МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**АЛГОРИТМ МУРАВЬИНОЙ КОЛОНИИ ДЛЯ ОПТИМИЗАЦИИ ДОСТАВКИ ОДНОРОДНОГО ГРУЗА РАЗЛИЧНЫМ КЛИЕНТАМ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. А. Дука

(подпись)

Направление подготовки 02.03.02 — «Фундаментальная информатика и\_\_\_\_\_

(код, наименование)

информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ курс\_\_\_\_\_3\_\_\_\_\_

Направленность (профиль)\_\_\_\_Математическое и программное обеспечение компьютерных технологий\_\_\_\_

Научный руководитель

канд. техн. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. Е. Полупанова

(подпись, дата)

Нормоконтролер

преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. А. Нигодин

(подпись, дата)

Краснодар

2024

**РЕФЕРАТ**

Курсовая работа 38 с., 4 ч., 29 рис., 1 табл., 10 источн., 4 прил.

ОПТИМАЛЬНЫЙ ПУТЬ, ГРАФ, МУРАВЬИНЫЙ АЛГОРИТМ

Объектом исследования в данной практической работе является алгоритм поиска оптимального пути обхода нескольких вершин в заданном графе произвольных размеров.

Цель работы: создание программы, позволяющей демонстрировать базовые возможности для работы с алгоритмом муравьиной колонии, а также служащей основой для дальнейших исследований и доработок в направлении оптимизации маршрутов доставки груза с несколькими целевыми точками

Методологическая основа исследования включает в себя эвристический метод, анализ результатов, графический метод.

В результате работы было разработано приложение, которое строит граф для доставки, находит оптимальный маршрут для доставки однородного груза с помощью алгоритма муравьиной колонии и визуализирует полученный маршрут на графе.

Научная новизна работы заключается в том, что предлагается алгоритм для модифицированной задачи о коммивояжёре, заключающейся в поиске оптимального пути в графе для прохождения через обязательные точки один раз без возвращения в точку исхода, другие готовые решения которой труднодоступны и не распространены.

В результате решения поставленных задач были определены оптимальные параметры для базового муравьиного алгоритма, которые позволяют оптимизировать время и качество работы алгоритма поиска оптимального маршрута доставки. Разработанный алгоритм в большинстве случаев решает поставленную задачу нахождения кратчайшего пути доставки, при этом не хуже модифицированных точных алгоритмов, используемых в современных навигационных системах.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc184843895)

[1 Общие сведения об алгоритмах поиска пути в графе и оптимизации маршрута 7](#_Toc184843896)

[1.1 Понятие графа 7](#_Toc184843897)

[1.2 Точные алгоритмы поиска путей в графах 10](#_Toc184843898)

[1.3 Эвристические алгоритмы 12](#_Toc184843899)

[1.4 Необходимость разработки эвристического алгоритма 14](#_Toc184843900)

[2 Задача оптимизации маршрута 16](#_Toc184843901)

[2.1 Постановка задачи доставки груза 16](#_Toc184843902)

[2.2 Математическая постановка задачи 18](#_Toc184843903)

[2.3 Муравьиный алгоритм 20](#_Toc184843904)

[3 Реализация приложения 22](#_Toc184843905)

[3.1 Основные сведения о программе и её компоненты 22](#_Toc184843906)

[3.2 Интерфейс и этапы работы с программой 25](#_Toc184843907)

[3.3 Проверки и ограничения 29](#_Toc184843908)

[4 Требуемые модификации 31](#_Toc184843909)

[ЗАКЛЮЧЕНИЕ 33](#_Toc184843910)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#_Toc184843911)

[ПРИЛОЖЕНИЕ А Файл управления моделью 37](#_Toc184843912)

[ПРИЛОЖЕНИЕ Б 54](#_Toc184843913)

[ПРИЛОЖЕНИЕ В 55](#_Toc184843914)

[ПРИЛОЖЕНИЕ Г 56](#_Toc184843915)

# ВВЕДЕНИЕ

Актуальность данной работы состоит в необходимости разработки новых алгоритмов и методов решения трудноразрешимых задач оптимизации маршрутов в логистике и в навигации. Одной из таких задач является поиск оптимального пути обхода нескольких точек доставки. В настоящее время традиционные точные алгоритмы, например, Дейкстры, не могут эффективно решить эту задачу без дополнительных модификаций при больших входных данных и дополнительных условий за приемлемое время. В подобных случаях могут помочь эвристические алгоритмы, основанные на природных явлениях, такие как генетические алгоритмы, жадные, эволюционные. Одним из них является алгоритм муравьиной колонии, способный решить NP-сложную задачу поиска оптимального маршрута, требующую экспоненциального времени для решения 𝑂*(n!)* – а значит использование и разработка эвристических методов для нахождения приближенного решения востребованы.

Основная цель работы – создание программы, позволяющей демонстрировать базовые возможности для работы с алгоритмом муравьиной колонии, а также служащей основой для дальнейших исследований и доработок в направлении оптимизации маршрутов доставки груза с несколькими целевыми точками.

Для реализации поставленной цели предполагается решить следующие задачи:

* изучить теорию алгоритмов, решающих задачу поиска кратчайшего пути в графах, а также модифицированную задачу о коммивояжёре;
* разработать базовую программу, предназначенную для решения задачи поиска оптимального пути обхода одной точки доставки груза;
* поэкспериментировать с разными видами графов и входными данными, включающими количество пунктов назначения и параметры алгоритма;
* найти и проанализировать доступность и функционал готовых современных продуктов, решающих поставленную задачу, и сравнить со своей программой;
* найти способы модифицировать алгоритм и программу для решения задачи оптимизации обхода нескольких точек назначения, которые потребуется для дальнейшего развития темы работы.

Объектом исследования в данной практической работе является алгоритм поиска оптимального пути обхода нескольких вершин в заданном графе произвольных размеров.

Предметом исследования является выявление качественных модификаций работы базового муравьиного алгоритма для поиска оптимального пути обхода нескольких точек в графе.

Информационной базой исследования являются результаты работы программы. Методологическая основа исследования включает в себя эвристический метод (многократный запуск программы для различных графов и оценка эффективности приближенных решений), сбор статистической информации, а также оценка среднего времени работы алгоритма поиска оптимального пути обхода, графический метод (построение графа и отображение стоимости и уровня феромонов на определенных маршрутах), который позволяет визуализировать и анализировать влияние входных данных на процесс и результат работы программы, аналогия (сравнение работы алгоритма с механизмом или явлением в природе, в частности с использованием муравьиными колониями феромонов при поиске и отборе оптимальных путей), синтез выводов (выявление влияния различных параметров на эффективность и производительность работы алгоритма).

Научная новизна работы заключается в том, что предлагается новый алгоритм для модифицированной задачи о коммивояжёре, заключающейся в поиске оптимального пути прохождения в графе через обязательные точки один раз без возвращения в точку исхода, другие готовые решения которой труднодоступны и не распространены.

Теоретическая и практическая значимость работы состоит в широком применении алгоритма решения задачи нахождения оптимального пути обхода нескольких конечных вершин в заданном графе произвольных размеров в различных областях, где важна скорость и эффективность. К примеру, в решении этой задачи нуждается логистика с планированием коммерческих маршрутов, медицина для оптимизации маршрутов бригад скорой помощи или навигация в городской среде для служб доставки, где необходимо учитывать множество точек назначения. Кроме того, подобная задача встречается в робототехнике, а именно в планировании маршрутов роботов с препятствиями, например, в производственных помещениях. Также, алгоритм поиска маршрута с несколькими узлами необходим при проектировании сетей связи и оптимизации маршрутов передачи данных между узлами.

# Общие сведения об алгоритмах поиска пути в графе и оптимизации маршрута

## Понятие графа

Граф — это математическая структура, которая состоит из множества вершин (узлов) и множества соединяющих их рёбер. Эта структура позволяет наглядно представить сложные взаимосвязи и взаимодействия между объектами или явлениями. Графы как схемы связи в различных областях по типу компьютерных сетей, транспортных [2].

Для понимания объекта работы с алгоритмом необходимо классифицировать возможные типы графов в работе, так как каждый тип задает различные условия маршрутизации.

Неориентированный взвешенный граф – не имеющий четкого направления ребра, являющегося двусторонним, и с весовой стоимостью, представляющей, например, время или расстояние перемещения между двумя узлами. Связный – в таком графе каждая вершина является достижимой, то есть возможен путь из любой вершины в каждую. Пример такого графа представлен на рисунке 1.

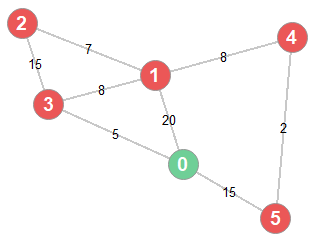


Рисунок 1 – Неориентированный взвешенный связный граф

Полный – граф, в котором каждые две вершины соединены ребром. То есть из любого узла есть ребро в любое другое. Зависимость количества рёбер в неориентированном полном графе от набора вершин представлена в формуле (1.1):

(1.1)

где

– искомое количество рёбер графа;

– искомое количество рёбер графа;

Пример полного графа представлен на рисунке 2.

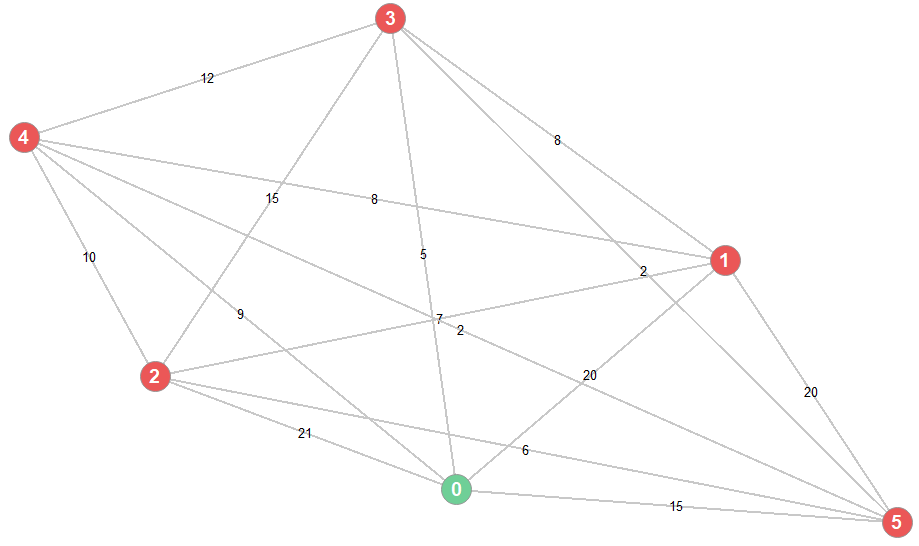


Рисунок 2 – Полный граф

Неполный – граф, в котором существуют вершины, не имеющие связи с каждой другой. Наличие возможных “тупиковых” узлов имеет особое значение при обработке событий маршрутизации. Пример неполного графа представлен на рисунке 3.

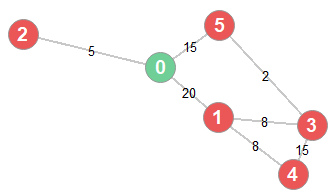


Рисунок 3 – Неполный граф

## Точные алгоритмы поиска путей в графах

Точные алгоритмы поиска путей в графах обеспечивают нахождение кратчайшего пути и возвращают решение, соответствующее критерию оптимальности. Часто они основаны на переборе всех возможных вариантов решений, среди которых находится лучшее, подходящее под определенные условия задачи.

Алгоритм Дейкстры – находит кратчайшее расстояние от одной из вершин графа до всех остальных [1]. Алгоритм находит все возможные пути от одной вершины до остальных, начиная с исходной постепенно расширять поиск жадным подходом, выбирая на каждом шаге наименьшее по стоимости ребро до соответствующей вершины. Таким образом для нахождения решения задачи будет перебрана каждая вершина графа, что ухудшает производительность при большом количестве узлов графа.

Алгоритм Беллмана-Форда – находит кратчайшие пути, работая как с отрицательными, так и с положительными весами ребер. Суть заключена в выполнении релаксации рёбер на каждом шаге, то есть в поиске более короткого пути до начальной точки через соседние вершины. Так же этот алгоритм имеет сложность, прямо пропорционально зависящую от количества ребер и вершин.

Алгоритм Флойда-Уоршелла – использует динамическое программирование для поиска кратчайших путей между всеми узлами графа, постепенно минимизируя расстояние с учетом промежуточных вершин. Это итерационный метод поиска пути, в каждой итерации которого обновляется расстояние между всеми парами вершин с поиском более короткого через промежуточную вершину по формуле (1.2):

(1.2)

где

– текущее расстояние между вершинами и ;

– потенциально более короткий путь через вершину .

Таким образом, выше затронутые точные или “строгие” алгоритмы являются одними из основных “традиционных” методов поиска кратчайшего пути в графе. Все они имеют общую черту в нахождении или переборе большинства или всего множества из возможных вариантов путей к каждой вершине, что понижает производительность обработки графов при повышении количества узлов и ребер.

## Эвристические алгоритмы

Эвристика – научная отрасль, изучающая избирательный поиск решения задачи на основе логического обоснования, методы которой дают достаточно оптимальное решение на основе алгоритма, не являющегося гарантированно точным.

Эвристический алгоритм – алгоритм решения поставленной задачи, дающий достаточно хорошее и полное, но не полностью и не точно обоснованное теоретически, решение в большинстве случаев. Суть таких алгоритмов заключена в использовании их в задачах с высокой вычислительной сложностью, благодаря отличительной в данных условиях от точных алгоритмов быстроте поиска конечного решения. Зачастую такие алгоритмы вдохновлены природными, социальными или эволюционными явлениями, суть которых отражается в самом названии.

Рассмотрим наиболее известные эвристические алгоритмы.

1. Генетические алгоритмы: образно основаны на процессах естественного отбора и генетики, процессах скрещивания и мутирования. Каждое решение является индивидом, проходящим этап комбинирования с другими, отбора в лучшие или неподходящие решения, таким образом процесс повторяется до тех пор, пока результат решения не станет оптимальным для условия задачи, либо пока не исчерпается время для поиска или количество итераций не превзойдёт заданного.
2. Алгоритм пчелиной колонии: вдохновлен поведением пчел, ищущих нектар, основываясь на его предполагаемом качестве на отдельных путях. За качество зачастую принимается количество потенциального ресурса – “пчёлы”, находящие большие запасы передают информацию о пройденном пути, после чего он становится более приоритетным.
3. Иммунный алгоритм: несет общую схожесть с принципами иммунитета, решения задачи идентифицируются как антигены, которые отбираются системой на основе классификации решений, чтобы выбрать наиболее подходящие и комбинировать их с другими.
4. Алгоритм симулированного отжига: имитирует процесс отжига металлов, в основе которого лежит его нагрев и последующее охлаждение, во время чего частицы материала могут становится стабильнее, суть алгоритма в том, чтобы на начальных этапах находить более “агрессивные” решения, что может привести к нахождению даже плохого, но сглаживаемого последующими итерациями решения, благодаря уменьшению вероятности выбора плохого решения в сравнении с другим, такой подход помогает избежать проблемы остановки в локальных оптимумах, не позволяющих нахождению глобально корректного решения.
5. Алгоритм муравьиной колонии – алгоритм, основанный на процессе поиска пути к ресурсу муравьиными колониями, суть идеи состоит в оставлении на удачном пути феромонного следа и его постепенного испарения или обновления большей концентрацией со временем, таким образом с каждой итерацией выявляются более привлекательные пути с меньшей стоимостью и большей скоростью прохождения.

## Необходимость разработки эвристического алгоритма

Таким образом для обоснования необходимости выбора и разработки эвристических алгоритмов в условиях поиска решений задач с более сложными и масштабными входными данными можно привести сравнительную характеристику точных и эвристических алгоритмов:

* гарантированное решение: точные алгоритмы всегда гарантируют нахождение точного решения задачи, когда эвристические алгоритмы предоставляют только приближённое решение;
* сложность вычислений: точные алгоритмы имеют высокую вычислительную сложность, когда эвристические алгоритмы отличаются меньшей сложностью и более высокой производительностью;
* применимость: точные алгоритмы подходят для решения задач малой и средней сложности, когда эвристические алгоритмы предназначены для решения больших и сложных задач, где требуется исследовать большое пространство решений;
* гибкость и адаптивность: точные алгоритмы имеют ограниченную гибкость и плохо адаптируются для других типов задач, когда эвристические алгоритмы хорошо подходят для задач с дополнительными условиями без необходимости модификации самого алгоритма;
* риск застревания: точные алгоритмы всегда находят глобальный минимум (оптимальное решение), когда эвристические алгоритмы имеют риск застревания в локальном минимуме, особенно если решение ограничено узкой областью.

Отметим теперь достоинства применения муравьиного алгоритма, демонстрирующего высокую эффективность и применимость в задачах транспортной логистики, особенно при решении задач маршрутизации. Основные преимущества включают:

* эффективность в задачах коммивояжера: муравьиные алгоритмы отлично справляются с задачей поиска маршрутов, особенно при малом количестве узлов, где возможен даже полный перебор, однако алгоритм показывает гораздо более высокую скорость работы;
* работа с большими графами: алгоритм успешно применяется к задачам с большим числом узлов, несмотря на их вычислительную сложность (NP-трудность), при этом сохраняется приемлемая скорость сходимости;
* сравнение с другими методами: по сравнению с глобальными методами оптимизации, такими как генетические алгоритмы или нейронные сети, муравьиные алгоритмы часто обеспечивают более качественные решения в задачах логистики;
* стабильность решений: алгоритм учитывает весь исторический опыт колонии, что снижает влияние ошибок начального выбора и делает решения более устойчивыми, особенно в сравнении с генетическими алгоритмами;

Несмотря на эти преимущества, алгоритм обладает рядом ограничений:

* сложность анализа: теоретическое исследование характеристик муравьиных алгоритмов затруднено, так как решения зависят от вероятностных распределений;
* зависимость от параметров: для эффективной работы требуется тщательная настройка параметров, таких как интенсивность испарения феромонов, количество муравьёв и эвристические коэффициенты;
* эмпирический подход: настройка алгоритма часто требует серии экспериментов, так как его поведение зависит от множества факторов;
* непредсказуемость сходимости: хотя алгоритм демонстрирует хорошие результаты, время достижения решения не всегда можно точно предсказать.

# Задача оптимизации маршрута

## Постановка задачи доставки груза

Одной из известных задач оптимизации маршрута является задача о коммивояжёре – путешественнике, перед которым предстаёт задача посетить все пунктов торговли, не повторяя пройденных маршрутов и точек, и вернуться в исходную, откуда начал. В данной работе эта задача предстаёт в модифицированном варианте, описать которую можно характеризующими её пунктами:

* выбирается начальная точка отправления и конечная;
* после прохождения всех обязательных пунктов назначения, последний становится конечным;
* обратный путь повторяется и полностью дублируется в обратном порядке.

Следует дать более устоявшееся название этому типу задач – оптимизация доставки грузов (Vehicle Routing Problem) – главная цель решения которого является планирование маршрутов или маршрутизация для транспортных средств оптимальным способом. Этот тип часто встречается в логистике и транспортировке при планировании посещения нескольких точек назначения.

Особенностями этой задачи является наличие нескольких параметров:

* груз: может быть однородным, то есть исчислимым в массе, объеме или количестве – в данной работе не учитываются показатели груза и используется понятие его однородности – исчисление в целых обособленных единицах;
* клиенты: может быть один или несколько клиентов – в нашей работе мы задаем пока вариант одного клиента, так как возможных маршрутов для нескольких конечных точек значительно больше, что и усложняет решение этой задачи и требует модификации базового алгоритма;
* граф: может быть ориентированным и неориентированным. В ходе данной работы будут использоваться двунаправленные рёбра;
* стоимость пути: время, расстояние, промежуточных узлов – все обозначения имеют похожее, но разное значение, например, скорость маршрута отличается от протяженности, так как в условиях маршрутизации самый короткий маршрут может оказаться не самым быстрым из-за пробок, ремонтных работ или климатических условий и особенностей дороги, в контексте задачи работы мы используем стоимость как общее понятие, так как вышеупомянутые факторы не имеют значения для неё.

## ****Математическая постановка задачи****

Пусть имеется неориентированный взвешенный граф G с непустым множеством вершин V и непустым множеством рёбер E с заданным весом. Для него задана начальная точка и множество конечных точек , которые нужно посетить в процессе выполнения маршрута

Задача сводится к минимизации целевой функции, то есть общего расстояния маршрута, который проходит через все конечные точки Т, как показано в формуле (2.1):

(2.1)

где

– минимизация значения выражения;

– сумма выражения;

– ребро, соединяющее вершину с вершиной ;

– множество рёбер графа;

– бинарная переменная со значением 1 и 0 если ребро включено в маршрут и нет соответственно;

– вес ребра.

Так же в нашей присутствует ряд ограничений, обусловленных поставленной задаче.

1. Связность маршрута: если вершина (точка) включена в маршрут, то из неё должно выходить ровно одно ребро (дорога), и в неё должно входить ровно одно ребро. Например, если вы доставляете груз в точку B, вы должны туда прийти и уйти.
2. Посещение конечной точки: заданная конечная точка обязательно должна быть включена в маршрут. К примеру, если начальная и конечная точки доставки указаны как A и B соответственно, то конечную точку нужно посетить обязательно произвольным маршрутом возможно другие посреднеческие.
3. Начальная точка: маршрут начинается с начальной точки .
4. Ребро включается только если обе точки посещены: дорога между двумя точками включается в маршрут только тогда, когда обе эти точки включены в маршрут. Например, нельзя использовать дорогу A→B, если вы не планируете посетить точку A или B.
5. Открытость маршрута: маршрут не должен возвращаться обратно в начальную точку . То есть если вы начали с A, доставка заканчивается в конечной точке, например, B или C, но не возвращается к A.

## ****Муравьиный алгоритм****

Алгоритм муравьиной колонии – это эвристический метод, предназначенный для приближённого решения задач поиска оптимальных маршрутов в графах. Его подход основан на имитации поведения муравьёв в природе, которые прокладывают пути от муравейника к источнику пищи.

В реальных условиях муравьи двигаются случайным образом, но, найдя источник пищи, возвращаются к муравейнику, оставляя на своём пути феромонный след. Другие муравьи, обнаружив эту тропу, с большей вероятностью будут следовать по ней. При каждом успешном нахождении пищи муравьи усиливают данный путь, увеличивая количество феромонов. Однако феромоны со временем испаряются, что снижает привлекательность старых троп и уменьшает вероятность их выбора.

Эффект испарения особенно важен для предотвращения зацикливания муравьёв на локально-оптимальных маршрутах. Длинные маршруты теряют феромоны быстрее из-за более продолжительного времени их обхода, тогда как короткие маршруты чаще обновляются, оставаясь более привлекательными. Без механизма испарения муравьи продолжали бы использовать лишь первый найденный маршрут, даже если он не является оптимальным.

Любой алгоритм, независимо от модификаций, можно представить в следующем виде: «Пока (условия выхода не выполнены) 1) создаем муравьев; 2) ищем решения; 3) обновляем феромон; 4) осуществляем в соответствии с опциями дополнительные действия» [3].

Алгоритм муравьиной колонии можно представить, как цикл, содержащий 3 этапа.

1. Создание муравьев: выбирается стартовая точка, куда помещаются муравьи. Они могут располагаться все в одной точке, в разных с повторением или в разных без повторений. На этом шаге задаётся начальное значение феромонов, чтобы на первом шаге вероятности перехода муравьев в соседние вершины не были нулевыми [3].
2. Поиск решения: вычисляется вероятность перехода муравья из вершины i в вершину j по следующей формуле (2.1):

[8] (2.1)

где

– уровень феромона;

– расстояние из i в j вершину;

– константные параметры: при = 0 алгоритм становится более жадным, так как выбирает ближайшего соседа, а при = 0 выбирается ребро только на основе феромона, что приводит к субоптимальным решениям.

1. Обновление феромонов: уровень феромонов изменяется с помощью следующие формулы (2.2):

[8] (2.2)

где

– интенсивность испарения;

– цена текущего решения для k-го муравья;

Q – параметр, имеющий значение порядка цены оптимального решения.

1. Осуществление дополнительного локального поиска.

# Реализация приложения

## Основные сведения о программе и её компоненты

Для реализации поставленной задачи была разработана программная система с графическим интерфейсом. В качестве фреймворка для построения графического интерфейса выбрана библиотека PyQt5, которая обеспечивает широкие возможности по созданию интерфейсов. Отрисовка графа и визуализация результатов осуществляются при помощи pyqtgraph, что позволяет интерактивно отображать вершины, рёбра и уровни феромонов. Для работы с графом, его вершинами, рёбрами и весами используется библиотека NetworkX, упрощающая операции над графовыми структурами.

Программа, диаграмма классов которой изображена на рисунке 4, реализована с применением архитектурного паттерна MVC(Model-View-Controller), вот её основные разделы и методы:

1. controller.py: основной файл, отвечающий за координацию работы модели и представления, содержит:
2. run() – метод, запускающий приложение и связывающий представление с моделью;
3. load\_graph() – метод загрузки графа из текстового файла и проверки корректности данных;
4. update\_start\_node() – метод для обновления стартовой вершины в модели;
5. update\_end\_nodes() – метод проверки и обновления конечных точек в модели;
6. set\_parameters() – метод задания параметров алгоритма (коэффициент испарения, интенсивность феромонов и др.);
7. start\_algorithm() – метод, запускающий выполнение муравьиного алгоритма;
8. stop\_algorithm() – метод, останавливающий выполнение алгоритма;
9. reset\_graph() – метод для сброса состояния алгоритма и начальных/конечных точек;
10. graph\_model.py – файл, содержащий реализацию модели работы графа и алгоритма муравьиной колонии, включает:
11. initialize\_pheromones() – метод инициализации феромонов на рёбрах графа;
12. simulate\_ant\_path() – метод симуляции пути для одного муравья с проверкой достижения конечных точек;
13. select\_next\_edge() – метод выбора следующей вершины для муравья на основе вероятности;
14. update\_pheromones() – метод обновления уровня феромонов на рёбрах в каждой итерации;
15. calculate\_path\_length() – метод расчёта общей длины маршрута для заданного пути;
16. run\_aco() – основной метод запуска муравьиного алгоритма с итерациями;
17. window\_view.py – файл представления, отвечающий за интерфейс пользователя, включает:
18. load\_styles() – метод загрузки стилей интерфейса из файла;
19. create\_controls() – метод создания элементов управления для задания начальных и конечных точек, параметров алгоритма;
20. create\_buttons() – метод добавления кнопок управления (загрузка графа, запуск алгоритма и сброс);
21. update\_start\_node\_combo() – метод обновления выпадающего списка начальных точек;
22. update\_canvas() – метод визуализации графа на виджете PyQtGraph: отображение вершин (стартовой, конечных и промежуточных), отрисовка рёбер с толщиной, зависящей от уровня феромонов, подпись рёбер с указанием расстояний между вершинами.

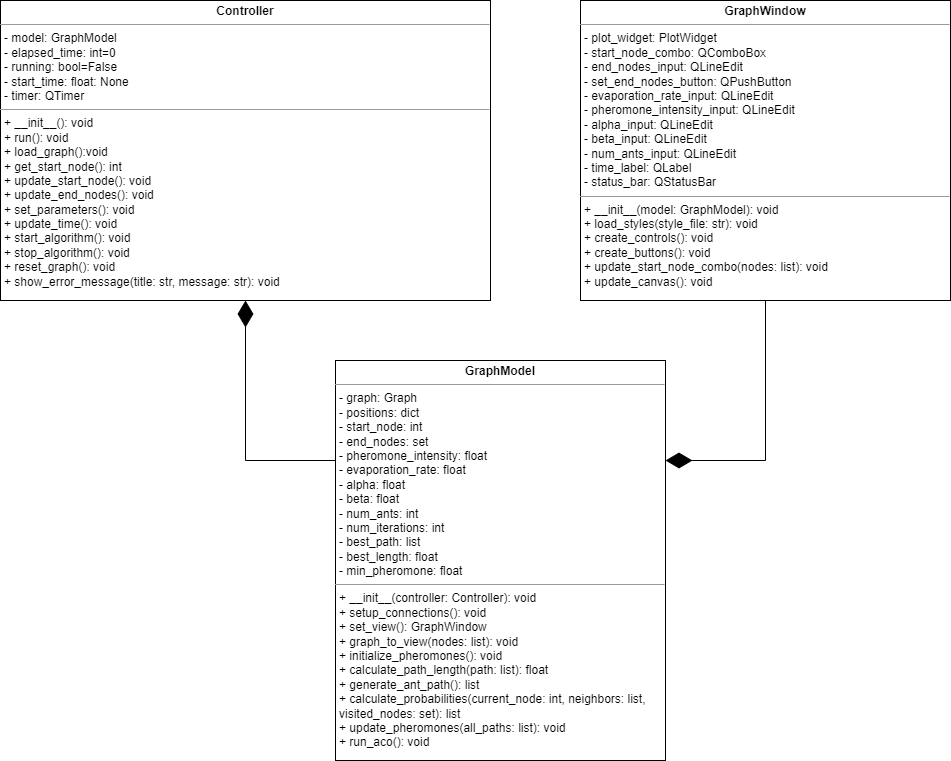


Рисунок 4 – Диаграмма классов программы

## Интерфейс и этапы работы с программой

При запуске программы пользователь видит основное окно со структурированным интерфейсом, представленном на рисунке 5. Сюда входит: центральная область с графическим виджетом отображения графа, которая изначальна пуста; панель для ввода настройки и ввода параметров алгоритма (начальная и конечные точки, коэффициент испарения феромонов, интенсивность феромонов, константные переменные веса расстояния и феромонов, количество муравьёв); ряд функциональных кнопок “Загрузить граф”, “Задать параметры”, “Запуск”, “Остановка”, “Сброс”; таймер для подсчета времени работы алгоритма; статус-бар, который отображает в текстовом формате состояние программы, результат, а так же ошибки при возникновении.

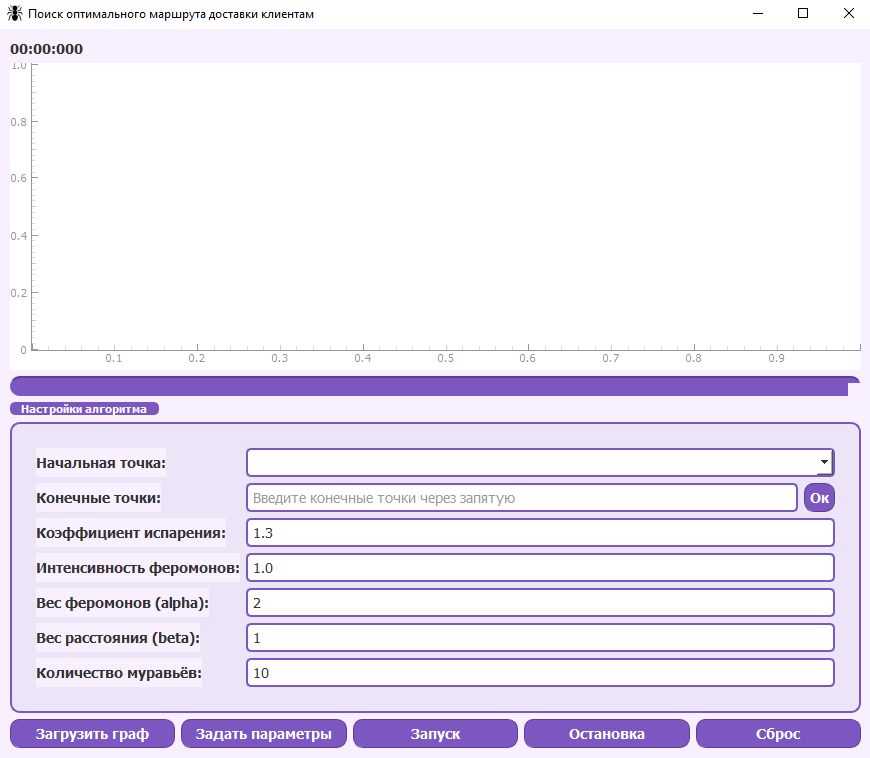


Рисунок 5 – Интерфейс основного окна программы

Поэтапная работа программы включает в себя несколько шагов.

1. Загрузка графа: Пользователь нажимает кнопку «Загрузить граф», выбирает файл, содержащий вершины и рёбра с весами. Граф отображается на экране, вершины распределяются по плоскости с помощью алгоритма расстановки (spring layout). Пример отображенного графа показан на рисунке 6.

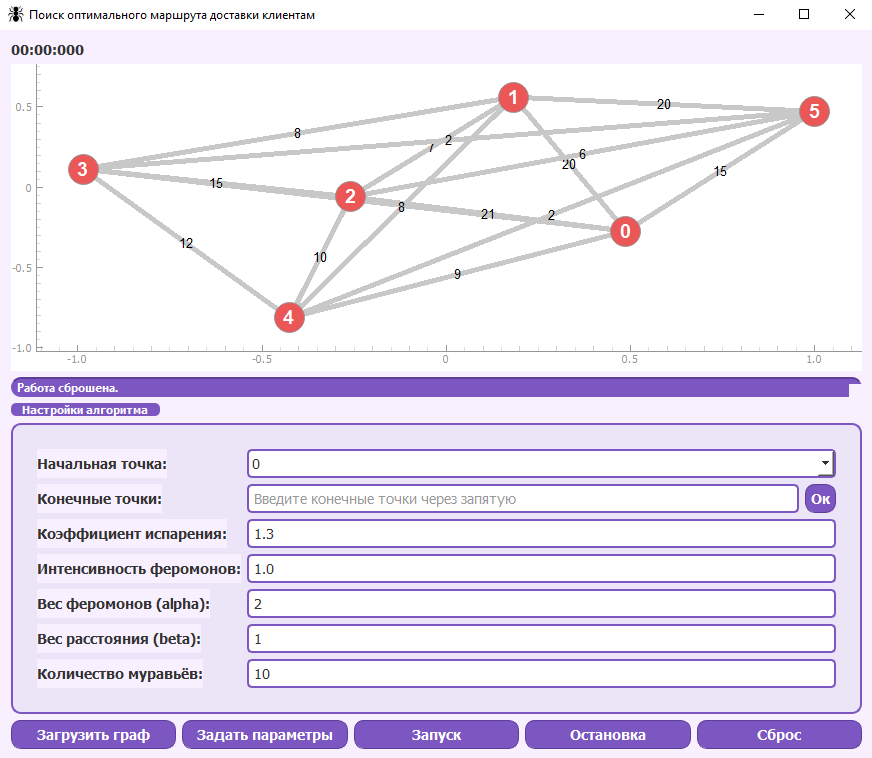


Рисунок 6 – Пример отображения загруженного графа

1. Задание параметров: Пользователь вводит параметры алгоритма в соответствующие поля и нажимает «Задать параметры». В модели обновляются значения ρ, Q, α, β, количество муравьёв и итераций.
2. Выбор стартовой и конечных вершин: из выпадающего списка выбирается стартовая вершина. В текстовое поле вводятся конечные вершины, по нажатию «Ок» они устанавливаются в модели. Пример графа с выбранными точками и заданными параметрами представлен на рисунке 7.



Рисунок 7 – Пример графа с выбранными точками и параметрами

1. Запуск алгоритма: Пользователь нажимает «Запуск». Контроллер инициирует в модели итеративный процесс: генерируются муравьи, строятся пути, рассчитываются вероятности, обновляются феромоны. Блок-схема алгоритма представлена на рисунке 10. Визуально в интерфейсе можно наблюдать постепенные изменения толщины рёбер (по мере изменения концентрации феромонов), что можно увидеть на рисунке 8.

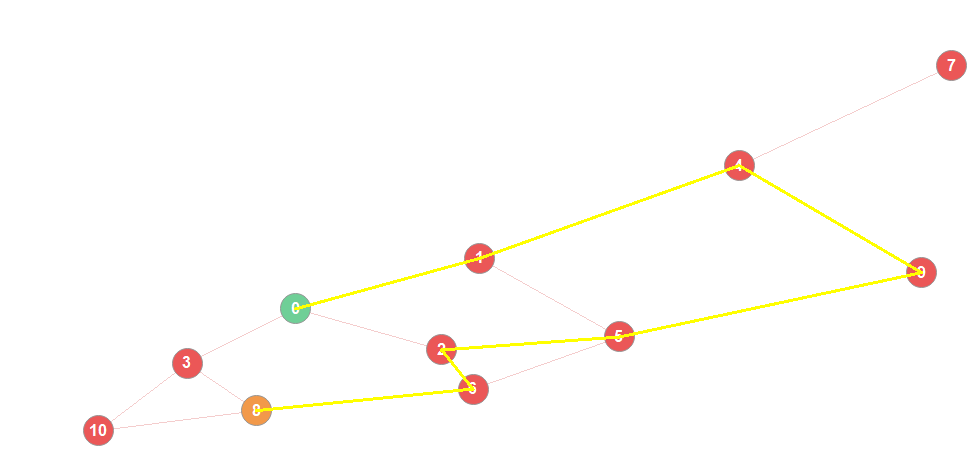


Рисунок 8 – Постепенное изменение уровня феромона на ребрах

1. Окончание расчётов: По достижении заданного числа итераций или улучшения качества решения программа останавливается. Лучший найденный маршрут и его длина отображаются в статус-баре, а на графе выделяется путь с повышенным уровнем феромонов, что можно увидеть на рисунке 9.

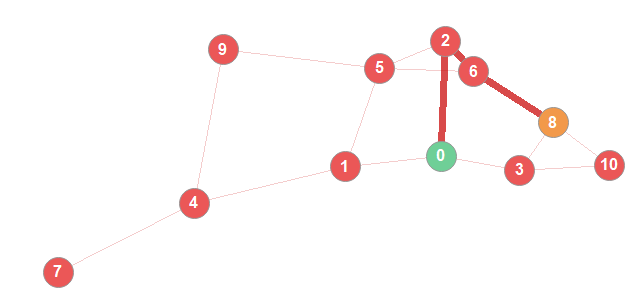


Рисунок 9 – Найденный путь с повышенным уровнем феромонов

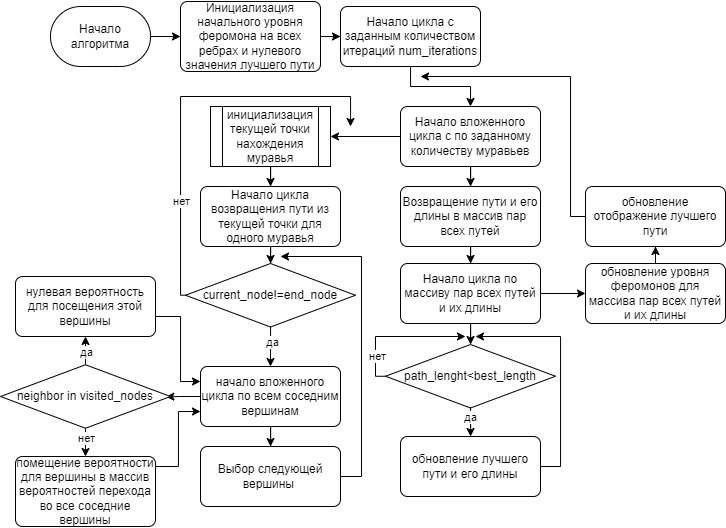


Рисунок 10 – Блок-схема базового алгоритма муравьиной колонии

## Проверки и ограничения

В процессе работы программы предусмотрен ряд проверок и валидаций данных, направленных на обеспечение корректности входных данных, параметров и целостности графовой структуры.

1. Проверка корректности графа при загрузке: при чтении файла с описанием графа (список рёбер с весами) проверяется, что у каждой строки имеется ровно три числовых значения (две вершины и вес ребра). В случае расхождения в формате или наличия недопустимых символов выводится сообщение об ошибке. Проверяется, что указываемые вершины для рёбер действительно являются целочисленными значениями. Если обнаружены некорректные данные, пользователь уведомляется через диалоговое окно об ошибке загрузки, пример чего можно увидеть на рисунке 11.

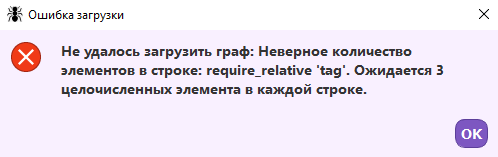


Рисунок 11 – уведомление об ошибке в диалоговом окне при загрузке графа

1. Проверка параметров алгоритма: перед запуском алгоритма муравьиной колонии проверяется корректность введённых параметров: коэффициент испарения феромонов (ρ) должен быть в диапазоне (0; 1). Если значение выходит за пределы, выводится соответствующее сообщение об ошибке, как показано на рисунке 12, и запуск не продолжается. Интенсивность феромонов (Q), веса α и β, а также количество муравьёв и итераций должны быть положительными числами. При нарушении этого условия также генерируется сообщение об ошибке, что показано на рисунке 13.

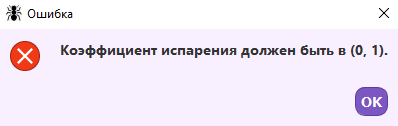


Рисунок 12 – сообщение о некорректности коэффициента испарения

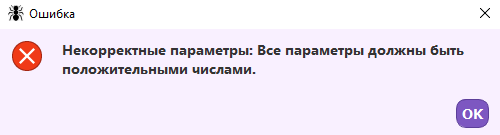


Рисунок 13 – сообщение о некорректных значениях параметров

1. Проверка стартовой и конечных вершин: при выборе стартовой вершины и вводе конечных вершин выполняется проверка существования всех указанных конечных вершин в графе. Если введена несуществующая вершина, выводится ошибка. Проверка на то, что конечные вершины не совпадают со стартовой (в случае, если алгоритм этого не предполагает), а также на отсутствие повторяющихся конечных точек. Проверка, что количество конечных вершин не превышает число вершин графа минус один.
2. Проверка наличия исходных данных для запуска алгоритма: перед тем, как начать выполнять итерации алгоритма, программа убедится: Задан ли граф (загружен ли файл), установлены ли стартовая точка и хотя бы одна конечная точка, корректно ли инициализированы параметры алгоритма.
3. Обработка ситуаций при построении маршрута муравьями: при пошаговом построении маршрутов муравьями проверяется, есть ли доступные вершины для перехода. Если список доступных ходов пуст, маршрут завершается неуспешно. Это предотвращает возникновение бесконечных циклов или попыток перехода к несуществующим вершинам.

# Требуемые модификации

Для решения более сложных задач, таких как оптимизация маршрутов, проходящих через несколько целевых точек, алгоритм требует дополнительных модификаций.

1. Динамическая корректировка вероятности выбора – увеличение вероятности выбора рёбер, ведущих к ещё не посещённым конечным точкам путем включения в формулу (2.1) расчета вероятности выбора следующей точки дополнительный множитель , представленный в формуле (4.1):

(4.1)

где

– узел на данном шаге и следующий соответственно;

– усилитель вероятности, принимающий целочисленные значения;

– множество доступных конечных вершин из узла ;

– множество посещенных конечных вершин.

1. Искусственное добавление феромонов: после каждой итерации добавление небольшого количества феромонов на рёбра, ведущие к конечным точкам, что представлено в формуле (4.2):

(4.2)

где

– уровень феромонов на ребре ;

– узел на данном шаге и следующий соответственно;

– малое значение (например, в пределах от 0.1 до 0.5);

– множество доступных конечных вершин из узла ;

– множество рёбер графа.

1. Динамическое выделение муравьёв: разделение муравьёв на группы с обособленными подзадачами поиска пути через отдельные конечные точки и последующее комбинирование в общий оптимальный маршрут.
2. Предварительная эвристика или комбинирование с жадным алгоритмом: использование минимального остовного дерева для определения порядка обхода точек.

Эти модификации могут позволить алгоритму не только учитывать несколько целевых точек, но и минимизировать длину маршрута, проходящего через них. Дальнейшая работа будет направлена на реализацию и тестирование указанных улучшений, что позволит значительно расширить область применения программы.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была изучена и реализована модель оптимизации доставки однородного груза клиентам с использованием алгоритма муравьиной колонии. Главной целью была разработка базового подхода для нахождения оптимального маршрута и исследовать возможные его модификации для поиска пути, проходящего через одну или несколько заданных точек доставки, с минимальными затратами времени и ресурсов, а также без необходимости возвращения в начальную точку.

Алгоритм муравьиной колонии, являясь эвристическим методом, показал свою эффективность в решении задач оптимизации. Его основная идея заключается в имитации поведения реальных муравьёв, использующих феромонные тропы для поиска кратчайшего пути между муравейником и источником пищи. Такой подход демонстрирует высокую адаптивность и гибкость, что делает его особенно полезным для сложных задач, где пространство решений велико, а точные алгоритмы требуют значительных вычислительных ресурсов.

Результатом работы стало создание программы, позволяющей демонстрировать базовые возможности для работы с алгоритмом муравьиной колонии, а также служащей основой для дальнейших исследований и доработок в направлении оптимизации маршрутов доставки груза с несколькими целевыми точками.

Практическое значение работы заключается в её применимости в создании более усовершенствованных версий алгоритма и приложений, пригодных в навигационных системах и в транспортной логистике.

Следует отметить, что хотя алгоритм муравьиной колонии является приближённым методом, он показал высокий потенциал в решении задач маршрутизации. Однако его эффективность зависит от правильной настройки параметров, таких как коэффициент испарения феромонов, количество муравьёв и весовые коэффициенты эвристики. В рамках работы были подобраны оптимальные значения этих параметров, что обеспечило стабильность и производительность алгоритма.

В заключение, результаты исследования подтверждают, что алгоритм муравьиной колонии может быть эффективно использован для решения задач маршрутизации с заданными точками доставки. В дальнейшем возможна модификация алгоритма для работы с несколькими точками доставки.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Костин, А. С.** Исследование алгоритма маршрутизации для решения задачи коммивояжера на примере алгоритма муравьиной колонии / А. С. Костин, Д. В. Кучко // Системный анализ и логистика. – 2023. – № 4(38). – С. 47-53. – DOI 10.31799/2077-5687-2023-4-47-53. – EDN VTXDDL. — Текст: непосредственный.
2. **Дольников, В. Л.** Теория графов. Алгоритмы на графах : Учеб. пособие : [Для студентов ЯрГУ] / В. Л. Дольников, О. П. Полякова ; В.Л. Дольн­иков, О.П. Полякова; М-во образования Рос. Федерации, Яросл. гос. ун-т им. П.Г. Демидова. – Ярославль : ЯрГУ, 2003. – 115 с. – ISBN 5-8397-0312-5. – EDN QJMPMP. — Текст: непосредственный.
3. **Штовба С. Д.** Муравьиные алгоритмы. Журнал «Математика в приложениях», 2003 г., №4, с. 70-75. — Текст: непосредственный.
4. **Ветрова, О. А.** Использование компьютерной модели "муравьиных" алгоритмов в задачах логистики / О. А. Ветрова, Т. М. Кузьмина, П. С. Калошин // Альманах мировой науки. – 2017. – № 1-1(16). – С. 41-43. – EDN XXDBTF. — Текст: непосредственный.
5. **Штовба, С. Д.** Муравьиные алгоритмы: теория и применение / С. Д. Штовба // Программирование. – 2005. – Т. 31, № 4. – С. 3-18. – EDN HSCKQV. — Текст: непосредственный.
6. **Сазонова, А. С.** Применение муравьиного алгоритма для решения задач коммивояжера, путём создания имитационной модели муравьиной колонии / А. С. Сазонова, А. С. Ашеко // Автоматизация и моделирование в проектировании и управлении : сборник научных статей Всероссийской конференции, Брянск, 22 мая 2023 года. – Курск: ЗАО «Университетская книга», 2023. – С. 7-14. – EDN NIFFBF. — Текст: непосредственный.
7. **Гергедава, Д. А.** Анализ эффективности алгоритма муравьиной колонии в задаче маршрутизации транспортных средств с ограниченной вместимостью / Д. А. Гергедава // Международный журнал информационных технологий и энергоэффективности. – 2024. – Т. 9, № 4(42). – С. 128-137. – EDN KEOUMW. — Текст: непосредственный.
8. **Курейчик, В. В.** Муравьиный алгоритм для решения оптимизационных задач с явно выраженной целевой функцией / В. В. Курейчик, М. А. Жиленков // Информатика, вычислительная техника и инженерное образование. – 2015. – № 2(22). – С. 10-21. – EDN UILTSX. — Текст: непосредственный.
9. **Тухбатуллин, M. C.** Муравьиный алгоритм и его модификации / M. C. Тухбатуллин // Наука: прошлое, настоящее, будущее : Международная научно-практическая конференция, Уфа, 15 августа 2015 года / Ответственный редактор: Сукиасян Асатур Альбертович. Том 1. – Уфа: Общество с ограниченной ответственностью "Аэтерна", 2015. – С. 78-80. – EDN UCYIVT. — Текст: непосредственный.
10. **Ходжамбетов, Ж.** Разработка приложения, реализующего муравьиный алгоритм для нахождения кратчайших путей / Ж. Ходжамбетов, Ж. Е. Сартабанова // Инновационные технологии на транспорте: образование, наука, практика : Материалы XLI Международной научно-практической конференции, Алматы, Казахстан, 03–04 апреля 2017 года / Под редакцией Б.М. Ибраева. Том 1. – Алматы, Казахстан: Казахская академия транспорта и коммуникаций им. М. Тынышпаева, 2017. – С. 81-85. – EDN ZHOHLZ. — Текст: непосредственный.

# ПРИЛОЖЕНИЕ А

**Файл управления моделью**

Файл controller.py

import sys

import time

import networkx as nx

import pyqtgraph as pg

import numpy as np

from app.model.graph\_model import GraphModel

from PyQt5.QtWidgets import QApplication, QMessageBox, QFileDialog

from PyQt5.QtCore import QObject,QTimer

class Controller(QObject):

"""Класс контроллера, координирующий модель и представление."""

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Инициализация модели и представления

self.model = GraphModel(self) # Модель для работы с графом

# Таймер для симуляции работы алгоритма

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time) # Подключаем таймер к обновлению времени

self.elapsed\_time = 0 # Начальное время в миллисекундах

self.running = False # Состояние работы алгоритма

self.start\_time = None # Время старта алгоритма

def show\_error\_message(self, title, message):

"""Показывает окно с ошибкой."""

msg\_box = QMessageBox(self.model.view)

msg\_box.setIcon(QMessageBox.Critical)

msg\_box.setText(message)

msg\_box.setWindowTitle(title)

msg\_box.exec\_()

def run(self):

"""Запуск приложения."""

self.model.set\_view()

self.model.view.show()

def load\_graph(self):

"""Загружает граф из файла и проверяет правильность формата."""

try:

file\_name, \_ = QFileDialog.getOpenFileName(self.model.view, 'Открыть файл графа', '',

'Text Files (\*.txt);;All Files (\*)')

if file\_name:

G = nx.Graph()

with open(file\_name, 'r') as file:

for line in file:

# Попробуем разделить строку на 3 компонента и проверить их на корректность

parts = line.split()

if len(parts) != 3:

raise ValueError(

f"Неверное количество элементов в строке: {line.strip()}. Ожидается 3 целочисленных элемента в каждой строке.")

try:

u, v, weight = map(int, parts)

except ValueError:

raise ValueError(f"Некорректные данные в строке: {line.strip()}. Ожидаются целые числа.")

# Добавляем ребро в граф с весом и феромоном

G.add\_edge(u, v, weight=weight, pheromone=1.0)

# Получаем вершины и сортируем их по возрастанию

sorted\_nodes = sorted(G.nodes)

self.model.graph = G

self.model.start\_node = sorted\_nodes[0] # Пример начальной точки

self.model.positions = nx.spring\_layout(G) # Генерация позиций узлов

self.model.graph\_to\_view(sorted\_nodes)

self.set\_parameters()

# Отображаем статус в статус-баре

self.model.view.status\_bar.showMessage(f"Граф загружен: {file\_name}")

else:

raise ValueError("Не выбран файл для загрузки.")

except Exception as e:

# Поймать любые исключения и вывести сообщение об ошибке

self.model.view.status\_bar.showMessage(f"Ошибка загрузки графа: {e}")

self.show\_error\_message("Ошибка загрузки", f"Не удалось загрузить граф: {str(e)}")

def get\_start\_node(self):

current\_text = self.model.view.start\_node\_combo.currentText()

return int(current\_text) if current\_text else None

def update\_start\_node(self):

"""Обновить начальную точку в модели."""

self.model.start\_node = self.get\_start\_node()

self.model.view.update\_canvas()

def update\_end\_nodes(self):

"""Обновить конечные точки в модели и выполнить проверки.""

# Получаем текст из поля ввода конечных точек

current\_text = self.model.view.end\_nodes\_input.text()

try:

# Преобразуем введенный текст в список целых чисел

end\_nodes = [int(x.strip()) for x in current\_text.split(',') if x.strip().isdigit()]

# Проверка на пустой список

if not end\_nodes:

raise ValueError("Необходимо ввести хотя бы одну конечную точку.")

# Проверка, что все введенные вершины существуют в графе

for node in end\_nodes:

if node not in self.model.graph.nodes:

raise ValueError(f"Вершина {node} не существует в графе.")

# Проверка на выбор всех точек, кроме начальной

total\_nodes = len(self.model.graph.nodes)

if len(end\_nodes) >= total\_nodes:

raise ValueError(f"Можно выбрать не более {total\_nodes - 1} конечных точек.")

# Проверка, что конечные точки не совпадают с начальной

if self.model.start\_node in end\_nodes:

raise ValueError(f"Начальная точка {self.model.start\_node} не может быть конечной.")

# Проверка на повторяющиеся точки

if len(end\_nodes) != len(set(end\_nodes)):

raise ValueError("Конечные точки не могут повторяться.")

# Если все проверки прошли успешно, обновляем конечные точки в модели

self.model.end\_nodes = end\_nodes

self.model.view.update\_canvas() # Обновление графа

except ValueError as e:

# Если возникла ошибка, показываем сообщение в статус-баре

self.model.view.status\_bar.showMessage(f"Ошибка: {str(e)}")

def set\_parameters(self):

"""Обработчик для задания параметров."""

try:

# Получаем параметры из view

evaporation\_rate = float(self.model.view.evaporation\_rate\_input.text())

pheromone\_intensity = float(self.model.view.pheromone\_intensity\_input.text())

alpha = float(self.model.view.alpha\_input.text())

beta = float(self.model.view.beta\_input.text())

num\_ants = int(self.model.view.num\_ants\_input.text())

# Проверка на корректность значений

if evaporation\_rate <= 0 or pheromone\_intensity <= 0 or alpha <= 0 or beta <= 0 or num\_ants <= 0:

raise ValueError("Все параметры должны быть положительными числами.")

# Передаем параметры в модель

self.model.evaporation\_rate = evaporation\_rate

self.model.pheromone\_intensity = pheromone\_intensity

self.model.alpha = alpha

self.model.beta = beta

self.model.num\_ants = num\_ants

# Выводим сообщение в статус-бар

self.model.view.status\_bar.showMessage("Параметры алгоритма успешно заданы.")

except ValueError as e:

# Если возникла ошибка при преобразовании значения, показываем ошибку

self.model.view.status\_bar.showMessage(f"Ошибка: {str(e)}")

self.show\_error\_message("Ошибка", f"Некорректные параметры: {str(e)}")

except Exception as e:

# Для любых других ошибок

self.model.view.status\_bar.showMessage(f"Ошибка: {str(e)}")

self.show\_error\_message("Ошибка", f"Не удалось задать параметры: {str(e)}")

def update\_time(self):

"""Обновление времени каждую миллисекунду с использованием time.time()."""

if self.running and self.start\_time is not None:

current\_time = time.time() # Текущее время в секундах

elapsed\_seconds = current\_time - self.start\_time # Время в секундах с начала старта

minutes = int(elapsed\_seconds // 60)

seconds = int(elapsed\_seconds % 60)

milliseconds = int((elapsed\_seconds \* 1000) % 1000) # Миллисекунды

time\_str = f"{minutes:02}:{seconds:02}:{milliseconds:03}"

self.model.view.time\_label.setText(time\_str)

def start\_algorithm(self):

"""Запуск алгоритма и старта таймера."""

if not self.running:

self.running = True

self.start\_time = time.time() # Фиксируем время старта

self.timer.start(10) # Обновляем каждую десятую миллисекунду

self.model.run\_aco()

self.model.view.status\_bar.showMessage("Алгоритм запущен.")

def stop\_algorithm(self):

if self.running:

self.running = False

self.timer.stop()

self.model.view.status\_bar.showMessage("Алгоритм остановлен.")

def reset\_graph(self):

"""Сброс состояния алгоритма, таймера и начальных/конечных точек."""

self.stop\_algorithm()

self.elapsed\_time = 0

self.model.view.time\_label.setText("00:00:000") # Сбрасываем метку времени

self.model.start\_node = None

self.model.end\_nodes = []

self.model.view.start\_node\_combo.setCurrentIndex(0)

self.model.view.end\_nodes\_input.clear()

self.model.view.status\_bar.showMessage("Работа сброшена.")

self.model.view.update\_canvas()

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

app = QApplication(sys.argv)

controller = Controller()

controller.run()

sys.exit(app.exec\_())

# ПРИЛОЖЕНИЕ Б

**Файл представления графа и интерфейса**

Файл window\_view.py

import pyqtgraph as pg

import networkx as nx  
from PyQt5.QtGui import QIcon  
from PyQt5.QtWidgets import QLabel,QStatusBar,QApplication, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QComboBox, QFileDialog, QLineEdit, QFormLayout, QGroupBox  
from PyQt5.QtCore import QTimer, QSize  
class GraphWindow(QWidget):  
 def \_\_init\_\_(self, model):  
 super().\_\_init\_\_()  
 self.model = model # GraphModel будет присваиваться через Controller  
 self.setWindowTitle('Поиск оптимального маршрута доставки клиентам')  
 self.setGeometry(100, 100, 1150, 800)  
 self.layout = QVBoxLayout(self)  
 # Загрузка стилей из файла  
 self.load\_styles("styles.qss")  
 icon = QIcon()  
 icon.addFile('ant\_picture.png', QSize(64, 64))  
 self.setWindowIcon(icon)  
 # Таймер отображения  
 self.time\_label = QLabel("00:00:000", self)  
 self.layout.addWidget(self.time\_label)  
 # Создаем виджет для графика с использованием pyqtgraph  
 self.plot\_widget = pg.PlotWidget()  
 self.layout.addWidget(self.plot\_widget)  
 self.plot\_widget.setBackground('w')  
 # Создаем статус-бар  
 self.status\_bar = QStatusBar(self)  
 self.layout.addWidget(self.status\_bar)  
 # Инициализация элементов интерфейса  
 self.create\_controls()  
 self.create\_buttons()  
 def load\_styles(self, style\_file):  
 """Загружает стили из указанного файла."""  
 with open(style\_file, "r") as file:  
 self.setStyleSheet(file.read())  
 def create\_controls(self):  
 controls\_layout = QFormLayout()  
 # Начальная точка  
 self.start\_node\_combo = QComboBox(self)  
 controls\_layout.addRow("Начальная точка:", self.start\_node\_combo)  
 # Конечные точки и кнопка "Ок"  
 self.end\_nodes\_input = QLineEdit(self)  
 self.end\_nodes\_input.setPlaceholderText("Введите конечные точки через запятую")  
 self.set\_end\_nodes\_button = QPushButton("Ок", self)  
 self.set\_end\_nodes\_button.clicked.connect(self.model.controller.update\_end\_nodes)  
 # Мы используем QHBoxLayout, чтобы разместить кнопку рядом с полем ввода  
 end\_nodes\_layout = QHBoxLayout()  
 end\_nodes\_layout.addWidget(self.end\_nodes\_input)  
 end\_nodes\_layout.addWidget(self.set\_end\_nodes\_button)  
 controls\_layout.addRow("Конечные точки:", end\_nodes\_layout)  
 # Параметры алгоритма  
 self.evaporation\_rate\_input = QLineEdit("1.3", self)  
 self.pheromone\_intensity\_input = QLineEdit("1.0", self)  
 self.alpha\_input = QLineEdit("2", self)  
 self.beta\_input = QLineEdit("1", self)  
 self.num\_ants\_input = QLineEdit("10", self)  
 controls\_layout.addRow("Коэффициент испарения:", self.evaporation\_rate\_input)  
 controls\_layout.addRow("Интенсивность феромонов:", self.pheromone\_intensity\_input)  
 controls\_layout.addRow("Вес феромонов (alpha):", self.alpha\_input)  
 controls\_layout.addRow("Вес расстояния (beta):", self.beta\_input)  
 controls\_layout.addRow("Количество муравьёв:", self.num\_ants\_input)  
 controls\_group = QGroupBox("Настройки алгоритма")  
 controls\_group.setLayout(controls\_layout)  
 self.layout.addWidget(controls\_group)  
 def create\_buttons(self):  
 buttons\_layout = QHBoxLayout()  
 self.load\_button = QPushButton('Загрузить граф')  
 buttons\_layout.addWidget(self.load\_button)  
 self.set\_param\_button = QPushButton('Задать параметры')  
 buttons\_layout.addWidget(self.set\_param\_button)  
  
 self.start\_button = QPushButton('Запуск')  
 buttons\_layout.addWidget(self.start\_button)  
 self.stop\_button = QPushButton('Остановка')  
 buttons\_layout.addWidget(self.stop\_button)  
 self.reset\_button = QPushButton('Сброс')  
 buttons\_layout.addWidget(self.reset\_button)  
 self.layout.addLayout(buttons\_layout)  
 def update\_start\_node\_combo(self, nodes):  
 """Обновляет список доступных начальных точек."""  
 self.start\_node\_combo.clear()  
 for node in nodes:  
 self.start\_node\_combo.addItem(str(node))  
 def update\_canvas(self):  
 """Очищает и перерисовывает граф на виджете pyqtgraph."""  
 self.plot\_widget.clear()  
 pos = self.model.positions # Получаем позиции из модели  
 edges = self.model.graph.edges(data=True)  
 # Отрисовка рёбер с учетом феромонов  
 for u, v, data in edges:  
 pheromone = data['pheromone']  
 weight = data['weight']  
 line = pg.PlotCurveItem(  
 [pos[u][0], pos[v][0]], [pos[u][1], pos[v][1]],  
 pen=pg.mkPen(color=(200, 200, 200), width=pheromone \* 5) # Толщина линии пропорциональна феромонам  
 )  
 self.plot\_widget.addItem(line)  
 # Добавляем текст для расстояния  
 mid\_x = (pos[u][0] + pos[v][0]) / 2  
 mid\_y = (pos[u][1] + pos[v][1]) / 2  
 text\_item = pg.TextItem(f"{weight}", anchor=(0.5, 0.5), color='black')  
 text\_item.setFont(pg.QtGui.QFont("Arial", 10))  
 text\_item.setPos(mid\_x, mid\_y)  
 self.plot\_widget.addItem(text\_item)  
 # Отрисовка узлов  
 for node in self.model.graph.nodes:  
 x, y = pos[node]  
 color = '#eb5757' # Красный для всех узлов, кроме стартового и конечного  
 if node == self.model.start\_node:  
 color = '#6fcf97' # Зеленый для стартового узла  
 elif node in self.model.end\_nodes: # Проверяем, является ли узел конечной точкой  
 color = '#f2994a'  
 scatter = pg.ScatterPlotItem([x], [y], symbol='o', size=30, brush=color)  
 self.plot\_widget.addItem(scatter)  
 # Текст внутри узлов  
 text = pg.TextItem(str(node), anchor=(0.5, 0.5), color='white')  
 text.setFont(pg.QtGui.QFont("Arial", 14, pg.QtGui.QFont.Bold))  
 text.setPos(x, y)  
 self.plot\_widget.addItem(text)

# ПРИЛОЖЕНИЕ В

**Файл модели графа с алгоритмом**

Файл graph\_model.py

import networkx as nx

import pyqtgraph as pg

import random

import math

from PyQt5.QtWidgets import QApplication

from app.view.window\_view import GraphWindow

class GraphModel:

def \_\_init\_\_(self, controller):

self.graph = None

self.positions = {}

self.start\_node = None

self.end\_nodes = set()

self.controller = controller

self.view = None

self.pheromone\_intensity = None

self.evaporation\_rate = None

self.alpha = None

self.beta = None

self.num\_ants = None

self.num\_iterations = 20

self.best\_path = None

self.best\_length = math.inf

self.min\_pheromone = 0.1

def setup\_connections(self):

"""Связывает сигналы интерфейса с действиями контроллера."""

self.view.start\_node\_combo.currentIndexChanged.connect(self.controller.update\_start\_node)

self.view.set\_end\_nodes\_button.clicked.connect(self.controller.update\_end\_nodes)

self.view.set\_param\_button.clicked.connect(self.controller.set\_parameters)

self.view.load\_button.clicked.connect(self.controller.load\_graph)

self.view.start\_button.clicked.connect(self.controller.start\_algorithm)

self.view.stop\_button.clicked.connect(self.controller.stop\_algorithm)

self.view.reset\_button.clicked.connect(self.controller.reset\_graph)

def set\_view(self):

self.view = GraphWindow(self)

self.setup\_connections()

return self.view

def graph\_to\_view(self, nodes):

self.view.update\_start\_node\_combo(nodes)

self.view.update\_canvas()

def initialize\_pheromones(self):

"""Инициализация феромонов для всех рёбер графа."""

for u, v in self.graph.edges:

self.graph[u][v]['pheromone'] = 1.0

def calculate\_path\_length(self, path):

"""Вычисляет длину пути."""

return sum(self.graph[u][v]['weight'] for u, v in zip(path, path[1:]))

def generate\_ant\_path(self):

"""Генерация пути для одного муравья."""

current\_node = self.start\_node

path = [current\_node]

visited\_nodes = {current\_node}

end\_node = next(iter(self.end\_nodes)) # Предполагаем, что пока только одна конечная вершина

while current\_node != end\_node:

neighbors = list(self.graph.neighbors(current\_node))

probabilities = self.calculate\_probabilities(current\_node, neighbors, visited\_nodes)

if not probabilities: # Если нет доступных узлов, тупик

return path, float('inf')

next\_node = random.choices(neighbors, weights=probabilities)[0]

path.append(next\_node)

visited\_nodes.add(next\_node)

current\_node = next\_node

path\_length = self.calculate\_path\_length(path)

return path, path\_length

def calculate\_probabilities(self, current\_node, neighbors, visited\_nodes):

"""Рассчитывает вероятности перехода к соседним узлам."""

probabilities = []

for neighbor in neighbors:

if neighbor in visited\_nodes:

probabilities.append(0)

else:

pheromone = self.graph[current\_node][neighbor]['pheromone']

distance = self.graph[current\_node][neighbor]['weight']

probabilities.append((pheromone \*\* self.alpha) \* ((1 / distance) \*\* self.beta))

total = sum(probabilities)

if total == 0:

return []

return [p / total for p in probabilities]

def update\_pheromones(self, all\_paths):

"""Обновляет феромоны на основе пройденных путей."""

# Шаг 1: Испарение феромонов

for u, v in self.graph.edges():

self.graph[u][v]['pheromone'] \*= (1 - self.evaporation\_rate)

if self.graph[u][v]['pheromone'] < self.min\_pheromone:

self.graph[u][v]['pheromone'] = self.min\_pheromone

# Шаг 2: Обновление феромонов

for path, path\_length in all\_paths:

if path\_length == float('inf'):

continue

pheromone\_contribution = self.pheromone\_intensity / path\_length

for u, v in zip(path, path[1:]):

self.graph[u][v]['pheromone'] += pheromone\_contribution

def run\_aco(self):

"""Запуск муравьиного алгоритма."""

if self.graph is None or self.start\_node is None or not self.end\_nodes:

self.controller.show\_error\_message("Ошибка", "Граф, начальная или конечная вершина не заданы.")

return

self.initialize\_pheromones()

for iteration in range(self.num\_iterations):

all\_paths = []

for \_ in range(self.num\_ants):

path, path\_length = self.generate\_ant\_path()

all\_paths.append((path, path\_length))

# Определение лучшего пути в этой итерации

for path, path\_length in all\_paths:

if path\_length < self.best\_length:

self.best\_length = path\_length

self.best\_path = path

# Обновление феромонов

self.update\_pheromones(all\_paths)

# Визуализация процесса

self.view.update\_canvas()

QApplication.processEvents()

# Итоговая визуализация лучшего пути

if self.best\_path:

for u, v in zip(self.best\_path, self.best\_path[1:]):

if self.graph.has\_edge(u, v):

self.graph[u][v]['pheromone'] += 2 \* self.pheromone\_intensity / self.best\_length

self.view.update\_canvas()

self.controller.model.view.status\_bar.showMessage(

f"Лучший путь: {self.best\_path}, длина: {self.best\_length}")