан алгоритм муравьиной колонии для задачи коммивояжера.

Были проведены обширные эксперименты, изменяя ключевые параметры алгоритма, чтобы определить их влияние на производительность и точность. Выяснилось, что каждый параметр существенно влияет на работу алгоритма, а также, что время работы напрямую зависит от размера матрицы расстояний.

Также был разработан дашборд с использованием Streamlit, который предоставляет пользователям удобный и интерактивный интерфейс. Пользователи могут вводить свои данные и экспериментировать с различными параметрами алгоритма, наблюдая за результатами в реальном времени. Визуализация маршрутов и значений приспособленности помогает пользователям лучше понимать процесс оптимизации и принимать обоснованные решения.

Хотя алгоритм показывает высокую эффективность, важно учитывать, что его производительность может снижаться при увеличении числа городов или сложности матрицы расстояний. В таких случаях может потребоваться использование более продвинутых методов или гибридных подходов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Клейнберг Дж., Тардос Е. Алгоритмы: разработка и применение. Классика Computers Science / Пер. с англ. Е. Матвеева. — СПб.: Питер, 2016. — 800 с.
2. Дасгупта С. и др. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО,2014. –– 320 с.
3. Handbook of Metaheuristics. International Series in Operations Research & Management Science. Volume 272. 2019. 604 p.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import numpy as np

START\_FROM = [

'random',

'form\_one',

'uniform'

]

ANT\_SYS\_MODIF = [

'cycle',

'density',

'quantity'

]

class AntAlgorithm:

def \_\_init\_\_(self,

a=1, # коэффициент запаха

b=1, # коэффициент расстояния

rho=0.5, # коэффициент высыхания

Q=1, # количество выпускаемого феромона

e=0, # количество элитных муравьев

local\_refresh=False, # локальное обновление феромона

where\_to\_start='random', # место начала движения муравьев

as\_modif='density', # модификация обновления феромона

random\_seed=-1 # фиксация генератора случайных чисел

):

self.a = a

self.b = b

self.rho = rho

self.Q = Q

self.e = int(e)

self.local\_refresh = local\_refresh

if where\_to\_start in START\_FROM:

self.where\_to\_start = where\_to\_start

else:

raise ValueError('wrong where to start')

if as\_modif in ANT\_SYS\_MODIF:

self.as\_modif = as\_modif

else:

raise ValueError('wrong ant-system modification name')

if random\_seed > 0:

np.random.seed(random\_seed)

# START\_FROM = {'random': randomly,

# 'form\_one': from\_one,

# 'uniform': uniform}

def fit(self,

L, # Матрица смежности графа

AGES=-1, # количество поколений

ANTS=-1, # количество муравьев в поколении

ph=-1 # начальное значение ферромона

):

self.L = L

self.CITIES = len(L)

if AGES > 0:

self.AGES = int(AGES)

else:

self.AGES = self.CITIES \* 50

if ANTS > 0:

self.ANTS = int(ANTS)

else:

self.ANTS = self.CITIES

if ph >= 0:

self.ph = ph

else:

self.ph = self.Q/self.CITIES

# инициализация матрицы "краткости" дуг графа

rev\_L = 1/L

# инициализация матрицы феромонов

tao = np.ones((self.CITIES, self.CITIES)) \* self.ph

self.BEST\_DIST = float("inf") # лучшая длина маршрута

self.BEST\_ROUTE = None # лучший маршрут

# матрица маршрутов муравьев в одном поколении (номера узлов графа)

antROUTE = np.zeros((self.ANTS, self.CITIES))

# вектор длины маршрута муравьев в одном поколении

antDIST = np.zeros(self.ANTS)

# вектор лучших длин маршрутов в каждом поколении

self.antBEST\_DIST = np.zeros(self.AGES)

self.antAVERAGE\_DIST = np.zeros(self.AGES)

# основной цикл алгоритма

# ---------- начало освновного цикла ----------

for age in range(self.AGES):

antROUTE.fill(0)

antDIST.fill(0)

# ---------- начало цикла обхода графа муравьями ----------

for k in range(self.ANTS):

if self.where\_to\_start == 'random':

# начальное расположение муравья в графе (случайное)

antROUTE[k, 0] = np.random.randint(

low=0, high=self.CITIES-1, size=1)

elif self.where\_to\_start == 'uniform':

# начальное расположение муравья в графе (равномерное)

antROUTE[k, 0] = k % self.CITIES

elif self.where\_to\_start == 'from\_one':

# начальное расположение муравья в графе (все с одного)

antROUTE[k, 0] = 1

else:

assert 'wrong where to start'

# ---------- начало обхода графа k-ым муравьем ----------

for s in range(1, self.CITIES):

# текущее положение муравья

from\_city = int(antROUTE[k, s-1])

P = (tao[from\_city] \*\* self.a) \* \

(rev\_L[from\_city] \*\* self.b)

# вероятность посещения уже посещенных городов = 0

for i in range(s):

P[int(antROUTE[k, i])] = 0

# вероятность выбора направления, сумма всех P = 1

assert (np.sum(P) > 0), \

"Division by zero. P = %s,"\

" \n tao = %s \n rev\_L = %s" % (

P, tao[from\_city], rev\_L[from\_city])

P = P / np.sum(P)

# выбираем направление

isNotChosen = True

while isNotChosen:

rand = np.random.rand()

for p, to in zip(P, list(range(self.CITIES))):

if p >= rand:

# записываем город №s в вектор k-ого муравья

antROUTE[k, s] = to

isNotChosen = False

break

# локальное обновление феромона

if self.local\_refresh:

for s in range(self.CITIES):

city\_to = int(antROUTE[k, s])

city\_from = int(antROUTE[k, s-1])

if self.as\_modif == 'cycle':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + \

(self.Q / antDIST[k])

elif self.as\_modif == 'density':

tao[city\_from, city\_to] = tao[city\_from,

city\_to] + self.Q

elif self.as\_modif == 'quantity':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + \

(self.Q / L[city\_from, city\_to])

else:

assert 'wrong ant-system modification name'

tao[city\_to, city\_from] = tao[city\_from, city\_to]

# ---------- конец цила обхода графа ----------

# вычисляем длину маршрута k-ого муравья

for i in range(self.CITIES):

city\_from = int(antROUTE[k, i-1])

city\_to = int(antROUTE[k, i])

antDIST[k] += L[city\_from, city\_to]

# сравниваем длину маршрута с лучшим показателем

if antDIST[k] < self.BEST\_DIST:

self.BEST\_DIST = antDIST[k]

self.BEST\_ROUTE = antROUTE[k]

# ---------- конец цикла обхода графа муравьями ----------

# ---------- обновление феромонов----------

# высыхание по всем маршрутам (дугам графа)

tao \*= (1-self.rho)

# цикл обновления феромона

for k in range(self.ANTS):

for s in range(self.CITIES):

city\_to = int(antROUTE[k, s])

city\_from = int(antROUTE[k, s-1])

if self.as\_modif == 'cycle':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + \

(self.Q / antDIST[k])

elif self.as\_modif == 'density':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + self.Q

elif self.as\_modif == 'quantity':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + \

(self.Q / L[city\_from, city\_to])

else:

assert 'wrong ant-system modification name'

tao[city\_to, city\_from] = tao[city\_from, city\_to]

# проход элитных е-муравьев по лучшему маршруту

if self.e > 0:

for s in range(self.CITIES):

city\_to = int(self.BEST\_ROUTE[s])

city\_from = int(self.BEST\_ROUTE[s-1])

if self.as\_modif == 'cycle':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + \

((self.Q \* self.e) / self.BEST\_DIST)

elif self.as\_modif == 'density':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + self.Q \* self.e

elif self.as\_modif == 'quantity':

tao[city\_from, city\_to] = \

tao[city\_from, city\_to] + \

((self.Q \* self.e) / L[city\_from, city\_to])

else:

assert 'wrong ant-system modification name'

tao[city\_to, city\_from] = tao[city\_from, city\_to]

# ---------- конец обновления феромона ----------

# конец поколения муравьев

# сбор информации для графиков

self.antBEST\_DIST[age] = self.BEST\_DIST

self.antAVERAGE\_DIST[age] = np.average(antDIST)

return tao

def get\_best\_route(self):

return self.BEST\_ROUTE

def get\_best\_dist(self):

return self.BEST\_DIST

def get\_best\_dists(self):

return [age for age in range(self.AGES)], self.antBEST\_DIST

def get\_average\_dists(self):

return [age for age in range(self.AGES)], self.antAVERAGE\_DIST