**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Казанский национальный исследовательский технический университет**

**им. А.Н. Туполева-КАИ»**

Тема: «Проект по созданию модели для оптимизации маршрутов с учетом расстояния, времени в пути и потребления топлива с использованием современных языков программирования»

Выполнил: Ученик 10 «И» Хамидуллин. Э.Д.

Руководитель проекта: Филиппов Т. А.

Научный консультант: Филиппов Т. А.

Казань 2024.г

**Оглавление стр.**

Введение

**Введение**

В современном обществе постоянно возникает задача по оптимизации транспортных маршрутов, ввиду быстрого развития инфраструктуры и увеличивающихся требований к эффективной логистике. Поэтому становится всё более важным создание интеллектуальных систем, которые могут учитывать множество факторов, чтобы уменьшить затраты и повысить эффективность. Основными критериями при выборе оптимального маршрута являются расстояние, время в пути и расход топлива. Современные языки программирования и технологии открывают новые возможности для решения таких задач, способствуя созданию моделей, которые не только анализируют большие объемы данных, но и предоставляют решения в режиме реального времени.

**Актуальность:** Рост грузо- и пассажироперевозок требует повышения эффективности логистики. Постоянный рост цен на топливо и необходимость быстрой доставки делают минимизацию затрат и времени критичными. Современные языки программирования и машинное обучение позволяют создавать точные и адаптивные модели, учитывающие динамику дорожной ситуации (пробки, ремонт, погода) и характеристики транспорта. Модель позволит снизить транспортные расходы, время доставки и негативное воздействие на окружающую среду благодаря оптимизации расхода топлива. Проект важен для транспортных компаний, логистических операторов и всей транспортной системы в целом.

**Цель**

этого проекта заключается в создании модели, которая интегрирует данные о дорожной сети и транспортных средствах, принимая во внимание различные условия движения, такие как загруженность дорог и погодные условия, для предложений оптимальных маршрутов. Благодаря использованию современных языков программирования, таких как Python и R, а также инструментов машинного обучения и анализа данных, можно обеспечить высокую точность и скорость при планировании маршрутов. Таким образом, проект нацелен на повышение эффективности логистических процессов, снижая временные и финансовые затраты для бизнеса и общества в целом.

**Задачи**

* Разработка модели маршрутизации
* Выбор и интеграция API геоданных: например, Google Maps API, OpenStreetMaps.
* Разработка модуля расчета потребления топлива
* Тестирование и оптимизация модели
* Визуализация

Объект исследования — **транспортные маршруты**, которые необходимо оптимизировать с учётом различных параметров (расстояние, время в пути, потребление топлива).

Предмет исследования — **методы и алгоритмы**, используемые для оптимизации транспортных маршрутов.

**Методы исследования**

* Математическое моделирование
* Алгоритмические подходы:
* Программная реализация
* Экспериментальные исследования и тестирование

**Теоретическая значимость проекта**, посвященного созданию модели для оптимизации маршрутов с учетом расстояния, времени в пути и потребления топлива, заключается в разработке метода, который может быть применен в условиях воздействия множества технико-эксплуатационных и экономических факторов. Этот метод обеспечивает объективное решение задач маршрутизации в условиях неопределенности.

**Практическая значимость проекта** заключается в возможности улучшения технико-экономических показателей предприятия за счет экономии трудовых и финансовых ресурсов. Применение разработанной методики также повышает точность планирования грузоперевозок и исключает субъективные ошибки, создавая автоматизированную систему управления перевозками.

**Ι. Основная часть**

1.1 Обзор существующих методов оптимизации маршрутов.

**Линейное программирование** (ЛП) представляет собой наиболее простой и детально изученный раздел исследования операций, также известный как математическое программирование. Этот раздел включает методы, которые позволяют находить оптимальные решения для широкого диапазона задач.

Задачи, решаемые в линейном программировании, описываются с использованием линейных функций, уравнений и неравенств. В частности:

* Целевая функция L(X) является линейной функцией от элементов решения.
* Ограничения, накладываемые на возможные решения, выражаются в виде линейных уравнений или неравенств.

**Метод северо-западного угла** является эвристическим подходом *(эвристический алгоритм (эвристика) —*[*алгоритм*](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)*решения задачи, включающий практический метод, не являющийся гарантированно точным или оптимальным, но достаточный для решения поставленной задачи.)* для нахождения решения, при котором значения размещаются в матрице решений, начиная с верхней левой ячейки. Заполнение начинается с ресурса с наименьшей стоимостью и продолжается последовательно, что позволяет получить предварительное решение.

***Эвристические алгоритмы*** делятся на:

1) двухфазные алгоритмы;

2) конструктивные алгоритмы;

3) улучшающие алгоритмы.

**Метод минимальных тарифов**  
Эвристический метод нахождения решения, когда значения расставляются в матрице решений начиная с наиболее дешевого плеча поставки и далее по порядку увеличения стоимости перевозки (предварительное решение)

**Метод Фогеля**  
Эвристический метод нахождения решения, когда предварительно по каждому столбцу и строке матрицы тарифов рассчитывается вспомогательный коэффициент, равный разности двух наименьших тарифов в этой строке или столбце. Первым распределяется ресурс в столбце или строке с наибольшим значением коэффициента, далее по порядку убывания величины коэффициента (предварительное решение, близкое к оптимальному)

**Алгоритм Свира (дворника-стеклоочистителя)**  
Один из эвристических (нематематических) вариантов построения маршрута при большом количестве точек. Заключается в нанесении на карту всех точек маршрута и воображаемом движении "дворника-стеклоочистителя" по карте из места отправления по кругу и стирании заказов (точек маршрута). Как только сумма заказов "стертых" магазинов достигнет вместимости транспортного средства, фиксируется сектор, обслуживаемый одним кольцевым маршрутом, и намечается путь объезда потребителей.

**Метаэвристические алгоритмы**,

методы оптимизации, разработанные в соответствии с стратегиями, изложенными в метаэвристической структуре, - как следует из названия, всегда эвристичны по своей природе. Этот факт отличает их от точных методов, которые приходят с доказательством того, что оптимальное решение будет найдено в конечном (хотя и часто непомерно большом) количестве времени. Поэтому метаэвристика разработана специально для того, чтобы найти решение, которое «достаточно хорошо» в вычислительном времени, которое «достаточно мало». В результате они не подвергаются комбинаторному взрыву – явлению, в котором вычислительное время, необходимое для нахождения оптимального решения NP-трудных задач, возрастает как экспоненциальная функция размера задачи.

Список из самых часто используемых метаэвристических алгоритмов:

*1)* ***муравьиный алгоритм*** *(Ant Algorithm);*

*2) детерминированный отжиг (Deterministic Annealing);*

*3) генетические алгоритмы (Genetic Algorithms);*

*4) поиск с запретами (Tabu Search).*

1.2 Математическая модель задачи маршрутизации транспорта

1. Граф: Задача представляется в виде графа G=(V, E), где:

V - множество вершин (узлов): Представляет собой точки, которые нужно посетить (склады, клиенты, точки загрузки/выгрузки и т.д.).

E - множество ребер: Представляет собой пути между вершинами (дороги, маршруты). Каждое ребро может иметь ассоциированные с ним параметры (расстояние, время проезда, стоимость и т.д.).

Муравьиный алгоритм (Ant Colony Optimization, ACO) — это метаэвристический алгоритм, вдохновленный поведением муравьев, которые ищут кратчайший путь к пище. Этот алгоритм часто применяется для решения задач оптимизации, включая задачу маршрутизации транспортных средств (Capacitated Vehicle Routing Problem, CVRP).

**2. Задачи маршрутизации транспорта (Vehicle Routing Problems, VRP)**

задачи комбинаторной оптимизации, в которых для парка транспортных средств, расположенных в одном или нескольких депо, должен быть определен набор маршрутов до нескольких отдаленных точек-потребителей.

**VRP** была сформулирована более 40 лет назад и сейчас является одной из интересных комбинаторных задач целочисленного программирования, относящаяся к классу NP-трудных задач, что означает, что вычислительная сложность задачи зависит от размера входных данных экспоненциально.

Задачу можно сформулировать следующим образом: используя ограниченное количество машин, доставить товары покупателям. Учитывая, ограничения:

* вместимость каждой машины
* время доставки товара покупателю
* количество точек доставки
* время работы водителя
* Оптимизировать пробег машин для экономии времени или топлива.

Для таких задач обычно достаточно искать приближенные решения, которые находятся достаточно быстро и достаточно точны для требуемых целей. Обычно это достигается разными эвристическими методами.

Задачи маршрутизации являются ключевыми в областях транспортных перевозок, перемещения и логистики. Во многих областях рынка доставка товара добавляет к его стоимости сумму, сравнимую со стоимостью самого товара. Тем не менее, использование компьютерных методов оптимизации доставки товара часто выражается в экономии порядка 5-20% от общей его стоимости.

**3.1 Разновидности VRP**

Обычно, в реальных задачах оптимизации возникает множество дополнительных ограничений и вариаций, наиболее важные из которых перечислены ниже.

1. Capacitated VRP (CVRP): каждое транспортное средство имеет ограниченную грузоподъемность.
2. VRP with Time Windows (VRPTW) или (TCVRP): каждый заказчик должен быть обслужен в определенное «временное окно».
3. Multiple Depot VRP (MDVRP): используются несколько депо для обслуживания клиентов.
4. VRP with Pick-Ups and Delivering (VRPPD): клиенты могут возвращать некоторые товары в депо.
5. VRP with Backhauls (VRPB): аналогично предыдущей, но возврат начинается только после доставки всех товаров из депо.
6. Split Delivery VRP (SDVRP): каждый клиент может обслуживаться одновременно несколькими машинами.
7. Periodic VRP (PVRP): доставка может осуществляться в течение нескольких дней.
8. Stochastic VRP (SVRP): некоторые компоненты задачи (количество и запросы клиентов, длина пути) могут иметь случайное поведение.
9. VRP with Satellite Facilities (VRPSF): существует возможность дозагрузки автомобиля на маршруте.

**Применение муравьиного алгоритма к CVRP**

Муравьиный алгоритм адаптируется для решения CVRP следующим образом:

1. **Инициализация феромонов**:
   * На каждом ребре графа (между клиентами и депо) инициализируется начальное значение феромона. Это значение может быть одинаковым для всех ребер или основано на эвристической информации (например, обратной пропорциональности расстоянию).
2. **Построение маршрутов**:
   * Каждый "муравей" (агент) строит маршрут, начиная с депо.
   * Муравей выбирает следующего клиента на основе вероятности, которая зависит от уровня феромона на ребре и эвристической информации (например, близости клиента).
   * Если добавление клиента превышает грузоподъемность транспортного средства, муравей возвращается в депо и начинает новый маршрут.
3. **Обновление феромонов**:
   * После того как все муравьи построили маршруты, феромоны обновляются.
   * Феромоны испаряются на всех ребрах (уменьшаются на определенный коэффициент).
   * Феромоны добавляются на ребра, которые были частью лучших маршрутов (например, маршрутов с минимальной общей длиной).
4. **Итерации**:
   * Процесс построения маршрутов и обновления феромонов повторяется в течение нескольких итераций.
   * Алгоритм стремится к тому, чтобы феромоны накапливались на ребрах, которые являются частью хороших решений.
5. **Завершение**:
   * Алгоритм завершается после достижения определенного числа итераций или выполнения другого критерия остановки.
   * Лучшее найденное решение возвращается как результат.

**3.2 Математическая модель задачи CVRP**

Бинарная переменная определяет перемещение транспортного средства по дуге в решении задачи. Целочисленная модель линейного программирования задачи CVRP [4] может быть представлена как:

Минимизация целевой функции:

(1)

С учетом условий:

(2)

(3)

(4)

(5)

(6)

(7)

Целевая функция (1) минимизирует общую стоимость перевозок. Ограничение модели (2) обеспечивает, чтобы каждого клиента посетило ровно одно транспортное средство. Следующие ограничения (3) и (4) гарантируют, что каждое транспортное средство может покинуть депо только один раз, а количество транспортных средств, посещающих каждый узел, равно количеству покидающих его. В (5) задаются ограничения вместимости транспортного средства, удостоверяющие, что сумма требований клиентов, принадлежащих маршруту транспортного средства, меньше или равна емкости транспортного средства, следующего по этому маршруту. Ограничение (6) отклоняет решения, которые не содержат циклов, исключающих депо. Оставшееся обязательное ограничение (7) определяет области определения переменных. Эта модель известна как a three-index vehicle flow formulation. Число неравенств для исключения циклов без депо увеличивается экспоненциально с увеличением количества узлов.

**ГЛАВА 2 РЕАЛИЗАЦИЯ ЭВРИСТИЧЕСКОГО АЛГОРИТМА ДЛЯ РЕШЕНИЯ ЗМТ**

Применение эвристического алгоритма перемещения в условиях ограничений по грузоподъемности можно разделить на несколько этапов:

1) обоснование выбора технических средств;

2) подготовка исходных данных;

3) реализация алгоритма и архитектуры приложения;

4) проведение вычислительного эксперимента. Готовый эвристический алгоритм перемещения, схематично изображенный на рисунке, должен будет решать задачу маршрутизации транспорта с учетом ограничения по грузоподъемности. Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

**II Практическая часть.**

Код решает задачу маршрутизации с использованием муравьиного алгоритма (Ant Colony Optimization, ACO) для задачи CVRP (Capacitated Vehicle Routing Problem). Основная цель — найти оптимальный маршрут для транспортного средства, которое должно посетить несколько точек (например, рестораны McDonald's в Москве), учитывая ограничение на грузоподъемность. Код также включает визуализацию маршрутов на карте и анимацию движения.

**Основные этапы работы кода:**

**1. Импорт библиотек**

* Используются библиотеки для работы с данными (pandas, numpy), визуализации (matplotlib, seaborn, folium, plotly), работы с графами (osmnx, networkx) и реализации муравьиного алгоритма (random, math).

**2. Класс AntCVRP**

* Класс реализует логику муравьиного алгоритма для задачи CVRP.
* **Параметры**:
  + A, B, U, P — параметры алгоритма, влияющие на выбор пути и обновление феромонов.
* **Методы**:
  + \_\_init\_\_: Инициализация муравья с начальным узлом, грузоподъемностью и графом.
  + init: Инициализация списка узлов для посещения.
  + choose\_next: Выбор следующего узла на основе вероятностей, учитывая феромоны и эвристическую информацию.
  + h\_value: Вычисление значения для вероятностного выбора следующего узла.
  + local\_updating\_rule: Локальное обновление феромонов на пройденном пути.
  + better: Сравнение двух путей.
  + end: Проверка завершения маршрута.

**3. Загрузка данных**

* Загружается CSV-файл с данными о местоположениях ресторанов McDonald's в Москве.
* Данные фильтруются по городу и переименовываются для удобства.

**4. Визуализация данных на карте**

* Создается интерактивная карта с использованием библиотеки folium.
* На карту добавляются маркеры для каждой точки. Начальная точка выделяется красным цветом, остальные — синим.

**5. Создание графа дорожной сети**

* С помощью библиотеки osmnx создается граф дорожной сети для Москвы.
* Граф дополняется информацией о скорости и времени в пути для каждого ребра.

**6. Нахождение ближайших узлов**

* Для каждой точки из данных находится ближайший узел в графе дорожной сети.

**7. Расчет матрицы расстояний**

* Создается матрица расстояний между узлами с использованием функции f, которая вычисляет кратчайшее расстояние между двумя узлами с учетом времени в пути.
* Если путь недостижим, используется значение np.inf.

**8. Визуализация тепловой карты**

* Матрица расстояний визуализируется в виде тепловой карты с использованием библиотеки seaborn.

**9. Решение задачи маршрутизации с использованием AntCVRP**

* Инициализируется муравей с начальным узлом и грузоподъемностью.
* Муравей выбирает следующий узел для посещения на основе вероятностей, учитывая феромоны и эвристическую информацию.
* После завершения маршрута обновляются феромоны на пройденном пути.

**10. Получение путей между узлами**

* Для каждого отрезка маршрута вычисляется кратчайший путь между узлами с использованием алгоритма Дейкстры.

**11. Визуализация маршрута на карте**

* Маршрут визуализируется на карте с использованием библиотеки folium.

**12. Создание анимации маршрута**

* Создается анимация движения по маршруту с использованием библиотеки plotly.
* Анимация включает:
  + Точки маршрута.
  + Начальную и конечную точки.
  + Линии, соединяющие точки маршрута.

Преимущества кода:

* Гибкость: Код может быть адаптирован для других городов и задач маршрутизации.
* Визуализация: Интерактивная карта и анимация делают результаты наглядными.
* Оптимизация: Использование муравьиного алгоритма позволяет находить близкие к оптимальным решения.

**Заключение:**

Разработанная модель демонстрирует высокую эффективность в решении задачи оптимизации маршрутов. Она может быть использована в логистике, транспортных компаниях, службах доставки и других областях, где требуется минимизация затрат на перемещение. Результаты экспериментов показали, что использование динамической адаптации муравьиного алгоритма (CVRP) будет эффективнее, чем использование простого муравьиного алгоритма. Точность вычислений увеличивается, уменьшается разброс получаемых решений, а главное, позволяет уменьшить время вычислений за счет уменьшения значений параметров итераций и численности муравейника, не ухудшив значения генерируемых решений и не используя посторонние способы улучшения. Таким образом, проект успешно решает поставленные задачи и предоставляет прочную основу для дальнейшего развития и внедрения в реальные системы. Использование динамической адаптации муравьиного алгоритма открывает новые возможности для повышения эффективности бизнеса и улучшения качества жизни.

**Литература**

1. Петросян Леон Аганесович и Зенкевич Николай Анатольевич. Принципы устойчивой кооперации. Управление большими системами: сборник трудов, (3):100–120, 2009.
2. Jean-Francois Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks, 30(2):105–119, 1997.
3. V. Zakharov and M. Dementieva. Multistage cooperative games and problem of time consistency. International Game Theory Review, 6:157–170, 2004.
4. V.V. Zakharov and A.N. Shchegryaev. Multi-period cooperative vehicle routing games. Contributions to Game Theory and Management, 7(2):349– 359, April 2014.P. Stodola, J. Mazal, M. Podhorec, O. Litva. Using the Ant Colony Optimization Algorithm for the Capacitated Vehicle Routing. Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014
5. Synthesis of Active Traffic Management Experiences in Europe and the United States // FHWA, 2010. 30. Active Traffic Management for Arterials // National Cooperative Highway Research Program, 2013.
6. Inter-jurisdictional Coordination for Traffic Management in “Large City Technical Exchange and Assistance Program” // New York University, 2000.
7. Building the ITI: Putting the National Architecture into Action, Mitretek Systems, FHWA, April 1996.
8. Егунов М. М., Шувалов В. П. Анализ структурной надёжности транспортной сети // Вестник СибГУТИ. 2012. № 1. С. 54–60.
9. Холодов Я. А. и др. Моделирование транспортных потоков—актуальные проблемы и перспективы их решения // ТРУДЫ МФТИ. 2010. Т. 2. № 4. С. 152.
10. Дышленко С. Г. Маршрутизация в транспортных сетях/ Интеллектуальные ИТ в управлении//ИТНОУ. 2018. №1. – С 15–20.
11. Лёвин Б. А. Информационное моделирование при управлении транспортом // Перспективы науки и образования. 2017. № 3 (27). С. 50–54.

Приложение

*import* pandas *as* pd  
*import* numpy *as* np  
*import* matplotlib.pyplot *as* plt  
*import* seaborn *as* sns  
*import* folium  
*import* plotly.express *as* px  
*import* osmnx *as* ox  
*import* networkx *as* nx  
*import* random  
*import* math  
*from* typing *import* Dict, List  
  
*# Класс AntCVRP  
class* AntCVRP:  
 A = 1.04 *# Весовой коэффициент для феромона* B = 1.5 *# Весовой коэффициент для эвристической информации* U = 4 *# Параметр для обновления феромона* P = 0.74 *# Параметр для обновления феромона  
  
 def \_\_init\_\_*(*self*, start\_node: *int*, capacity: *int*, graph):  
 *"""  
 Инициализация муравья.* ***:param*** *start\_node: Начальный узел.* ***:param*** *capacity: Грузоподъемность муравья.* ***:param*** *graph: Граф.  
 """  
 self*.start\_node = start\_node  
 *self*.cur\_node = start\_node  
 *self*.cur\_capacity = capacity  
 *self*.max\_capacity = capacity  
 *self*.graph = graph  
 *self*.nodes\_to\_visit: Dict[*int*, *int*] = {} *# Узлы, которые нужно посетить  
  
 def* init(*self*, nodes\_to\_visit: Dict[*int*, *int*]):  
 *"""  
 Инициализация муравья перед началом маршрута.* ***:param*** *nodes\_to\_visit: Узлы для посещения.  
 """  
 self*.nodes\_to\_visit = nodes\_to\_visit.copy()  
  
 *def* choose\_next(*self*) -> *int*:  
 *"""  
 Выбор следующего узла для посещения.* ***:return****: Следующий узел.  
 """* max\_node = 0  
 total\_sum = 0  
 possible\_nodes: Dict[*int*, *int*] = {}  
  
 *# Фильтрация узлов, которые можно посетить с учетом грузоподъемности  
 for* node *in self*.nodes\_to\_visit:  
 *if self*.graph.nodes[node].get('demand', 0) <= *self*.cur\_capacity:  
 possible\_nodes[node] = node  
 total\_sum += *self*.h\_value(node)  
  
 *if* total\_sum != 0:  
 *# Вероятностный выбор следующего узла* rand = random.random()  
 segment = 0  
 *for* node *in* possible\_nodes:  
 segment += *self*.h\_value(node) / total\_sum  
 *if* rand < segment:  
 max\_node = node  
 *break  
  
 if* max\_node != 0:  
 *# Удаляем узел из списка для посещения и уменьшаем грузоподъемность  
 del self*.nodes\_to\_visit[max\_node]  
 *self*.cur\_capacity -= *self*.graph.nodes[max\_node].get('demand', 0)  
 *else*:  
 *# Возвращаемся на склад (начальный узел) и восстанавливаем грузоподъемность  
 self*.cur\_capacity = *self*.max\_capacity  
  
 *return* max\_node  
  
 *def* h\_value(*self*, node: *int*) -> *float*:  
 *"""  
 Вычисление значения для вероятностного выбора следующего узла.* ***:param*** *node: Узел.* ***:return****: Значение для выбора.  
 """* tau = *self*.graph.edges[*self*.cur\_node, node].get('tau', 1.0) *# Феромон* etha = 1.0 / *self*.graph.edges[*self*.cur\_node, node].get('length', 1.0) *# Эвристическая информация* value = math.pow(tau, *self*.A) \* math.pow(etha, *self*.B)  
 *return* value *if* value != 0 *else float*('inf') *# Возвращаем минимальное значение, если value == 0  
  
 def* local\_updating\_rule(*self*, path: List[*int*], length: *float*):  
 *"""  
 Локальное обновление феромона на пути.* ***:param*** *path: Пройденный путь.* ***:param*** *length: Длина пути.  
 """  
 for* i *in range*(1, *len*(path)):  
 curr\_vertex = path[i - 1]  
 next\_vertex = path[i]  
 *# Обновление феромона на ребре (curr\_vertex -> next\_vertex)* new\_tau = *self*.graph.edges[curr\_vertex, next\_vertex].get('tau', 1.0) + *self*.U / (*self*.P \* length)  
 *self*.graph.edges[curr\_vertex, next\_vertex]['tau'] = new\_tau  
 *# Обновление феромона на ребре (next\_vertex -> curr\_vertex)* new\_tau\_reverse = *self*.graph.edges[next\_vertex, curr\_vertex].get('tau', 1.0) + *self*.U / (*self*.P \* length)  
 *self*.graph.edges[next\_vertex, curr\_vertex]['tau'] = new\_tau\_reverse  
  
 *def* better(*self*, path\_value1: *float*, path\_value2: *float*) -> *bool*:  
 *"""  
 Сравнение двух путей.* ***:param*** *path\_value1: Значение первого пути.* ***:param*** *path\_value2: Значение второго пути.* ***:return****: True, если первый путь лучше.  
 """  
 return* path\_value1 < path\_value2  
  
 *def* end(*self*) -> *bool*:  
 *"""  
 Проверка завершения маршрута.* ***:return****: True, если все узлы посещены и муравей вернулся на склад.  
 """  
 return not self*.nodes\_to\_visit *and self*.cur\_node == *self*.start\_node  
  
  
*# Загрузка данных*city = "Moscow"  
opt = pd.read\_csv("McDonalds.csv") *# Загрузка данных*opt = opt[opt["city"] == city][["city", "address", "latitude", "longitude"]].reset\_index(drop=*True*) *# Фильтрация по городу*opt = opt.reset\_index().rename(columns={"index": "id", "latitude": "y", "longitude": "x"}) *# Переименование столбцов  
  
print*("Total locations:", *len*(opt))  
*print*(opt.head(3))  
  
*# Подготовка данных для визуализации*data = opt.copy()  
data["color"] = np.where(data['id'] == 0, 'red', 'blue') *# Начальная точка красная, остальные синие*start = data[data["id"] == 0][["y", "x"]].values[0]  
*print*("Starting point:", start)  
  
*# Создание карты*map = folium.Map(location=start, tiles="cartodbpositron", zoom\_start=12)  
  
*# Добавление маркеров на карту  
for* \_, row *in* data.iterrows():  
 color = "red" *if* row["id"] == 0 *else* "blue" *# Начальная точка красная, остальные синие* folium.CircleMarker(  
 location=[row["y"], row["x"]],  
 radius=5,  
 color=color,  
 fill=*True*,  
 fill\_color=color,  
 fill\_opacity=0.7,  
 popup=f"ID: {row['id']}<br>Address: {row['address']}" *# Подсказка с ID и адресом* ).add\_to(map)  
  
map  
  
*# Создание графа*G = ox.graph\_from\_point(start, dist=10000, network\_type="drive")  
G = ox.add\_edge\_speeds(G)  
G = ox.add\_edge\_travel\_times(G)  
fig, ax = ox.plot\_graph(G, bgcolor="black", node\_size=5, node\_color="white", figsize=(16, 8))  
  
*# Нахождение ближайших узлов*start\_node = ox.distance.nearest\_nodes(G, start[1], start[0])  
opt["node"] = opt[["y", "x"]].apply(*lambda* x: ox.distance.nearest\_nodes(G, x[1], x[0]), axis=1)  
opt = opt.drop\_duplicates("node", keep='first')  
*print*(opt.head())  
  
*# Функция для вычисления расстояния  
def* f(a, b):  
 *try*:  
 d = nx.shortest\_path\_length(G, source=a, target=b, method='dijkstra', weight='travel\_time')  
 *except*:  
 d = np.inf *# Используем np.inf для обозначения недостижимости  
 return* d  
  
*# Создание матрицы расстояний*distance\_matrix = np.asarray([[f(a, b) *for* b *in* opt["node"].tolist()] *for* a *in* opt["node"].tolist()])  
distance\_matrix = pd.DataFrame(distance\_matrix, columns=opt["node"].values, index=opt["node"].values)  
  
*print*(distance\_matrix.head())  
  
*# Создание тепловой карты*heatmap = distance\_matrix.copy()  
  
*# Преобразование значений для визуализации  
for* col *in* heatmap.columns:  
 heatmap[col] = heatmap[col].apply(*lambda* x:  
 0.3 *if* np.isinf(x) *else # Недостижимые узлы* (1 *if* x == 0 *else* 0.7) *# Нулевое расстояние или достижимые узлы* )  
  
*# Визуализация тепловой карты*fig, ax = plt.subplots(figsize=(10, 5))  
sns.heatmap(heatmap, vmin=0, vmax=1, cbar=*False*, ax=ax)  
plt.title("Heatmap of Travel Times")  
plt.show()  
  
*# Решение задачи маршрутизации с использованием AntCVRP*drivers = 1  
lst\_nodes = opt["node"].tolist()  
*print*("Start:", start\_node, "Total locations to visit:", *len*(lst\_nodes) - 1, "Drivers:", drivers)  
  
*# Инициализация муравья*ant = AntCVRP(start\_node, capacity=100, graph=G)  
nodes\_to\_visit = {node: node *for* node *in* lst\_nodes *if* node != start\_node}  
ant.init(nodes\_to\_visit)  
  
*# Поиск маршрута*route\_idx = []  
*while not* ant.end():  
 next\_node = ant.choose\_next()  
 route\_idx.append(next\_node)  
 ant.cur\_node = next\_node  
  
*print*("Route for driver (nodes):", route\_idx)  
  
*# Получение путей между узлами  
def* get\_path\_between\_nodes(lst\_route):  
 lst\_paths = []  
 *for* i *in range*(*len*(lst\_route)):  
 *try*:  
 a, b = lst\_route[i], lst\_route[i + 1]  
 *except*:  
 *break  
 try*:  
 path = nx.shortest\_path(G, source=a, target=b, method='dijkstra', weight='travel\_time')  
 *if len*(path) > 1:  
 lst\_paths.append(path)  
 *except*:  
 *continue  
 return* lst\_paths  
  
lst\_paths = get\_path\_between\_nodes(route\_idx)  
  
*# Визуализация маршрута на карте  
for* path *in* lst\_paths:  
 ox.plot\_route\_folium(G, route=path, route\_map=map, color="blue", weight=1)  
  
map  
  
*# Создание анимации маршрута  
def* df\_animation\_multiple\_path(G, lst\_paths, parallel=*True*):  
 df = pd.DataFrame()  
 *for* path *in* lst\_paths:  
 lst\_start, lst\_end = [], []  
 start\_x, start\_y = [], []  
 end\_x, end\_y = [], []  
 lst\_length, lst\_time = [], []  
  
 *for* a, b *in zip*(path[:-1], path[1:]):  
 lst\_start.append(a)  
 lst\_end.append(b)  
 lst\_length.append(*round*(G.edges[(a, b, 0)]['length']))  
 lst\_time.append(*round*(G.edges[(a, b, 0)]['travel\_time']))  
 start\_x.append(G.nodes[a]['x'])  
 start\_y.append(G.nodes[a]['y'])  
 end\_x.append(G.nodes[b]['x'])  
 end\_y.append(G.nodes[b]['y'])  
  
 tmp = pd.DataFrame(*list*(*zip*(lst\_start, lst\_end, start\_x, start\_y, end\_x, end\_y, lst\_length, lst\_time)),  
 columns=["start", "end", "start\_x", "start\_y", "end\_x", "end\_y", "length", "travel\_time"])  
 df = pd.concat([df, tmp], ignore\_index=(*not* parallel))  
  
 df = df.reset\_index().rename(columns={"index": "id"})  
 *return* df  
  
df = pd.DataFrame()  
tmp = df\_animation\_multiple\_path(G, lst\_paths, parallel=*False*)  
df = pd.concat([df, tmp], axis=0)  
  
first\_node, last\_node = lst\_paths[0][0], lst\_paths[-1][-1]  
df\_start = df[df["start"] == first\_node]  
df\_end = df[df["end"] == last\_node]  
  
fig = px.scatter\_mapbox(data\_frame=df, lon="start\_x", lat="start\_y", zoom=15, width=900, height=700,  
 animation\_frame="id", mapbox\_style="carto-positron")  
  
fig.data[0].marker = {"size": 12}  
  
fig.add\_trace(px.scatter\_mapbox(data\_frame=opt, lon="x", lat="y").data[0])  
fig.data[1].marker = {"size": 10, "color": "black"}  
  
fig.add\_trace(px.scatter\_mapbox(data\_frame=df\_start, lon="start\_x", lat="start\_y").data[0])  
fig.data[2].marker = {"size": 15, "color": "red"}  
  
fig.add\_trace(px.scatter\_mapbox(data\_frame=df\_end, lon="start\_x", lat="start\_y").data[0])  
fig.data[3].marker = {"size": 15, "color": "green"}  
  
fig.add\_trace(px.line\_mapbox(data\_frame=df, lon="start\_x", lat="start\_y").data[0])  
  
fig.show()