МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка алгоритма поиска оптимального маршрута для доставки грузов»

Обучающийся Е.И. Шарков
(Инициалы Фамилия) (личная подпись)

Руководитель доктор техн. наук, доцент, С.В. Мкртычев
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. пед. наук, доцент, С.А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Тема бакалаврской работы – «Разработка алгоритма поиска оптимального маршрута для доставки грузов».

Объект исследования – задачи оптимизации маршрута для доставки грузов.

Предмет исследования – алгоритм поиска оптимального маршрута для доставки грузов.

Цель данной работы заключается в разработке алгоритма поиска оптимального маршрута для доставки грузов.

Бакалаврская работа состоит из введения, трёх разделов, заключения и списка литературы.

В первом разделе описаны методы для поиска оптимального маршрута.

Во втором разделе описаны алгоритмы для поиска оптимального маршрута.

В третьем разделе описывается разработанное программное обеспечение и производится его тестирование.

В заключении представлены выводы по проделанной работе.

В работе использованы 22 рисунка и 3 таблицы, список литературы содержит 20 источников.

Общий объем выпускной квалификационной работы составляет 41 страницы.

Abstract

The topic of the bachelor's thesis is "Development of an algorithm for finding the optimal route for cargo delivery".

The object of the study is the task of optimizing the route for cargo delivery.

The subject of the study is an algorithm for finding the optimal route for cargo delivery.

The purpose of this work is to develop an algorithm for finding the optimal route for cargo delivery.

The bachelor's thesis consists of an introduction, three sections, a conclusion and a list of references.

The first section describes the methods for finding the optimal route.

The second section describes algorithms for finding the optimal route.

The third section describes the developed software and tests it.

In conclusion, the conclusions on the work done are presented.

The work uses 22 illustrations and 3 tables, the list of references contains 20 sources. The total volume of the final qualifying work is 41 pages.

Содержание

Введение 5
1 Постановка задачи исследования и анализ методов поиска оптимального маршрута для доставки грузов 6
1.1 Постановка задачи исследования 6
1.2 Методы поиска оптимального маршрута для доставки грузов 7
1.2.1 Классические методы оптимизации маршрутов 7
1.2.2 Эвристические методы оптимизации маршрутов 9
1.3 Сравнительный анализ методов поиска оптимального 11
маршрута для доставки грузов 11
2 Обзор и анализ алгоритмов поиска оптимального маршрута для доставки грузов 15
2.1 Классические алгоритмы поиска оптимального маршрута 15
2.1.1 Алгоритм ветвей и границ 15
2.1.2 Алгоритм Дейкстры 16
2.2 Эвристические алгоритмы 17
2.2.1 Алгоритм ближайшего соседа 17
2.2.2 Алгоритмы локального поиска 18
2.2.3 Алгоритм имитации отжига 19
2.2.4 Генетический алгоритм 20
2.2.5 Алгоритм муравьиной колонии 21
2.3 Сравнительный анализ алгоритмов поиска оптимального маршрута для доставки грузов 23
3 Программная реализация и тестирование алгоритма поиска оптимального маршрута для доставки грузов 26
3.1 Выбор языка программирования и среды разработка 26
3.2 Разработка программного обеспечения 26
3.3 Тестирование программного обеспечения 35
Заключение 38
Список используемой литературы и источников 39

Введение

Современный мир характеризуется высокими объемами внутренней и международной торговли, что требует эффективного управления логистическими процессами для обеспечения бесперебойного и своевременного перемещения товаров. В текущих условиях оптимизация маршрутов доставки грузов становится критически важной задачей. Разработка эффективного алгоритма для поиска оптимального маршрута актуальна по многим причинам, например, увеличение объема грузоперевозок. В условиях постоянно увеличивающегося объема грузов, эффективное управление маршрутами необходимо для обеспечения конкурентоспособности компаний и удовлетворения потребностей клиентов. Также оптимизация маршрутов позволяет сократить затраты на транспортировку, включая топливные расходы, амортизацию транспортных средств и рабочее время водителя.

Объект исследования – задачи оптимизации маршрута для доставки грузов.

Предмет исследования – алгоритм поиска оптимального маршрута для доставки грузов.

Цель данной работы заключается в разработке алгоритма поиска оптимального маршрута для доставки грузов.

В рамках данной работы будут выполнены следующие задачи:

- постановка задачи исследования и произведен анализ методов поиска оптимального маршрута для доставки грузов;
- обзор и анализ алгоритмов поиска оптимального маршрута для доставки грузов;
- программная реализация и тестирование алгоритма поиска оптимального маршрута для доставки грузов.

Результатом проделанной работы будет разработанный алгоритм поиска оптимального маршрута для доставки грузов и программное обеспечение.

1 Постановка задачи исследования и анализ методов поиска оптимального маршрута для доставки грузов

1.1 Постановка задачи исследования

Оптимальный маршрут для доставки груза – это такой маршрут, который обеспечивает наиболее эффективное перемещение груза из одной точки в другую. Критерии оптимальности маршрута могут быть разными, ниже приведены некоторые из них [4], [11]:

- путь, пройденный транспортом, должен быть минимальным;
- стоимость перевозки единицы груза должна быть минимально возможной;
- временные затраты на прохождение пути должны быть минимально возможными.

Математически данную задачу можно представить так (1):

$$G = (V, E), \tag{1}$$

где G – это граф;

V – множество вершин(городов);

E – множество ребер (дорог между городами).

Каждому ребру $e \in E$ сопоставлены два веса: $d(e)$ – длинна маршрута, $c(e)$- стоимость маршрута.

Задача состоит в том, чтобы найти такой маршрут P, который минимизирует функцию стоимости (2):

$$f(P) = w_1 \sum_{e \in P} d(e) + w_2 \sum_{e \in P} c(e) \to min, \tag{2}$$

где w_1 и w_2 – это веса, которые определяют относительную важность длинны и стоимости маршрута соответственно.

На рисунке 1 представлены данные необходимые для решения задачи.

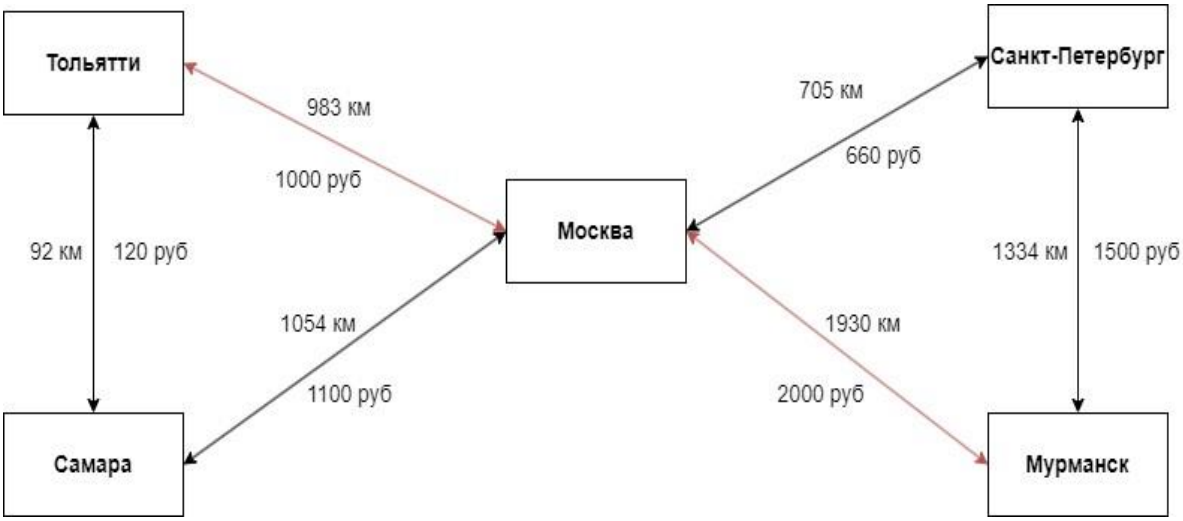

Рисунок 1 – Графическое представление данных задачи

В данном случае оптимальным маршрутом из Тольятти в Мурманск будет являться маршрут Тольятти-Москва-Мурманск, который отмечен на рисунке красным цветом.

1.2 Методы поиска оптимального маршрута для доставки грузов

1.2.1 Классические методы оптимизации маршрутов

Классические методы оптимизации маршрутов включают в себя такие подходы как линейное и динамическое программирование. Эти методы представляют собой формализованные способы нахождения оптимальных решений для задач, связанных с транспортировкой и логистикой. Рассмотрим эти методы подробнее.

Линейное программирование – математический метод оптимизации, который используется для нахождения оптимального решения задачи при наличии линейной целевой функции и линейных ограничений [10, с. 15]. Этот метод можно применить для минимизации совокупной протяженности

маршрута и его стоимости.

Задачу линейного программирования можно описать минимизацию линейной функции (3):

$$\sum_{i,j}\left(w_1 * d_{ij} + w_2 * c_{ij}\right) * x_{ij} \rightarrow min\,, \tag{3}$$

при условиях (4):

$$\begin{cases} \sum_{j} x_{ij} = 1 \ \forall i \\ \sum_{i} x_{ij} = 1 \ \forall j\,, \\ x_{ij} \in \{0,1\} \end{cases} \tag{4}$$

где d_{ij}– длина маршрута между городами i и j;

c_{ij} – стоимость маршрута между городами i и j;

w_1 и w_2 – относительные веса для длинны и стоимости маршрута соответственно;

x_{ij}– бинарная переменная, которая равна единице, если маршрут включен в путь, и нулю в противном случае.

Линейное программирование позволяет эффективно оптимизировать маршрут, но может потребовать много ресурсов при большом количестве городов и критериев оптимальности.

Динамическое программирование – метод решения сложных задач путем разбиения из на более простые подзадачи и хранения результатов этих подзадач для предотвращения повторных вычислений. Этот метод основан на принципе оптимальности Беллмана: любое оптимальное решение задачи включает оптимальные решения ее подзадач. Задача разбивается на подзадачи, которые решаются независимо, и их результаты используются для

построения решения всей задачи.

Поставленную задачу можно решить с помощью динамического программирования путем разбиения на подзадачи нахождения кратчайших и наименее затратных путей между отдельными парами городов, а затем объединения полученных результатов в единое решение [14].

1.2.2 Эвристические методы оптимизации маршрутов

Эвристические методы представляют собой альтернативные подходы к решению задачи оптимизации маршрутов. Эти методы используются для нахождения качественных решений в сложных и динамических условиях, где точные методы могут быть слишком затратными по времени и ресурсам. К таким методам относят жадные алгоритмы, методы локального поиска, метаэвристики. Рассмотрим подробнее каждый из этих методов.

Жадные алгоритмы основываются на принятии локально оптимальных решений на каждом шаге, надеясь, что это приведет к глобально оптимальному решению [19]. Такие алгоритмы просты в реализации и часто используются для начального построения маршрутов. Основная идея жадных алгоритмов заключается в выборе наилучшего на текущий момент варианта, не учитывая глобальные последствия [6]. К преимуществам таких алгоритмов можно отнести простоту реализации и высокую скорость работы с небольшими задачами. В то же время, они имеют ряд недостатков, так как они могут застревать в локальных оптимумах и отсутствует гарантия нахождения глобально оптимального решения.

Методы локального поиска начинают с какого-либо начального решения и пытаются улучшить его путем локальных изменений. Эти методы позволяют находить хорошие решения за разумное время. Основная идея заключается в последовательном улучшении текущего решения до тех пор, пока не будет достигнут локальный оптимум. Данный метод прост в реализации, но качество итогового решения зависит от начальной точки и

стратегии поиска.

Следующей ступенью в развитии эвристических методов можно назвать метаэвристические методы. Они представляют собой высокоуровневые процедуры, направленные на управление эвристиками для получения качественных решений сложных задач оптимизации. К наиболее известным метаэвристикам относятся генетические и муравьиные алгоритмы, а также имитация отжига.

Генетический алгоритм – метод оптимизации, который основывается на принципах естественного отбора и генетики. Путем моделирования процесса естественного отбора, метод способен находить решение сложных задач [8]. Он хорошо подходит для задач, где пространство поиска велико и имеет множество локальных оптимумов. Основная идея данного метода заключается в создании и поддержании популяции возможных решений, оценке приспособленности каждого индивидуума и использовании скрещивания и мутации для создания нового поколения решений [7]. Таким образом происходит постепенное улучшение популяции, а следовательно подбор оптимального решения. Схема работы показана на рисунке 2.

Рисунок 2 – Схема работы генетического алгоритма

Метод имитации отжига основан на физическом процессе отжига металлов. Основная суть метода состоит в том, что процесс начинается с начального решения и высокой «температуры», на каждом шаге текущему решению предлагается случайное изменение и создается новое решение. Если новое решение лучше текущего, оно принимается, в обратном случае оно может быть принято с определенной вероятностью, зависящей от разницы в значениях решений и текущей температуры [19]. Температура постепенно снижается, тем самым уменьшая вероятность принятия ухудшающих решений. Этот метод хорошо показывает себя при решении сложных задач с большим количество исходных данных и множеством локальных минимумов.

1.3 Сравнительный анализ методов поиска оптимального маршрута для доставки грузов

Для проведения сравнительного анализа всех вышеприведенных методов можно использовать следующие критерии:

- точность решений – способность метода находить оптимальное или близкое к нему решение;
- время выполнения – время, которое необходимо для нахождения решения;
- затраты вычислительных ресурсов – количество затраченных вычислительных ресурсов, необходимых для поиска решения. Например, процессорное время или память;
- гибкость и адаптивность – способность метода адаптироваться к меняющимся условиям и требованиям задачи;
- сложность реализации – сложность реализации и настройки метода на практике;
- гарантия нахождения глобального оптимума – способность гарантировать нахождение глобально оптимального решения.

Оценка методов по указанным критериям приведена в таблице 1.

Таблица 1 – Оценка методов поиска оптимального маршрута

Критерий	Методы поиска оптимального маршрута					
	Линейное программирование	Динамическое программирование	Жадные алгоритмы	Методы локального поиска	Генетические алгоритмы	Имитация отжига
Точность решений	Высокая	Высокая	Низкая	Средняя	Средняя	Средняя
Время выполнения	Большое для сложных задач	Большое для сложных задач	Маленькое	Среднее	Большое	Большое
Затраты вычислительных ресурсов	Большие	Большие	Маленькие	Маленькие	Большие	Большие
Гибкость и адаптивность	Низкая	Средняя	Высокая	Высокая	Высокая	Высокая
Сложность реализации	Высокая	Высокая	Низкая	Низкая	Средняя	Средняя
Гарантия нахождения глобального оптимума	Есть	Есть	Нет	Нет	Нет	Нет

Так же для проведения адекватного сравнительного анализ анализа рассмотрим основные преимущества и недостатки каждого метода (таблица 2).

Таблица 2 – Преимущества и недостатки методов

Метод	Преимущества	Недостатки
Линейное программирование	Широкое применение: используется в разнообразных отраслях, таких как транспорт, экономика и логистика. Хорошая теоретическая база: существуют многочисленные исследования и документация.	Сложность моделирования: требует детального понимания задачи для корректного моделирования. Ограниченность в применении: неприменимо к нелинейным задачам или задачам с дискретными переменными.

Продолжение таблицы 2

Динамическое программирование	Решение широкого круга задач: подходит для задач с перекрывающимися подзадачами, таких как оптимизация маршрутов, планирование и управление запасами.	Трудности в разбиении задач: требует тщательного разбиения задачи на подзадачи и определения рекурсивных соотношений. Сложность в отладке: Ошибки могут быть трудно обнаружимы из-за рекурсивной природы метода.
Жадные алгоритмы	Легкость объяснения и понимания: простейшие алгоритмы легко понять и объяснить даже неспециалистам. Быстрая адаптация к изменяющимся данным: можно быстро пересчитывать решения при небольших изменениях данных.	Местные решения могут не совпадать с глобальными: выбор на каждом шаге может не привести к глобально оптимальному решению. Ограниченное применение: эффективны только для определенных типов задач, где оптимальное решение может быть достигнуто жадным образом.
Методы локального поиска	Многообразие вариантов: существуют различные методы локального поиска, такие как табу-поиск, что позволяет выбирать наиболее подходящий для конкретной задачи. Хорошо работают с реальными данными: Эффективны для задач, где точная модель недоступна.	Возможность застревания в локальных минимумах: требует дополнительных механизмов для выхода из локальных минимумов. Параметризация: результаты сильно зависят от выбора начальных параметров и эвристик.
Генетические алгоритмы	Природное моделирование: основаны на естественных процессах эволюции, что позволяет эффективно решать сложные задачи. Параллельная обработка: возможность параллельного выполнения улучшает производительность.	Сложность настройки: требует настройки множества параметров, таких как размер популяции, вероятность мутации и кроссовера. Эффективность сильно зависит от проблемной области: может не быть эффективным для всех типов задач.
Имитация отжига	Интуитивное обоснование: основан на физическом процессе отжига, что делает его понятным и логически обоснованным. Хорошо работает для задач с многими локальными минимумами: способен находить хорошие приближенные решения.	Сложность выбора параметров: результаты сильно зависят от выбора начальной температуры и скорости охлаждения. Время выполнения: может требовать значительного времени для сходимости к хорошему решению.

Можно сделать вывод, что все методы поиска оптимального маршрута имеют свои сильные и слабые стороны и их эффективность зависит от конкретных условий задачи.

Выводы по разделу 1

Первый раздел посвящен постановке задачи и обзору методов поиска оптимального маршрута для доставки грузов. Были разобраны разнообразные классические и эвристические методы, а также проведен их сравнительный анализ.

Классические методы обеспечивают наилучшую точность, но могут быть неэффективными для больших задач и не могут адаптироваться к изменяющимся условиям. В то же время, эвристические методы гибки и адаптивны, но могут иметь более низкую точность.

В итоге, можно сказать, что нужно выбирать метод или комбинацию методов в зависимости от требований. Гибридные подходы, которые сочетают преимущества разных методов могут предоставить наилучшие результаты.

2 Обзор и анализ алгоритмов поиска оптимального маршрута для доставки грузов

2.1 Классические алгоритмы поиска оптимального маршрута

2.1.1 Алгоритм ветвей и границ

Алгоритм ветвей и границ – мощный инструмент для решения комбинаторных задач, в частности и для нахождения оптимального маршрута для доставки грузов.

Принцип работы данного алгоритма заключается в разветвлении и ограничении. Сначала алгоритм начинает с полного множества возможных маршрутов и рекурсивно делит его на подмножества(ветви), тем самым создавая дерево решений. Например, все возможные маршруты между населенными пунктами. Далее для каждого подмножества вычисляется нижняя граница (минимальная возможная длина и стоимость маршрута). Если нижняя граница текущей ветви хуже, чем уже известное оптимальное решение, эта ветвь отсекается (ограничивается) [6], [20].

Пошаговое описание алгоритма представлено ниже.

Пусть x – вектор маршрутов, $f(x)$ – целевая функция, включающая критерии длинны $d(x)$ и стоимости $c(x)$.

Шаг 1: начинаем с начального множества маршрутов S_0.

Шаг 2: на каждом шаге выберем подмножество S для исследования.

Шаг 3: делим на S подмножества $S_1, S_2, \dots, S_k$.

Шаг 4: для каждого подмножества S_i вычисляем нижние границы $B_d(S_i)$ и $B_c(S_i)$ для длины и стоимости соответственно.

Шаг 5: если $B_d(S_i) \geq d(x^*)$ и $B_c(S_i) \geq d(x^*)$ для текущего лучшего решения x^*, то подмножество S_i отсекается, иначе оно добавляется в очередь для дальнейшего исследования.

2.1.2 Алгоритм Дейкстры

Алгоритм Дейкстры – это популярный алгоритм, который находит кратчайшие пути в графах с неотрицательными весами ребер [1]-[3], [13]. Его можно применить и для оптимизации маршрутов для доставки грузов, особенно когда ключевую роль играет длинна пути. В данном случае можно модифицировать алгоритм для работы с двумя критериями: длина и стоимость маршрута.

Ниже разберем работу алгоритма по шагам.

Шаг 1: производится инициализация. Пусть $G = (\mathrm{V}, \mathrm{E})$ – граф, где V – множество вершин(городов), а E – множество ребер (маршрутов между городами с заданной длинной и стоимости). Выбираем исходную вершину S (пункт отправления). Создаем два массива: $d[v]$ для хранения минимальных расстояний и $c[v]$ для хранения минимальных стоимостей от S до v. Инициализируем массивы (5):

$$d[v] = \begin{cases} 0, & \text{если } v = S \\ \infty & \text{иначе} \end{cases}, \qquad \mathrm{c}[v] = \begin{cases} 0, & \text{если } v = S \\ \infty & \text{иначе} \end{cases}. \tag{5}$$

Создаем приоритетную очередь и помещает туда начальную вершину S с приоритетом 0.

Шаг 2: основной цикл работы алгоритма. Пока приоритетная очередь не пуста извлекаем вершину u с минимальным $d[u]$ и для каждой соседней вершины v этой вершины применяем условие: если $d[u] + \text{длинна}\,(u, v) < d[v]$, то $d[v] = d[u] + \text{длина}(u, v)$. Так же для нового пути с меньшей стоимостью: если $c[u] + \text{стоимость}\,(u, v) < c[v]$, то $c[v] = c[u]\,\text{стоимость}(u, v)$. Далее приоритетная очередь для вершины v обновляется.

Шаг 3: завершение работы алгоритма. Когда приоритетная очередь пуста, массивы d и c содержат кратчайшие расстояния и минимальные

стоимости от вершины S до всех остальных вершин.

Применение данного алгоритма может помочь найти оптимальные маршруты для доставки грузов.

2.2 Эвристические алгоритмы

2.2.1 Алгоритм ближайшего соседа

Алгоритм ближайшего соседа – это жадный алгоритм, который последовательно выбирает следующий город, являющийся ближайшим непосещенным городом, тем самым строя маршрут по принципу «ближайшего соседа». Он часто используется для решения задачи коммивояжера и может быть адаптирован для поиска оптимального маршрута для доставки грузов. Данный алгоритм принимает локально оптимальные решение на каждом шаге с целью нахождения глобально оптимального решения.

Пример пошаговой работы алгоритма ближайшего соседа показан ниже.

Шаг 1: выбираем начальный город $v_o \in V$. Пусть T – текущий маршрут, а S– множество посещенных городов. $T = [v_o], S = \{v_o\}$.

Шаг 2: пока $|S| < |V|$, находим ближайший непосещенный город v_{next} (6):

$$v_{next} = \arg\min(w_1 * d(v_{current}, v) + w_2 * c(u_{current}, v)), \quad (6)$$

где $d(v_{current}, v)$ – расстояние между текущим и следующим городом;

$c(u_{current}, v)$ – стоимость перемещения между ними;

w_1 и w_2 – веса, соответствующие важности расстояния и стоимости соответственно.

Добавляем v_{next} в маршрут T и во множество посещенных городов S и обновляем текущий город.

Шаг 3: добавляем начальный город в конец маршрута T для замыкания

цикла.

Жадные алгоритмы часто используются в комбинации с более сложными алгоритмами в качестве отправной точки, так как могут помочь улучшить общее качество найденного решения.

2.2.2 Алгоритмы локального поиска

Алгоритмы локального поиска – это класс эвристических алгоритмов, которые начинаю с некоторого начального решения и постепенно улучшают его, переходя к соседним решениям, пока не будет найдено решение, не имеющее улучшений в своем непосредственном окружении. Часто применяются для решения задач оптимизации, где требуется найти максимально или минимально оптимальное значение [16].

Алгоритм может быть использован для решения задачи поиска оптимального маршрута. Основная идея состоит в выборе какого-либо начального маршрута и постепенном его улучшении, путем замены его частей на более выгодные, пока не будет достигнут локальный оптимум.

Ниже разберем пошаговую работу алгоритма.

Шаг 1: инициализация. Начинаем с произвольного маршрута P_0, создаем текущее лучшее решение $P_{best} = P_0$ и определяем общую функцию стоимости маршрута $f(P) = w_1 * d(P) + w_2 * c(P)$, которая комбинирует его длину и стоимость.

Шаг 2: поиска соседних решений. Определяем множество соседних решений $N(P_{current})$. Соседние решения можно создавать различными способами, например, перестановкой двух городов местами, обратным упорядочиванием части маршрута или перемещением города в другое место.

Шаг 3: выбор лучшего соседнего решения. Для каждого соседнего решения $P' \in N(P_{current})$ вычисляется стоимость $f(P')$ и находится лучшее соседнее решение с минимальной стоимостью $P_{best\_neighbor} = \arg\min f(P')$.

Шаг 4: обновление текущего решения. Если $f(P_{best\_neighbor}) <$

$f(P_{current})$, то $P_{current} = P_{best\_neighbor}$. Так же если $f(P_{current}) < f(P_{best})$, то $P_{best} = P_{current}$. Иначе, если ни одно соседнее решение не улучшает текущее, то алгоритм завершает свою работу.

Шаг 5: завершение. Возвращается лучшее найденное решение P_{best}.

Алгоритм локального поиска может помочь значительно сократить время поиска приемлемого решения по сравнению с переборочными методами.

2.2.3 Алгоритм имитации отжига

Алгоритм имитации отжига – это стохастический метод оптимизации, который использует аналогию с физическим процессом отжига металлов, где материал постепенно охлаждается для достижения стабильной структуры. Данный алгоритм позволяет временно принимать ухудшающие решения для избегания локальных минимумов и поиска глобального оптимума. Так же этот алгоритм может быть использован для решения задачи оптимизации маршрутов.

Ниже опишем процесс работы алгоритма.

Шаг 1: инициализация. Создается начальное решение P_0, который представляет собой случайный маршрут, включающий все заданные города. Устанавливается текущее решение $P_{current} = P_0$ лучшее решение $P_{best} = P_0$. Так же устанавливается начальная температура T_0, минимальная температура T_{min} и коэффициент охлаждения a. Температура контролирует вероятность принятия ухудшающих решений и постепенно уменьшается в ходе работы алгоритма

Шаг 2: основной цикл. Работа алгоритма продолжается пока не будет достигнута минимальная температура. Создается новое состояние P_{new} путем небольшого изменения текущего маршрута $P_{current}$. Далее вычисляется изменение стоимости маршрута $\Delta E = f(P_{new}) - f(P_{current})$, если $\Delta E < 0$ – принимаем новое решение $P_{current} = P_{new}$, так как оно улучшает текущее

состояние. Если $\Delta E > 0$, принимаем P_{new} с вероятностью $e^{-\Delta E/T}$. После обновления текущего решения обновляется температура $T \leftarrow a *$.

Благодаря своей способности избегать локальных оптимумов данный алгоритм особенно полезен для сложных задач, где другие алгоритмы могут оказаться менее эффективными.

2.2.4 Генетический алгоритм

В основе принципов генетического алгоритма лежит принцип естественного отбора и эволюции. Он позволяет находить оптимальные решения в задачах с большим пространством поиска, что делает его крайне полезным в решении поставленной задачи [18].

Схема работы генетического алгоритма представлена ниже.

Шаг 1: создается начальная популяция маршрутов(хромосом), где каждая хромосома представляет собой возможный маршрут, включающий все заданные города. Определяется количество маршрутов в популяции.

Шаг 2: для каждого маршрута проводится оценка его приспособленности, которая зависит от длины и стоимости маршрута, по формуле $f(P) = w_1 * d(P) + w_2 * c(P)$.

Шаг 3: начинается процесс селекции. На основе функции приспособленности выбираются пары маршрутов(родителей) для скрещивания. Наиболее приспособленные маршруты имеют больше шансов быть выбранными [9].

Шаг 4: для каждой пары родителей создается два новых маршрута(детей) путем обмена частями маршрута. Например, в одноточечном кроссовере выбирается случайная точка разреза и части маршрутов после этой точки меняются местами между родителями.

Шаг 5: с небольшим шансом может произойти мутация, то есть отдельные города в маршруте могут поменяться местами, что позволяет исследовать новые маршруты и избежать локальных минимумов.

Шаг 6: происходит замена маршрутов по принципу элитизма: лучшие маршруты из текущей популяции сохраняются для следующего поколения, а остальные заменяются новыми маршрутами, полученными путем кроссовера и мутации.

Шаг 7: повторение шагов селекции, кроссовера, мутации и замены выполняется до тех пор, пока не будет достигнута заданное число поколений или другой критерий остановки.

Данный алгоритм может быть легко адаптирован под различные условия задачи, что делает его гибким и эффективным в нахождении решения различных сложных задач.

2.2.5 Алгоритм муравьиной колонии

Алгоритм муравьиной колонии основывается на моделировании поведения реальных муравьев, которые прокладывают путь от своей колонии к источнику пищи. Муравьи оставляют феромоны на своем пути и с увеличением количества феромонов вероятность выбора этого пути другими муравьями возрастает [15]. Со временем муравьи находят оптимальные пути благодаря усилению феромонных следов на кратчайших маршрутах. Можно адаптировать данный алгоритм под условие задачи нахождения оптимального маршрута по двум критериям.

Основные шаги алгоритма описаны ниже.

Шаг 1: создается граф на основе данных о расстояниях и стоимости маршрутов. Задаются начальные параметры алгоритма, такие как количество муравьев, коэффициенты испарения и усиления феромонов. Инициализируются феромонные следы на всех ребрах графа некоторым малым значением.

Шаг 2: каждый муравей начинает свой путь с начальной вершины. На каждой вершине муравей выбирает следующее ребро на основе вероятности, зависящей от количества феромонов и его эвристической оценки. Вероятность

выбора ребра (i, j) муравьем k определяется формулой (7):

$$P_{i,j}^{k} = \frac{\left[\tau_{i,j}\right]^{\alpha}\left[\eta_{i,j}\right]^{\beta}}{\sum_{l \in allowed}\left[\tau_{i,l}\right]^{\alpha}\left[\eta_{i,l}\right]^{\beta}}, \quad (7)$$

где $\tau_{i,j}$ – количество феромонов на ребре (i, j) ;

$\eta_{i,j} = \frac{1}{d_{i,j}+c_{i,j}}$ – эвристическая информация, учитывающая расстояние $d_{i,j}$ и стоимость $c_{i,j}$;

α и β – параметры, управляющие значимостью феромонов и эвристической информации;

$allowed$ – множество вершин, которые муравей еще не посетил.

Шаг 3: после завершения всех муравьиных путей феромоны на ребрах обновляются. Происходит испарение феромонов на ребрах, то есть их количество уменьшается с учетом коэффициента испарения p по формуле $\tau_{i,j} = (1 - p) * \tau_{i,j}$. Так же происходит увеличение феромонов на ребрах, пройденных муравьями по формуле (8):

$$\tau_{i,j} = \tau_{i,j} + \sum_{k} \Delta\tau_{i,j}^{k}, \quad (8)$$

где $\Delta\tau_{i,j}^{k}$ – количество феромонов, оставленных муравьем k на ребре (i, j) , которое пропорционально обратной суммарной длине и стоимости пути муравья k.

Шаг 4: алгоритм повторяется до достижения заданного числа итераций или до достижения стабильности феромонных следов.

Данный алгоритм позволяет находить оптимальные маршруты, но может быть сложен ввиду необходимости тонкой настройки параметров для получения качественного решения [17].

2.3 Сравнительный анализ алгоритмов поиска оптимального маршрута для доставки грузов

Для проведения сравнительного анализ различных алгоритмов поиска оптимального маршрута для доставки грузов приведем основные достоинства и недостатки каждого из рассмотренных алгоритмов (таблица 3).

Таблица 3 – Достоинства и недостатки алгоритмов

Алгоритм	Достоинства	Недостатки
Ветвей и границ	Гарантированно находит оптимальное решение, учитывает сложные критерии, точный метод	Экспоненциальная сложность в худшем случае, плохо масштабируется для больших графов, высокие требования к памяти
Дейкстра	Высокая эффективность и производительность, легкость реализации, хорошо масштабируется для средних графов	Оптимален только для длины, не учитывает стоимость, не работает с отрицательными весами, требует модификации для учета двух весов
Ближайший сосед	Простота реализации, высокая скорость выполнения для небольших графов	Часто не находит оптимальное решение, решения часто далеки от оптимальных, ограниченная применимость для сложных задач
Локальный поиск	Простота реализации, хорошо масштабируется для средних графов, гибкость в адаптации для различных задач	Может застревать в локальных оптимумах, качество решений зависит от начального состояния и стратегии, не гарантирует нахождение глобального оптимума
Имитация отжига	Способен избегать локальных минимумов, хорошо масштабируется для больших графов, найденные решения часто близки к оптимальному	Сложность настройки параметров, могут быть высокие временные затраты, требует значительных вычислительных ресурсов
Генетический алгоритм	Способен находить решения близкие к глобальному оптимуму, хорошо масштабируется для больших задач, гибкость и адаптивность	Высокая сложность реализации, требует настройки множества параметров, высокие временные затраты

Продолжение таблицы 3

Муравьиная колония	Способен находить хорошие решения, хорошо масштабируется для больших задач, эффективен для решения задач с несколькими весами	Высокая сложность реализации, требует настройки множества параметров, высокие временные затраты

В качестве заключения приведем краткое описание каждого из ранее изученных алгоритмов:

- алгоритм ветвей и границ отлично подходит для задач, в которых нужно найти оптимальное решение, но не применим к большим графам из-за экспоненциальной сложности;
- алгоритм Дейкстры эффективен и легко реализуем, но требует модификации для учета двух весов;
- алгоритм ближайшего соседа быстрый и простой, но часто находит неоптимальные решения, что ограничивает его применение;
- алгоритм локального поиска гибкий и легко адаптируемый, но может застревать в локальных оптимумах, что не гарантирует нахождения глобального оптимума;
- алгоритм имитации отжига хорошо масштабируется и избегает локальных минимумов, но требует сложной настройки параметров и может быть медленным;
- генетический алгоритм – мощный и гибкий, способен находить решения близкие к оптимальному, но требует настройки и высоких вычислительных ресурсов [12, с. 50-51];
- алгоритм муравьиной колонии эффективен для задач с несколькими весами и хорошо масштабируется, но сложен в реализации и требует значительных временных затрат.

Исходя из вышеописанного, можно сделать вывод, что данные алгоритмы используются для решения различных задач и эффективность

каждого из них зависит от условий конкретной задачи.

Выводы по разделу 2

В данном разделе были рассмотрены различные алгоритмы для поиска оптимального маршрута для доставки грузов. Также был произведен их сравнительный анализ.

Каждый из рассмотренных алгоритмов способен решить поставленную задачу, но мной был выбран комбинированный алгоритм, который будет состоять из алгоритма Дейкстры и генетического алгоритма.

В рамках данной задачи, алгоритм Дейкстры будет применен для локальной оптимизации, то есть для поиска наилучшего маршрута между парами городов, а генетический алгоритм будет глобально оптимизировать маршрут, определяя наиболее выгодный порядок посещения указанных городов.

3 Программная реализация и тестирование алгоритма поиска оптимального маршрута для доставки грузов

3.1 Выбор языка программирования и среды разработка

Для реализации алгоритма был выбран язык программирования Python. Этот язык является хорошим выбором для нахождения оптимального маршрута для доставки грузов.

Python обладает обширной системой библиотек для математических вычислений и работы с графами. Например, библиотека NetworkX предоставляет большие возможности для создания и анализа графов, а библиотека matplotlib помогает визуализировать граф.

Так же Python имеет легкий и интуитивный синтаксис, что ускоряет процесс разработки программного обеспечения и делает код более читаемым и поддерживаемым. Большое количество документации и литературы упрощают изучение и использование языка

Python работает на всех платформах, что позволяет разрабатывать кроссплатформенные приложения.

В итоге Python предоставляет все необходимые инструменты эффективной реализации алгоритма поиска оптимального маршрута с учетом коэффициентов расстояния и стоимости.

В качестве среды разработки был выбран PyCharm, так как предоставляет полнофункциональную интегрированную среду разработки, которая облегчает написание, отладку и тестирование кода на Python и предоставляет все необходимые инструменты для разработки программ.

3.2 Разработка программного обеспечения

Для реализации программы поиска оптимального маршрута для доставки грузов был выбран комбинированный метод, который включает в

себя алгоритма Дейкстры и генетический алгоритм

Далее я опишу программную реализацию разработанного алгоритма

В начале я описываю раздел, отвечающий за импорт необходимых библиотек (рисунок 3):

- Pandas необходим для чтения данных из Excel файлов с расширением .xslx.
- NetworkX используется для создания графов из данных, прочитанных из Excel файлов.
- random применяется в работе генетического алгоритма для создания случайного процесса мутации.
- matplotlib используется для визуализации графа маршрутов.

```
import pandas as pd
import networkx as nx
import random
import matplotlib.pyplot as plt
```

Рисунок 3 – Импорт необходимых библиотек

Далее данные из двух Excel файлов записываются в соответствующие датафреймы. В первый датафрейм записываются данные о протяженности маршрутов между городами, а во второй стоимость этих маршрутов. Первый столбец таблицы используется в качестве индекса, то есть названия города. Затем имеющиеся датафреймы преобразуются в граф с помощью NetworkX (рисунок 4).

```
# Чтение данных о длине маршрутов из Excel
df_distance = pd.read_excel( io: 'rast1.xlsx', index_col=0)

# Чтение данных о стоимости маршрутов из Excel
df_cost = pd.read_excel( io: 'cost1.xlsx', index_col=0)

# Создание графов
G_distance = nx.from_pandas_adjacency(df_distance)
G_cost = nx.from_pandas_adjacency(df_cost)
```

Рисунок 4 – Чтение данных и создание графов

С клавиатуры вводится список городов, через которые обязательно должен пролегать маршрут. Затем следует проверка существуют ли указанные города в графе. Также проводится проверка на количество введенных городов. Если городов для построения маршрута недостаточно, программа завершает работу. Код представлен на рисунке 5.

```
print("Введите список городов через запятую (например, Москва, Санкт-Петербург, Казань):")
input_cities = input().strip().split(',')

# Проверка, что все введенные города существуют в графе
cities = [city.strip() for city in input_cities if city.strip() in G_distance.nodes]

if len(cities) < 2:
    print("Необходимо ввести как минимум два существующих города.")
    exit()
```

Рисунок 5 – Ввод и проверка данных

Объявляем функцию (рисунок 6), которая создает объединенный граф, в котором ребра имеют комбинированный вес, рассчитанный на основе коэффициентов важности w_1 и w_2 для расстояния и стоимости соответственно.

```python
def create_combined_graph(G_distance, G_cost, w1, w2):
    G_combined = nx.Graph()
    for u, v in G_distance.edges():
        combined_weight = w1 * G_distance[u][v]['weight'] + w2 * G_cost[u][v]['weight']
        G_combined.add_edge(u, v, weight=combined_weight)
    return G_combined
```

Рисунок 6 – Создание объединенного графа

Функция на рисунке 7 оценивает общую стоимость маршрутов между городами. В ней используется алгоритм Дейкстры для нахождения кратчайшего пути между парами городов в каждом маршруте с учетом весовых коэффициентов. Фактически в данной функции алгоритм Дейкстры выполняет локальную оптимизацию, то есть находит наилучший маршрут между любыми двумя городами.

```python
def path_cost(G_combined, G_distance, G_cost, path):
    total_cost = 0
    total_distance = 0
    full_path = []
    for i in range(len(path) - 1):
        # Используем алгоритм Дейкстры для поиска кратчайшего пути в комбинированном графе
        try:
            shortest_path = nx.dijkstra_path(G_combined, path[i], path[i + 1], weight='weight')
            for j in range(len(shortest_path) - 1):
                if shortest_path[j] not in full_path:
                    full_path.append(shortest_path[j])
                total_cost += G_cost[shortest_path[j]][shortest_path[j + 1]]['weight']
                total_distance += G_distance[shortest_path[j]][shortest_path[j + 1]]['weight']
        except nx.NetworkXNoPath:
            print(f"Нет доступного пути между {path[i]} и {path[i + 1]}.")
            return float('inf'), float('inf'), []
    full_path.append(path[-1])
    return total_cost, total_distance, full_path
```

Рисунок 7 – Функция расчета общей стоимости маршрута

Создается функция, которая инициализирует популяцию индивидов (маршрутов) для работы генетического алгоритма (рисунок 8). Для каждого индивида в популяции создается копия списка городов и с помощью функции random.shuffle копия списка городов перемешивается случайным образом. После перемешивания городов получается один случайный маршрут, который добавляется в популяцию. Данная функция обеспечивает случайное, но разнообразное начальное состояние популяции, что помогает избегать локальных оптимумов.

```
def initialize_population(cities, population_size):
    population = []
    for _ in range(population_size):
        individual = cities[:]
        random.shuffle(individual)
        population.append(individual)
    return population
```

Рисунок 8 – Инициализация популяции

Функция создает пустой список, который будет содержать оценки приспособленности каждого индивидуума в популяции (рисунок 9). Для каждого индивидуума вызывается функция path_cost, которая вычисляет стоимость, длину маршрута и полный путь, а затем полученные данные добавляются в список оценок.

```
def evaluate_population(population, G_combined, G_distance, G_cost):
    fitness_scores = []
    for individual in population:
        cost, distance, full_path = path_cost(G_combined, G_distance, G_cost, individual)
        fitness_scores.append((cost, distance, full_path))  # сохраняем полный путь
    return fitness_scores
```

Рисунок 9 – Функция оценки приспособленности

Функция на рисунке 10 выбирает родителей для последующего скрещивания. Для этого список с оценками приспособленности сортируется по первому элементу, который представляет собой общую стоимость маршрута. После сортировки выбираются первые два элемента списка, так как они имеют наименьшую стоимость. В итоге, выбранные родители возвращаются в качестве результата работы функции

```
def select_parents(fitness_scores):
    fitness_scores.sort(key=lambda x: x[0])  # Сортировка по стоимости
    return fitness_scores[:2]
```

Рисунок 10 – Функция селекции родителей

Функция скрещивания (рисунок 11) получает размеры родителей, затем выбираются две случайные точки раздела в геномах родителей. Эти точки указывают, где произойдет разрез генов и где начнутся и закончатся гены, которые будут перекрещены. Для каждого ребенка пустые списки, затем в них копируется части родительского генома. Далее с помощью функции fill_child, которая будет рассмотрена ниже, заполняются оставшиеся части генома потомков генами другого родителя, что гарантирует уникальность генов.

```
def crossover(parent1, parent2):
    size = len(parent1)
    start, end = sorted(random.sample(range(size), k: 2))
    child1 = [None] * size
    child2 = [None] * size

    child1[start:end] = parent1[start:end]
    child2[start:end] = parent2[start:end]

    fill_child(child1, parent2, start, end)
    fill_child(child2, parent1, start, end)

    return child1, child2
```

Рисунок 11 – Функция скрещивания

Функция, которая заполняет оставшиеся части генома потомка генами от другого родителя представлена на рисунке 12. Она заполняет все незаполненные места (гены со значением None) и в каждой итерации цикла проверяет, есть ли в геноме ребенка текущий ген родителя. Если гена нет, то он добавляется на текущую позицию. Функция играет важную роль в скрещивании, где она гарантирует что каждый родительский ген присутствует в геноме потомка.

```
def fill_child(child, parent, start, end):
    size = len(parent)
    p_idx, c_idx = end, end
    while None in child:
        if parent[p_idx % size] not in child:
            child[c_idx % size] = parent[p_idx % size]
            c_idx += 1
        p_idx += 1
```

Рисунок 12 – Функция для заполнения оставшейся части геномов

Функция мутации (рисунок 13) помогает в поддержании генетического разнообразия популяции. Сначала функция генерирует случайное число от нуля до единицы, а затем сравнивает его с заданной вероятностью мутации, если число меньше, то запускается процесс мутации. При мутации выбираются два случайных гена и меняются местами. Затем мутировавший индивидуум возвращается в качестве результата работы функции.

```
def mutate(individual, mutation_rate):
    if random.random() < mutation_rate:
        i, j = random.sample(range(len(individual)), k: 2)
        individual[i], individual[j] = individual[j], individual[i]
    return individual
```

Рисунок 13 – Функция мутации

В участке кода, представленном на рисунке 14, объявляются переменные для работы программы: размер популяции, количество поколений, вероятность мутации и коэффициенты важности. Также создается комбинированный граф и инициализируется популяция.

```
population_size = 100
generations = 500
mutation_rate = 0.05
w1 = 0.5  # вес для длины маршрута
w2 = 0.5  # вес для стоимости маршрута

# Создание комбинированного графа
G_combined = create_combined_graph(G_distance, G_cost, w1, w2)

# Инициализация популяции
population = initialize_population(cities, population_size)
```

Рисунок 14 – Объявление параметров программы

Запускается основной цикл генетического алгоритма (рисунок 15), который будет работать заданное количество поколений. Внутри цикла работают все вышеописанные функции, благодаря чему происходит процесс отбора оптимального решения.

```
for generation in range(generations):
    fitness_scores = evaluate_population(population, G_combined, G_distance, G_cost)
    parents = select_parents(fitness_scores)
    new_population = [parents[0][2], parents[1][2]]  # Элитизм: сохраняем лучших

    while len(new_population) < population_size:
        parent1 = random.choice(parents)[2]
        parent2 = random.choice(parents)[2]
        child1, child2 = crossover(parent1, parent2)
        new_population.extend([mutate(child1, mutation_rate), mutate(child2, mutation_rate)])

    population = new_population
```

Рисунок 15 – Основной цикл работы генетического алгоритма

В конце работы генетического алгоритма функция evaluate_population применяется к последнему поколению и возвращает индивидуума с минимальной общей стоимостью маршрута. Далее из индивидуума извлекаются данные о стоимости, расстоянии и полном маршруте (рисунок 16).

```
best_individual = min(evaluate_population(population, G_combined, G_distance, G_cost), key=lambda x: x[0])

best_path_cost, best_path_distance, best_full_path = best_individual
```

Рисунок 16 – Определение лучшего индивидуума

Код, представленный на рисунке 17, выводит информацию об оптимальном маршруте, а также граф, отображающий его.

```
print(f'Оптимальный маршрут, проходящий через указанные города: {best_full_path}')
print(f'Общая длина оптимального маршрута: {best_path_distance}')
print(f'Общая стоимость оптимального маршрута: {best_path_cost}')

# Построение графа
pos = nx.spring_layout(G_distance)
nx.draw(G_distance, pos, with_labels=True)
plt.show()
```

Рисунок 17 – Вывод результатов

В этом подразделе были подробно описаны шаги, которые были проделаны для разработки программного обеспечения. В следующем подразделе проведем тестирование разработанного программного обеспечения и сделаем вывод о его работоспособности.

3.3 Тестирование программного обеспечения

Для проверки работоспособности написанного мной программного обеспечения проведем тестирование.

Данные о маршрутах представлены в виде двух Excel таблиц (рисунки 18 и 19). Значения в таблицах выбраны произвольно для полной демонстрации работы алгоритма. Если в ячейке нулевое значение это значит, что прямого маршрута не существует.

	Тольятти	Самара	Москва	Санкт-Петербург	Мурманск	Воронеж	Донецк
Тольятти	0	10	983	11000	0	857	0
Самара	10	0	0	1000	2661	931	1406
Москва	983	0	0	705	1930	519	1205
Санкт-Петербург	11000	1000	705	0	1334	0	0
Мурманск	0	2661	1930	1334	0	2453	3139
Воронеж	857	931	519	0	2453	0	692
Донецк	0	1406	1205	0	3139	692	0

Рисунок 18 – Матрица расстояний

	Тольятти	Самара	Москва	Санкт-Петербург	Мурманск	Воронеж	Донецк
Тольятти	0	10000	660	110	0	560	0
Самара	10000	0	0	100	4100	700	1020
Москва	660	0	0	580	3200	480	990
Санкт-Петербург	0	100	580	0	2600	0	0
Мурманск	0	4100	3200	2600	0	3900	5500
Воронеж	560	700	480	0	3900	0	480
Донецк	0	1020	990	0	5500	480	0

Рисунок 19 – Матрица стоимостей

Задаем начальные параметры (рисунок 20) и запускаем программу.

```
population_size = 50
generations = 100
mutation_rate = 0.05
w1 = 0.5  # вес для длины маршрута
w2 = 0.5  # вес для стоимости маршрута
```

Рисунок 20 – Заданные начальные параметры

Далее вводим с клавиатуры список городов, через которые обязательно должен пролегать маршрут и получаем оптимальный (рисунок 21).

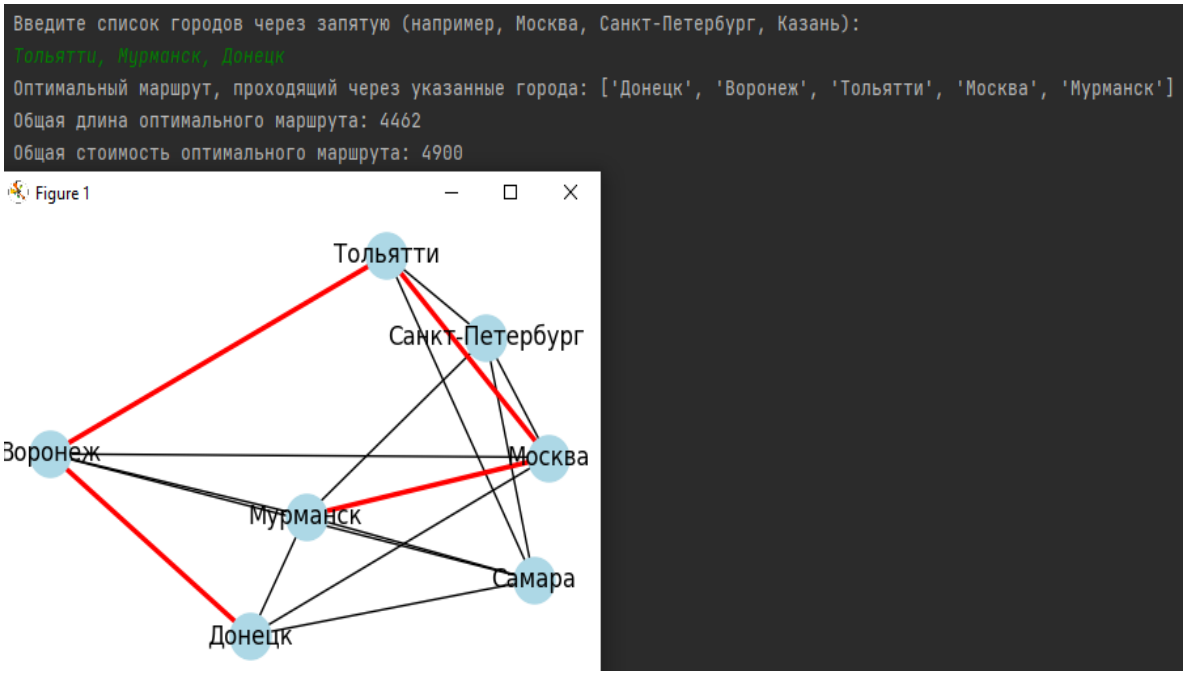

Рисунок 21 – Результат работы программы

Программа нашла маршрут. Путем математических вычислений проверим получившуюся итоговую длину и стоимость маршрута. Длина $= 692 + 857 + 983 + 1930 = 4462$, стоимость $= 480 + 560 + 660 + 3200 = 4900$. Из расчета видно, что длинна и стоимость вычисляются правильно.

Также проверим работу программы при смене приоритета. Установим значения коэффициентов важности 0.1 для расстояния, 0.9 для стоимости.

Результат работы программы при измененных коэффициентах представлен на рисунке 22.

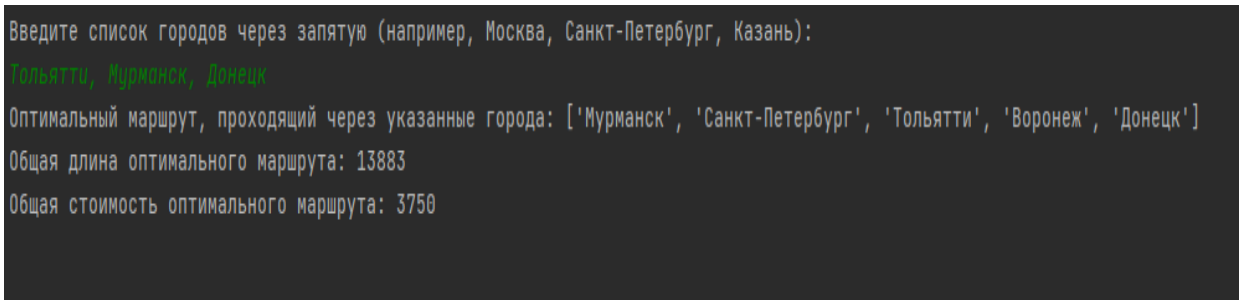

Рисунок 22 – Результат работы с коэффициентами 0.1 и 0.9

Как видно из результатов работы программы, при увеличении важности стоимости, алгоритм строит другой маршрут и уменьшает стоимость по сравнению с программой с равнозначными коэффициентами.

Из представленных результатов тестирования можно сделать вывод о том, что разработанное программное обеспечение функционирует корректно и предоставляет необходимый для работы функционал. В процессе тестирования ошибок в работе программы не обнаружено.

Выводы по разделу 3

В ходе работы над разделом было разработано и протестировано программное обеспечение для поиска оптимального маршрута для доставки грузов.

Программа была написана на основе генетического алгоритма и алгоритма Дейкстры. Был выбран язык программирования Python и среда разработки PyCharm.

В ходе тестирования была продемонстрирована правильность работы программы. Ошибок в работе программы не обнаружено.

Заключение

В данной выпускной квалификационной работе было проведено изучение предметной области и на основе полученных знаний было разработано программное обеспечение для нахождения оптимального маршрута для доставки грузов.

В процессе проведения исследования были изучены различные методы и алгоритмы для поиска оптимальных маршрутов, проведен их сравнительный анализ, что позволило создать теоретическую базу для разработки собственного алгоритма.

При описании программной реализации были исследованы необходимые инструменты и технологии.

Для разработки был выбран язык программирования Python и дополнительные библиотеки. Программное обеспечение было написано в интегрированной среде разработки PyCharm.

В процессе тестирования программы были представлены примеры входных данных и результаты их обработки. Программа успешно выполнила поставленную задачу и нашла оптимальный маршрут, проходящий через заданные населенные пункты.

Как видно из результатов работы программы, при увеличении важности стоимости, алгоритм строит другой маршрут и уменьшает стоимость по сравнению с программой с равнозначными коэффициентами.

В результате тестирования ошибок не обнаружилось, а результаты соответствовали ожидаемым.

Подводя итоги, можно сказать, что разработанное программное обеспечение эффективно справляется с нахождением оптимального маршрута и позволяет автоматизировать процесс его поиска.

Список используемой литературы и источников

1. Алгоритм Дейкстры [Электронный ресурс] // HexLet URL: https://ru.hexlet.io/courses/graphs/lessons/shortest_paths/theory_unit – (дата обращения 27.03.2024).
2. Алгоритм Дейкстры [Электронный ресурс] // Алгоритмика/CS URL: https://ru.algorithmica.org/cs/shortest-paths/dijkstra/ – (дата обращения 21.03.2024).
3. Алгоритм Дейкстры нахождения кратчайшего пути [Электронный ресурс] // prog-cpp URL: https://prog-cpp.ru/deikstra/ – (дата обращения 24.03.2024).
4. Алгоритм нахождения оптимального пути в транспортной сети в условиях изменяемых параметров [Электронный ресурс] // Bsur URL: https://libeldoc.bsuir.by/bitstream/123456789/51995/1/Koryachko_Algoritm.pdf – (дата обращения 15.04.2024).
5. Алгоритмы поиска пути: Алгоритм дейкстры и A* [Электронный ресурс] // Habr URL: https://habr.com/ru/companies/otus/articles/748470/ – (дата обращения 18.03.2024).
6. Галяутдинов Р.Р. Задача коммивояжера — метод ветвей и границ // Сайт преподавателя экономики. [2023]. URL: https://galyautdinov.ru/post/zadacha-kommivoyazhera (дата обращения: 24.04.2024).
7. Генетические алгоритмы – эволюционные методы поиска [Электронный ресурс] // algolist manual URL: https://algolist.manual.ru/ai/ga/ga3.php – (дата обращения 03.04.2024).
8. Генетические алгоритмы и задачи актуальной математики [Электронный ресурс] // СГУ URL: http://elibrary.sgu.ru/uch_lit/1348.pdf – (дата обращения 12.04.2024).
9. Генетический алгоритм. Просто о сложном. Рассказ Марка Андреева [Электронный ресурс] // HABR URL: https://habr.com/ru/articles/128704/

– (дата обращения 06.04.2024).

10. Каштаева, С.В. Методы оптимизации : учебное пособие / С.В. Каштаева; Министерство сельского хозяйства Российской Федерации, федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский аграрно-технологический университет имени академика Д.Н. Прянишникова». – Пермь : ИПЦ «Прокростъ», 2020. – 84 с
11. Метод поиска оптимального и наиболее близких к нему маршрутов перевозки грузов [Электронный ресурс] // cyberleninka URL: https://cyberleninka.ru/article/n/metod-poiska-optimalnogo-i-naibolee-blizkih-k-nemu-marshrutov-perevozki-gruzov – (дата обращения 03.03.2024).
12. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.
13. Поиск в графе с весами: алгоритм Дейкстры. [Электронный ресурс] // Spbu URL: https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms1_2022/okhotin_algorithms1_2022_l4.pdf – (дата обращения 30.03.2024).
14. Применение математических методов динамического программирования к организации системы доставки грузов [Электронный ресурс] // Журнал «Human Progress» URL: http://progress-human.com/images/2018/Tom4_6/Dolgova.pdf – (дата обращения 21.04.2024).
15. Разработка «направленного» муравьиного алгоритма для оптимизации производственного расписания [Электронный ресурс] // cyberleninka URL: https://cyberleninka.ru/article/n/razrabotka-napravlennogo-muravinogo-algoritma-dlya-optimizatsii-proizvodstvennogo-raspisaniya/viewer – (дата обращения 15.03.2024).
16. Development of an Algorithm for Constructing a Route Based on the Data of

GPX-Files [Электронный ресурс] // IEEE Xplore URL: https://ieeexplore.ieee.org/document/10113405 – (дата обращения 28.04.2024).

17. Genetic Algorithm- A Literature Review [Электронный ресурс] // IEEE Xplore URL: https://ieeexplore.ieee.org/document/8862255 – (дата обращения 16.04.2024).
18. Introduction to Ant Colony Optimization [Электронный ресурс] // geeksforgeeks URL: https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization – (дата обращения 25.04.2024).
19. Introduction to Branch and Bound – Data Structures and Algorithms Tutorial [Электронный ресурс] // geeksforgeeks URL: https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial – (дата обращения 22.04.2024).
20. What is Route Optimization Algorithm and How Does it Work? [Электронный ресурс] // upper URL: https://www.upperinc.com/blog/route-optimization-algorithm – (дата обращения 22.04.2024).