Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Работа допущена к защите
Директор ВШПИ
_____П.Д. Дробинцев
"__"_____ 2017г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: Сервис построения маршрутов внутри зданий

Направление: *09.03.01 «Фундаментальная информатика и информационные технологии»*
Профиль: *09.03.01_06 «Распределенные информационные системы»*

Выполнил студент гр. 43507 В.Б. Борисов
Руководитель, к.т.н., проф. А.В. Самочадин

Санкт-Петербург
2017

Утверждаю
Директор ВШ ПИ
_____П.Д. Дробинцев
"___"_____ 2017г.

ЗАДАНИЕ
на выпускную работу
студенту В.Б. Борисову

1. Тема: Сервис построения маршрутов внутри зданий (утверждена распоряжением по институту от _____№_____)
2. Срок сдачи работы: 13.06.17
3. Исходные данные к проекту (работе).
   - 3.1 Крук Е.А., Овчинников А.А. Методы программирования и прикладные алгоритмы: Учебное пособие. ГУАП. — СПб., 2007. 166 с.
   - 3.2 Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. — Нижний Новгород, 2007, 85 с.
   - 3.3 Окулов С.М. Программирование в алгоритмах. — М.:Бином. Лаборатория знаний, 2002. 341 с.
4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов).
   - 4.1 Обзор алгоритмов решения задачи коммивояжера
   - 4.2 Разработка наиболее оптимального алгоритма для веб-сервиса
   - 4.3 Оценка работы выбранного алгоритма
5. Перечень графического материала с точным указанием обязательных чертежей.
6. Консультанты по проекту (с указанием относящегося к ним разделов проекта, работы).

Дата выдачи задания: _____ г.
Руководитель: _____к.т.н., проф. А.В. Самочадин
Задание принял к исполнению: _____В.Б.Борисов

Реферат

С. 45. Рис. 35. Табл. 13.

Сервис построения маршрутов внутри зданий

В данной выпускной работе были рассмотрены различные подходы и алгоритмы решения задачи коммивояжера. Разработано веб-приложение, которое строит маршруты внутри зданий, при помощи генетического алгоритма. Реализована возможность загружать свои собственные карты в различных форматах, которая позволит строить маршруты не только для карт, которые есть в наличии, но и для своих собственных. Веб-приложение протестировано для ряда зданий со сложной структурой.

Ключевые слова: ЗАДАЧА КОММИВОЯЖЕРА, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ВЕБ-СЕРВИС, СЕЛЕКЦИЯ, СКРЕЩИВАНИЕ, МУТАЦИЯ, КРИТЕРИЙ ОСТАНОВА, МЕТОД ВЕТВЕЙ И ГРАНИЦ, ПОЛНЫЙ ПЕРЕБОР, МЕТОД ОТЖИГА, МЕТОД БЛИЖАЙШЕГО СОСЕДА, АЛГОРИТМ ПРИМА-ЭЙЛЕР

Abstract

45 pages, 35 figures, 13 tables

Route planning service inside buildings

In this bachelor work, various approaches and algorithms for solving the traveling salesman problem were considered. Developed a web application that builds routes inside buildings, using a genetic algorithm. It is possible to upload your own maps in various formats, which will allow you to build routes not only for maps that are available, but also for your own. The web application has been tested for a number of buildings with complex structures.

Keywords: TRAVELLING SALESMAN PROBLEM, GENETIC ALGORITHM, WEB-SERVICE, SELECTION, CROSSOVER, MUTATION, STOP`S CRITERIA, BRANCH AND BOUND, BRUTE FORCE, SIMULATED ANNEALING, K-NEAREST NEIGHBOR ALGORITHM, PRIM-EULER ALGORITHM.

Оглавление

ВВЕДЕНИЕ 7

ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ И ПОСТАНОВКА ЗАДАЧИ 8

1.1 Актуальность решения задачи построения маршрутов внутри зданий 8
1.2 Средства маршрутизации внутри зданий 9
1.3 Объект и методы исследования 14
1.4 Постановка задачи 14

АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЕРА И ИХ СРАВНИТЕЛЬНЫЙ АНАЛИЗ 15

2.1 Задача коммивояжера 15
2.2 Алгоритмы решения задачи коммивояжера 16
2.2.1 Алгоритм полного перебора 16
2.2.2 Метод ветвей и границ 17
2.2.3 Алгоритм Прима-Эйлера 19
2.2.4 Метод ближайшего соседа 23
2.2.5 Метод отжига 24
2.2.6 Генетический алгоритм 25
2.3 Выбор алгоритма 27

РЕАЛИЗАЦИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА И ИНТЕРФЕЙСА ВЕБ-СЕРВИСА 30

3.1 Выбор средств разработки 30
3.2 Особенности реализации сервиса 31
3.3 Реализация генетического алгоритма 35
3.3.1 Операция селекции (отбор) 35
3.3.2 Операция скрещивания 36
3.3.3 Операция мутации 37
3.4 Интерфейс пользователя 38

ТЕСТИРОВАНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА И ОЦЕНКА РЕЗУЛЬТАТОВ 40

4.1 Тестирование интерфейса сервиса 40
4.2 Тестирование основных функций сервиса 42

ЗАКЛЮЧЕНИЕ 47

СПИСОК ЛИТЕРАТУРЫ 48

Список обозначений

ГЛОНАСС — глобальная навигационная спутниковая система
СГП — система глобального позиционирования
ПО — программное обеспечение
ТРЦ — торгово-развлекательный центр
ВГУЭС — Владивостокский государственный университет экономики и сервиса
ГУАП — Санкт-петербургский государственный университет аэрокосмического приборостроения
АПП — алгоритм полного перебора
МВиГ — метод ветвей и границ
МБС — метод ближайшего соседа
ГА — генетический алгоритм
HTML — HyperText Markup Language
CSS — Cascading Style Sheets
JS — JavaScript
ОМД — объектно-ориентированная модель

Введение

Многоэтажные здания со сложной структурой и планировкой, такие как аэропорты, торговые центры и музеи, посещает большое количество людей. Зачастую ориентироваться в подобных помещениях могут те, кто постоянно посещает такие здания, чего нельзя сказать о людях, которые находятся в них в первый раз. Поэтому возникает потребность реализовать систему, которая могла бы помочь всем желающим в решении проблемы маршрутизации внутри здании.

Такие системы, как глобальная навигационная спутниковая система (ГЛОНАСС) и система глобального позиционирования (СГП) успешно решают проблему маршрутизации вне зданий. Воспользовавшись данными технологиями, можно не только узнать о ближайших кафе, ресторанах, гостиницах, но и проложить маршрут к интересующим местам. Однако даже такие системы несовершенны. Существенный их недостаток — это ограниченная область применения. Данные технологии весьма трудно использовать внутри зданий, а зачастую практически невозможно.

Благодаря сервисам навигации внутри зданий люди смогут без каких-либо трудностей найти зал в музее, ближайший банкомат, аудиторию или полку с нужным товаром в магазине (больше не придется тратить время на поиск нужных товаров), свободное место на парковке и многое другое. Поэтому целью данной выпускной работы, является разработка веб-сервиса, который строит маршруты внутри зданий.

Выпускная работа содержит 4 основных раздела. В разделе 1 представлена актуальность проблемы. В разделе 2 содержится краткий обзор алгоритмов построения маршрутов и их сравнение при заданных условиях. В разделе 3 описан интерфейс и описание веб-приложения. В разделе 4 представлены результаты тестирования и проверки работоспособности веб-приложения.

Глава 1

Обзор существующих решений и постановка задачи

1.1 Актуальность решения задачи построения маршрутов внутри зданий

Изначально задача коммивояжера была направлена на то, чтобы построить выгодный маршрут между городами с возвратом в начальный город при условии, что посещать заданные города можно не более одного раза. Решение подобной задачи может, в значительной степени, помочь курьеру, которому необходимо доставить несколько товаров в разные города. Имея выгодный маршрут, курьер сможет за кратчайшие сроки доставить товар в нужный город, потратив к тому же меньше топлива и вернуться обратно. В современных реалиях задачу странствующего торговца можно решать и внутри зданий, имея при себе мобильный телефон, планшет или любое другое электронное устройство.

Если для посетителей, которые первый раз посетили музей или торгово-развлекательный комплекс (ТРЦ), решение проблемы маршрутизации обеспечит экономию времени, то для персонала это даст возможность грамотно организовать поток людей внутри здания. Для предприятий открывается возможность оптимизировать бизнес-процессы, а для продавцов, например, обнаружить факт приближения потенциального покупателя. Предположим, у нас есть торговый центр и написанное для него мобильное приложение, в котором регистрируются люди. При приближении покупателя к какому-нибудь магазину, программное обеспечение (ПО) получает уникальный идентификатор клиента, анализирует предыдущие покупки и, на основании полученной информации, может сделать уникальное предложение покупателю со скидкой и многое другое.

Приводя подобные примеры работы сервисов, уже можно с уверенностью сказать, что у данного направления есть большой потенциал и от решения данной задачи выигрывают все и решать её можно и нужно для зданий, которые посещает большое количество людей.

1.2 Средства маршрутизации внутри зданий

Благодаря большим коммерческим перспективам, разработки в области маршрутизации являются востребованными. Многие компании это понимают, и уже сейчас крупные корпорации, такие как Google и Apple, участвуют либо в создании новых приложений, либо в расширении тех, которые уже существуют (например, Google Maps).

В теории задача маршрутизации может решаться для любого здания, какой бы сложной структурой и планировкой оно не обладало. Существуют различные методы, которые применяются в навигации внутри помещении. В частности, компания Google в 2011 году, обновив приложение Google Maps, ввела возможность для ряда стран, в том числе и для России, добавлять поэтажные планы общественных строений [1,2]. К таким строениям относятся вокзалы, аэропорты, торговые центры и другие объекты. Пользователи могут сами добавлять в базу данных планы зданий, и после успешной проверки план появится (рис. 1.1). Данный сервис работает для всех известных платформ [3].

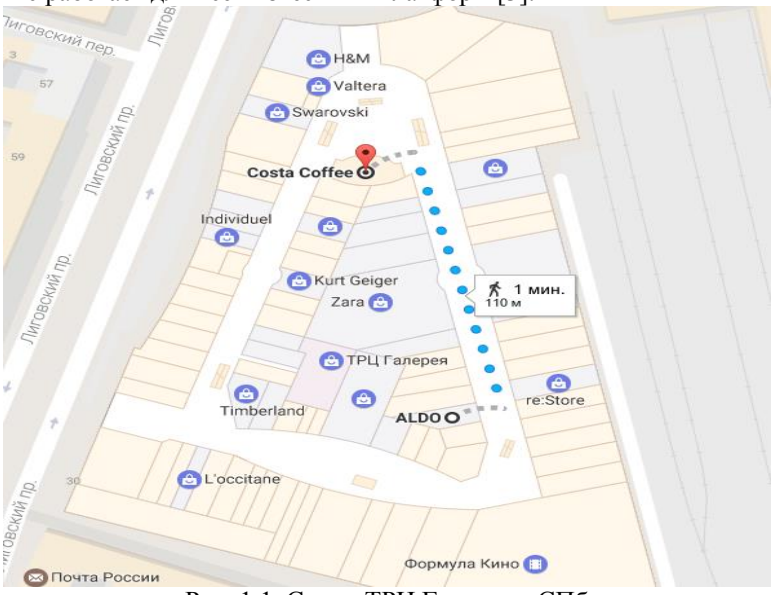


Рис. 1.1. Схема ТРЦ Галерея в СПб

Маршрутизацией внутри зданий занимаются не только крупные и известные компании, но и другие публичные организации. Как это часто бывает, та или иная компания разрабатывает свое ПО для конкретного типа зданий, а не для всех подряд, так как это очень трудоемко и дорого. Были рассмотрены лишь некоторые проекты, которые схожи по своей

структуре и назначению (приложения для аэропортов, торговых центров и университетов).

Первая популярная категория — приложения для аэропортов. Ярким представителем является приложение-ассистент в путешествиях — GateGuru (рис. 1.2). Помимо навигации внутри аэропортов, мобильное приложение предоставляет планировщик перелётов с функцией напоминания, различные каталоги, возможность арендовать машину и прочее [4]. Пассажиры, установив приложение на свой телефон, получают возможность определить свое местоположение внутри аэропорта и проложить маршрут до точки интереса, будь это паспортный контроль, стойка регистрации или зал получения багажа. Приложение также сможет уведомлять пассажиров о начале и завершении регистрации на рейс [5]. Как итог довольными остаются и пассажиры, и работники аэропорта. Пассажиры быстро получают информацию и строят маршрут, а работники, за счет оптимизации потоков пассажиров, могут с комфортом выполнять свою работу, не боясь давок и эксцессов.

Рис. 1.2. Экран плана аэропорта Washington Dulles в приложении GateGuru для Android.

Следующая категория приложений — приложения для торговых центров. Идея таких приложений очень схожа с приложениями для аэропортов. Цель данного ПО заключается в том, чтобы потребитель мог в любой момент времени найти нужный магазин, узнать всю нужную ему информацию (например, о наличии и стоимости товара), построить маршрут от своего местоположения до магазина. Также оно помогает владельцам магазина завлечь покупателя с помощью предоставления информации о специальных предложениях и скидках. Примерами таких приложений являются: Point Inside, Wizzy Shopper, Shopkick. [5]. Иллюстрация работы одного из них можно посмотреть на рис.1.3.

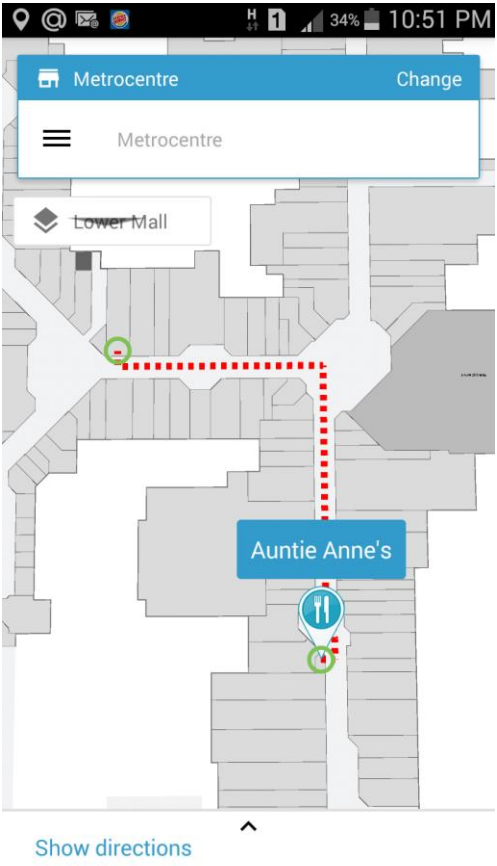


Рис. 1.3. Работа приложения для ТРЦ в Point Inside

Приложения для университетов позволяют студентам и преподавателям находить нужное место, определять свое местоположение и строить маршрут от одной точки к другой, с учетом этажей и лестниц. В качестве примера можно привести разработку Владивостокского государственного университета экономики и сервиса (ВГУЭС) — Nav-In [6] и государственного университета аэрокосмического приборостроения (ГУАП) — SUAI navigation [7]. Рассмотрим работу этих двух приложении. Пусть приложение Nav-In будет строить маршрут только для первого этажа (рис. 1.4), а приложение SUAI navigation будет строить маршрут на нескольких этажах. Результат на рис. 1.5 и рис. 1.6. Стоит отметить, что оба проекта могут строить маршрут как для одного этажа, так и для несколько этажей.

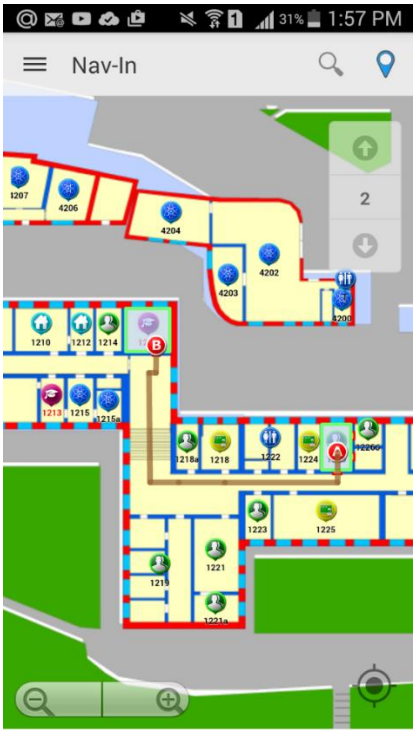


Рис. 1.4. Пример построения маршрута на первом этаже внутри кампуса ВГУЭС

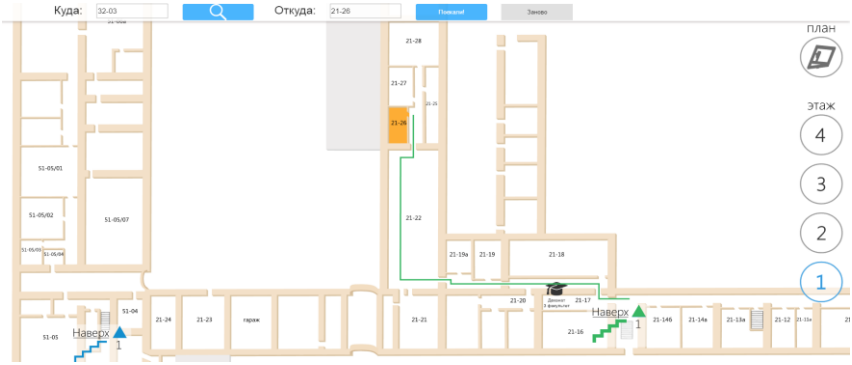


Рис. 1.5. Построение маршрута от начальной точки до ближайшей лестницы

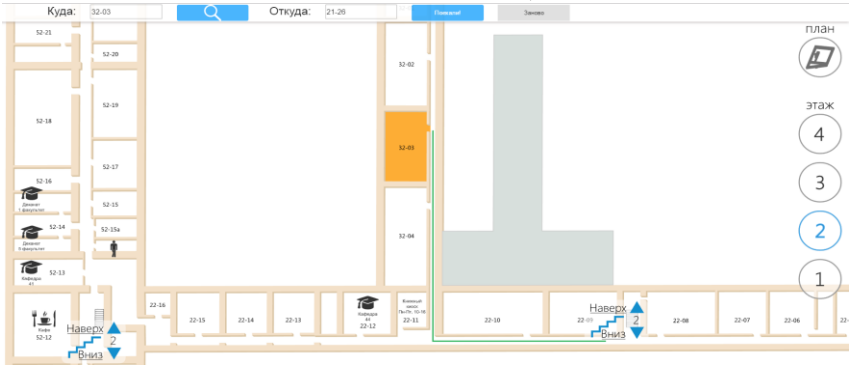


Рис. 1.6. Продолжение маршрута от лестницы до конечной точки

Несмотря на большое количество различных приложений и их реализаций, практически все они решают одну и ту же задачу, а именно строят маршрут внутри конкретного здания не более, чем для двух точек. ПО строит маршрут только для двух точек с учетом стен, этажей и конструкции здания. Общей проблемой этих приложений является невозможность построить маршрут, связывающий больше, чем две точки. Такие ситуации, когда человек должен побывать не в одном месте, а в нескольких, бывает часто. Например, если посетитель пришел в музей, то он захочет побывать в нескольких залах. Предположим, он захочет посетить 10 залов. Воспользовавшись приложениями, рассмотренными в данной главе, посетителю придется 10 раз строить маршрут от одной точки к другой, что не очень удобно.

1.3 Объект и методы исследования

В данной работе объектами являются точки, которые нужно посетить не более 1 раза и общий путь, проходящий через заданные точки. Для решения задачи маршрутизации необходимо сначала представить точки и всевозможные пути в виде математической модели, например, в виде взвешенного графа $G(V, E)$,где $V = \{v_0, v_1, \dots v_n\}$— множество вершин, а E — множество ребер $\{(v_i, v_j), i \neq j\}$ (рис. 1.7), то есть вершины графа будут соответствовать точкам интереса, которые хочет посетить человек, а ребра (i, j) будут являться путем между вершинами i и j.

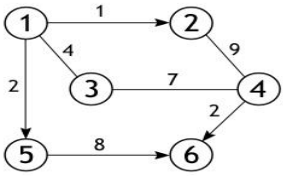


Рис. 1.7. Взвешенный граф

Каждому ребру(i, j) сопоставим расстояние $C_{i,j} \geq 0$,где C — матрица неотрицательных расстояний между точками интереса. Каждый маршрут будет характеризоваться длиной маршрута, которая представляет собой сумму, составляющих его ребер

1.4 Постановка задачи

Целью данной работы является разработка веб-сервиса для построения маршрутов внутри зданий. Ограничением будет являться количество вершин, участвующих в обходе. Их количество не должно превышать 100 вершин. В соответствии с поставленной целью необходимо решить следующие задачи:

1. Анализ существующих проектов и приложений, направленных на построение маршрутов внутри зданий.
2. Исследование алгоритмов, решающих задачу коммивояжера и сравнение их друг с другом по производительности и качеству.
3. Выбор алгоритма, который будет строить маршрут при заданных ограничениях.

Глава 2

Алгоритмы решения задачи коммивояжера и их сравнительный анализ

2.1 Задача коммивояжера

Задача коммивояжера представляет собой задачу отыскания кратчайшего гамильтонова пути в полном конечном графе с *N* вершинами [8]. Гамильтонов путь — такой путь, который проходит через каждую вершину графа один раз.

Находясь в точке, коммивояжер встает перед выбором следующей точки, которую он еще не посетил, всего существует $(n-1)!$ маршрутов, где n — количество вершин [9]. Формальное описание представлено ниже.

$$\begin{cases} Q(x) = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}\, x_{ij} \to min \\ \sum_{i=1}^{N} x_{ij} = 1, \forall j = 1,\acute{}N \\ \sum_{j=1}^{N} x_{ij} = 1, \forall i = 1,\acute{}N \\ x_{ij} \in \{0,1\} \end{cases} \quad (2.1.1)$$

где $Q(x)$ — функция критерия оптимальности, N — количество вершин в графе, c_{ij} — вес ребра, x_{ij} — двоичная переменная, которая сопоставляется каждому ребру, отвечающая за наличие перехода из i-вершины в j-вершину, если 1,то ребро принадлежит маршруту, если 0, то не принадлежит.

2.2 Алгоритмы решения задачи коммивояжера

Для решения задачи коммивояжера существуют следующие основные алгоритмы:

1. алгоритм полного перебора
2. метод ветвей и границ
3. алгоритм Прима-Эйлера
4. метод ближайшего соседа
5. генетический алгоритм
6. метод отжига

2.2.1 Алгоритм полного перебора

Одним из самых простых алгоритмов является алгоритм полного перебора (АПП). Пусть у нас будет n-вершин, тогда количество перестановок будет равняться $n!$ [10]. Так как задача является симметричной, то есть все пары ребер между одинаковыми вершинами имеют одну и ту же длину $c_{ij} = c_{ji}$ и начальная точка остается неизменной, то количество перестановок равняется $\frac{(n-1)!}{2}$.

Ссылаясь на требование о том, что количество точек не должно превышать значения 100 можно оценить количество перестановок. Оно будет иметь порядок 10^{157}. В результате получаем огромное число перестановок. Скорость роста факториала можно увидеть в таблице 2.1.

Таблица 2.1

Факториал	5	10	25	50	75	100
Значение	120	10^6	10^{25}	10^{64}	10^{109}	10^{157}

Плюсом этого алгоритма можно назвать точное решение задачи коммивояжера. Однако при большом количестве вершин продолжительность вычислений может занять непозволительно много времени, либо задача может вообще не решиться. Отсюда можно сделать вывод, что данный метод можно использовать только для задач малой размерности и стоит рассматривать другие алгоритмы, которые могут находить оптимальный маршрут.

2.2.2 Метод ветвей и границ

Еще один метод, который находит оптимальный маршрут — это метод ветвей и границ (МВиГ). По сути МВиГ является развитием АПП. Главным же отличием является наличие таких процедур как ветвление и нахождение оценок.

Работу алгоритма проще описать в несколько шагов:

1. По заданному графу составляем матрицу стоимости. Все элементы диагонали приравниваем к бесконечности.
2. В построенной матрице, ищем в каждой строке минимальный элемент и вычитаем его из всех элементов строки.
3. Такую же процедуру делаем для столбцов, элементы которых больше нуля. Получаем такую матрицу стоимости, где каждая строка и каждый столбец содержат хотя бы один нулевой элемент.
4. Для всех нулевых элементов рассчитываем коэффициент, который равен сумме наименьшего элемента строки и наименьшего элемента столбца. Далее, из всех посчитанных коэффициентов, выбираем максимальный. В гамильтонов контур вносится соответствующая дуга.
5. Удаляем строку и столбец, которые пересекают выбранный нуль.
6. Если присутствуют точки возврата, то меняем их значения на максимальное.
7. Повторяем шаги с 2-6 пока не останется матрица размерностью 2x2.
8. Получаем гамильтонов контур

Процедура ветвления заключается в том, чтобы последовательно разбить множество допустимых значений на подобласти меньших размеров, а процедура нахождения оценок заключается в поиске верхних и нижних границ. Обычно верхней границей является значение целевой функции на некотором допустимом решении, если допустимое решение дает наименьшую верхнюю границу, то оно называется рекордом. Если нижняя граница не меньше, чем верхняя, то она отбрасывается, в противном случае происходит смена рекорда и так до тех пор, пока не будут просмотрены все элементы [11].

Рассмотрим работу данного алгоритма на конкретном примере. Пусть есть некоторый граф, который представлен матрицей стоимости (Таблица 2.2).

Таблица 2.2

	1	2	3	4
1	∞	5	11	9
2	10	∞	8	7
3	7	14	∞	8
4	12	6	15	∞

Для начала находим минимальное значение в каждой строке. Найденные ячейки отмечены в Таблице 2.3.

Таблица 2.3

	1	2	3	4
1	∞	5	11	9
2	10	∞	8	7
3	7	14	∞	8
4	12	6	15	∞

Произведем редукцию строк, то есть вычтем из каждой строки минимальное значение. Результат в таблице 2.4.

Таблица 2.4

	1	2	3	4
1	∞	0	6	4
2	3	∞	1	0
3	0	7	∞	1
4	6	0	9	∞

В итоге в каждой строке будет хотя бы одна нулевая клетка. Далее находим минимальное значение в каждом столбце и делаем редукцию столбцов. Результат в таблице 2.5.

Таблица 2.5

	1	2	3	4
1	∞	0	5	4
2	3	∞	0	0
3	0	7	∞	1
4	6	0	8	∞

В итоге в каждом столбце будет хотя бы одна нулевая клетка.

Теперь, после редукции столбцов и строк, для каждой нулевой клетки находим оценку, значение которой равно сумме минимального элемента в строке и минимального элемента в столбце, в которых размещена данная клетка. При этом сама она не учитывается. Полученную оценку запишем в скобках напротив нуля. Результат в таблице 2.6.

Таблица 2.6

	1	2	3	4
1	∞	0(4)	5	4
2	3	∞	0(5)	0(1)
3	0(4)	7	∞	1
4	6	0(6)	8	∞

Выбираем клетку с максимальной оценкой, заменяем её на «∞» и выписываем путь, то есть (4->2). Та строка и столбец, где есть две «∞» вычеркиваем, а для того чтобы не возвращаться ставим в клетку, которая соответствует обратному пути еще одну «∞». Результат в таблице 2.7

Таблица 2.7

	1	2	3	4
1	∞	0(4)	5	4
2	3	∞	0(5)	∞
3	0(4)	7	∞	1
4	6	∞	8	∞

Когда все отрезки пути будут найдены, то останется лишь соединить их между собой и посчитать длину всего пути. В данном примере траектория пути получается: 4->2->3->1->4, а сам общий путь равен 30.

2.2.3 Алгоритм Прима-Эйлера

Следующий на очереди — это алгоритм Прима-Эйлера. Сам метод последовательно использует два алгоритма: на первом этапе — это алгоритм Прима, а на втором этапе — это алгоритм Эйлера. Рассмотрим их отдельно.

Алгоритм Прима.

Текущий алгоритм постепенно строит минимальное остовное дерево, то есть такое дерево, которое состоит из минимального подмножества ребер графа, где из любой вершины можно попасть в любую другую [12]. В качестве входных данных подается взвешенный неориентированный граф (рис. 2.1).

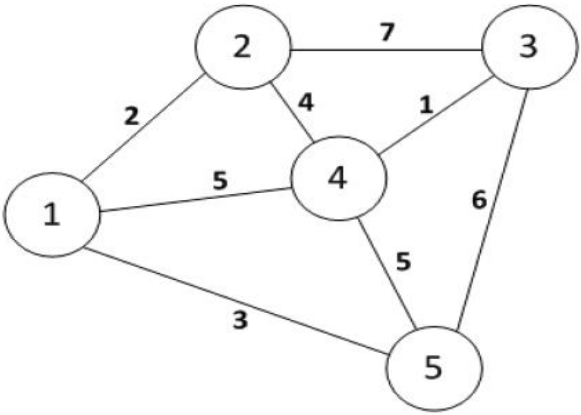
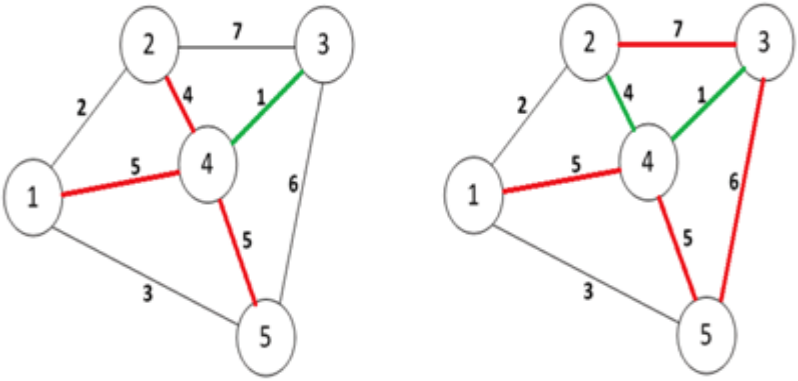


Рис. 2.1. Взвешенный неориентированный граф

Далее выбираем произвольную вершину, ищем минимальное ребро, исходящее из этой вершины и добавляем в остов. Этот алгоритм работает до тех пор, пока не будут выбраны все вершины. В итоге будет построено минимальное остовное дерево.

В качестве примера возьмем граф, изображенный на рис. 2.1. Построим для него минимальное остовное дерево (рис. 2.2). Стартовая вершина имеет номер 4.



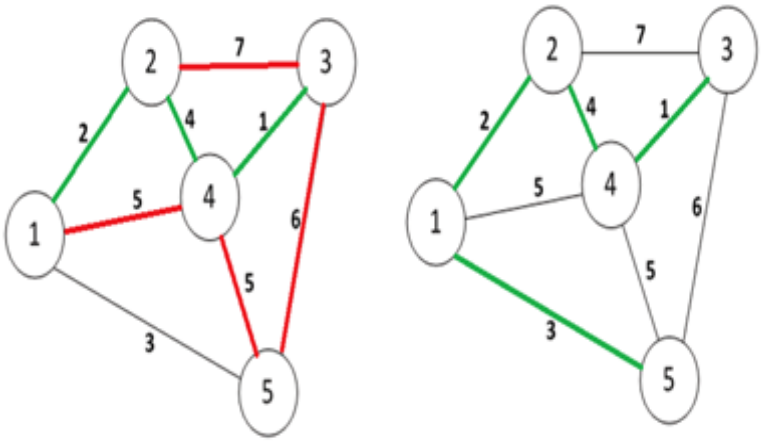


Рис. 2.2. Работа алгоритма Прима

После построения минимального остовного дерева наступает очередь второго алгоритма.

Алгоритм Эйлера.

Алгоритм берет результат предыдущего этапа и строит каркас. (рис. 2.3)

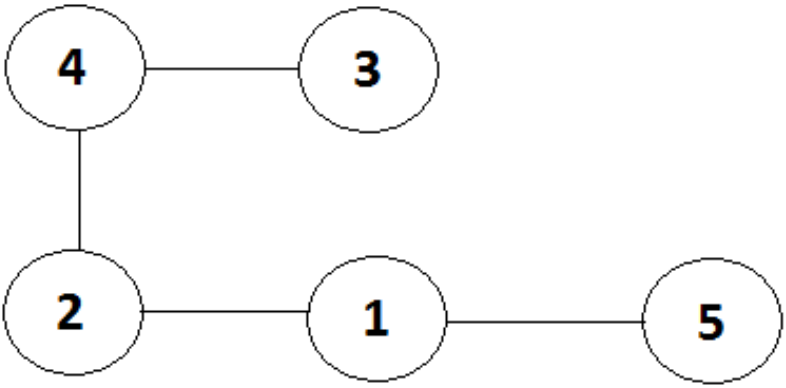


Рис. 2.3. Каркас минимального остова

Далее необходимо преобразовать каркас в эйлеров граф, который содержит цикл, проходящий через каждое ребро графа ровно один раз (рис. 2.4).

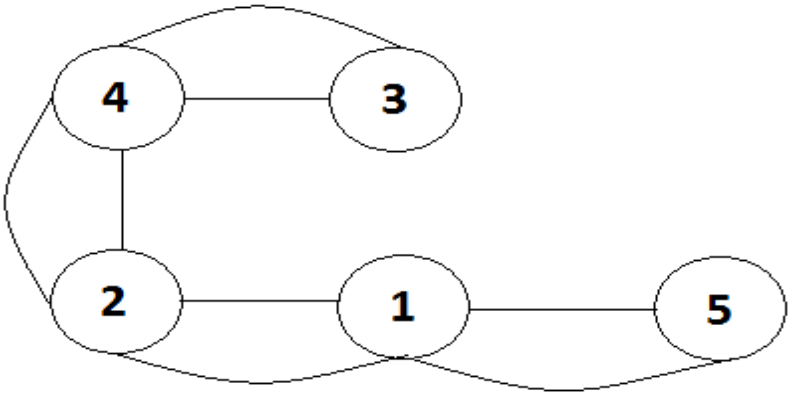


Рис. 2.4. Эйлеров граф

Цикл проходит каркас в следующем порядке: 1-2-4-3-4-2-1-5-1. После того как построили эйлеров граф нужно привести его в замкнутый граф, содержащий гамильтонов цикл [13]. Результат представлен на рис. 2.5.

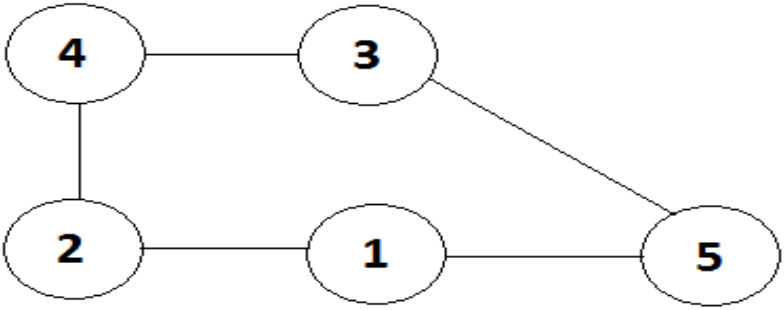


Рис. 2.5. Гамильтонов цикл

Проход гамильтонова цикла: 1-2-3-4-5.

2.2.4 Метод ближайшего соседа

Следующий алгоритм — метод ближайшего соседа (МБС). Данный алгоритм последовательно добавляет вершины в итоговый маршрут, начиная с начальной, которая по умолчанию является текущей. Далее в маршрут добавляется та вершина, которая находится ближе к текущей при условии, что маршрут не содержит ее. Добавленная вершина становится текущей. Этот алгоритм работает до тех пор, пока маршрут не будут содержать все вершины [16] (рис. 2.6.).

Рис. 2.6. Схема метода ближайшего соседа

Существует модификация данного алгоритма, которая, по предположению, должна работать быстрее обычного. Модификация содержит n-итерации. Для каждой вершины работает МБС. Как итог мы имеем несколько наборов маршрутов, которые необходимо преобразовать, чтобы они содержали нужную нам стартовую вершину [17].

Пусть у нас есть граф из 5 вершин, для каждой из них применяем МБС и получаем набор маршрутов:

1 — 1,2,3,4,5
2 — 2,1,3,4,5
3 — 3,2,4,1,5
4 — 4,3,5,2,1
5 — 5,4,3,2,1

Далее преобразовываем эти маршруты с нужной стартовой вершиной (например, 1):

1,2,3,4,5,1
1,3,4,5,2,1
1,5,3,2,4,1
1,4,3,5,2,1
1,5,4,3,2,1

Из получившихся маршрутов выбираем кратчайший.

2.2.5 Метод отжига

Следующий алгоритм — метод отжига. Алгоритм имитирует некий биологический процесс, а именно кристаллизацию вещества. В ходе этого процесса нагревают элемент до некоторой температуры, а затем начинается медленное охлаждение. Процесс завершается, когда температура падает до заранее заданного значения. Задача отжига — привести систему в состояние с наименьшей энергией. Чем ниже уровень энергии, тем меньше дефектов у кристаллической решетки [18].

В этом методе происходит минимизация энергии и эту задачу можно легко свети к минимизации маршрута. Дело в том, что данный метод схож с методом градиентного спуска, за одним исключением. Пусть есть целевая функция, которая $F(x) \leq F(\dot{x})$, где x— это элемент последовательности, а $\dot{x}$ — сосед, выбранный наугад. В методе градиентного спуска наличие такого соседа было бы отвергнуто, а в имитации отжига допускается добавление такого соседа в последовательность с вероятностью p, которая зависит от того на сколько плохо сосед ухудшил функцию. Вероятность p рассчитывается по следующей формуле:

$$p = e^{\frac{-\Delta F}{\Theta}}, (2.2.7.1)$$

где $\Delta F = F(\dot{x}) - F(x)$, e — число, больше единицы (обычно берут $e \backsimeq 2{,}7$), Θ — некоторое положительное число.

Для того, чтобы свести имитацию отжига к задаче коммивояжера, необходимо определить две функции: целевую функцию и функцию, порождающую новое состояние. Задача коммивояжера стремится минимизировать расстояние, следовательно, наша целевая функция будет иметь вид:

$$F(s_i) = \left[\sum_1^{N-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2}\right] + \\ + \sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}, \quad (2.2.5.2)$$

где $\boldsymbol{N}$— множество всех городов, $\boldsymbol{s_i}$— состояние на $\boldsymbol{i}$-шаге алгоритма $s_i \in S$, а (x_i, y_i) координаты точки.

Данная формула является суммой Евклидовых расстояний между парой точек. Так как по условию наша задача является замкнутой, то необходимо было добавить $\sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}$, что является расстоянием между последним и первым городом.

2.2.6 Генетический алгоритм

Последний рассматриваемый алгоритм — генетический алгоритм (ГА). Данный алгоритм является эвристическим алгоритмом поиска, который используют для решения задач оптимизации. Работу данного алгоритма можно сравнить с эволюцией в природе. Существуют основные шаги алгоритма, которые представлены ниже (рис. 2.7).

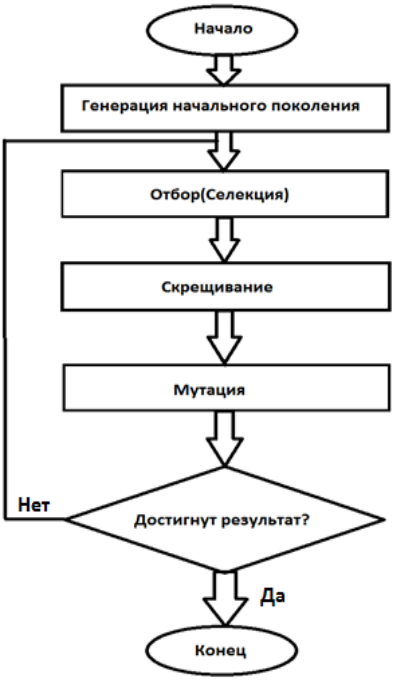


Рис. 2.7. Шаги алгоритма

Как видно из блок-схемы, основными операциями являются: селекция, скрещивание и мутация, пришедшие из биологии. Данную модель можно легко свести к задаче коммивояжера. Пусть каждое поколение будет представлять из себя маршрут, содержащий набор неповторяющихся геномов, где каждый ген соответствует узлу графа.

Введем понятие «качественная характеристика поколения», которое будет являться суммарным весом расстояний между узлами, то есть длина пути от начальной до конечной вершины. Начальное поколение создается случайным образом, а значит, что с большой вероятностью оно будет неоптимальным. Каждое дальнейшее поколение получается из предыдущего путем применения операции селекции, скрещивания и мутации [15].

Операция селекции отбирает только лучшие поколения (маршруты) для создания нового поколения. Чтобы произвести новое поколение нужно 2 родителя, поэтому в процессе отбора остается два лучших маршрута, которые будут являться родительскими. Также данная операция отбрасывает маршруты с одинаковой качественной характеристикой.

Существуют различные методы селекции:

1. Турнирная селекция. При турнирной селекции выбирается группа особей ($N \geq 2$), выбранные случайно. Далее выбирается особь с лучшим значением. Основные преимущества: нет преждевременной сходимости и стагнации, не требуется глобальное переупорядочивание и не требуется явное вычисление функции пригодности.
2. Метод рулетки. Является основным методом селекции. При использовании рулетки каждой особи ставится сектор колеса рулетки, размер которого обратно пропорционален длине маршрута и вычисляется по следующей формуле:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}, (2.2.5.1)$$

где p_i — вероятность выбора i особи, f_i— значение функции приспособленности для i особи, а N— количество особей в популяции. В теории данный метод является медленным.

3. Метод ранжирования. Вероятность выбора особи рассчитывается по формуле:

$$p_i = \frac{1}{N}\left(a - (a-b)\frac{i-1}{N-1}\right), (2.2.5.2)$$

где $a \in [1,2]$, $b = 2 - a$

Недостатком ранжирования является накладные расходы и трудность анализа сходимости.

4. Равномерное ранжирование. Вероятность выбора особи рассчитывается выражением:

$$p_i = \begin{cases} \frac{1}{\mu}, 1 \leq i \leq \mu \\ 0, \mu \leq i \leq N \end{cases},$$

где $\mu \leq N$

После отбора лучших маршрутов наступает этап скрещивания. На данном этапе происходит создание нового поколения путем скрещивания двух родительских маршрутов.

Последний этап — мутация. Данная операция очень похожа на размножение. Берется какое-то количество особей и случайным образом изменяет гены. Этот этап необходим для того, чтобы избежать преждевременную сходимость.

2.3 Выбор алгоритма

В подразделе 2.2 были рассмотрены различные алгоритмы решения задачи коммивояжера. Исходя из результатов таблицы 2.1, был сделан вывод, что АПП может применяться только для задач малой размерности и поэтому в тестировании он принимать участия не будет. Оставшиеся алгоритмы были протестированы для заданных ограничений. Перед сравнением всех оставшихся алгоритмов следует обратить внимание на ГА и МБС. У ГА есть разные виды селекции, а у МБС есть способ, который работает только для одной стартовой вершины и способ, работающий для *n*-вершин. Разные методы могут решать задачу за разное время. В связи с этим необходимо протестировать алгоритмы ГА, МВиГ и МБС независимо. Для выполнения тестирования для каждого алгоритма было сгенерировано 60 графов для разного количества вершин (соответственно 10 графов для 5, 10, 25, 50, 75 и 100 вершин). После того как каждый алгоритм найдет все кратчайшие пути для заданного количества вершин, будет вычислено среднее арифметическое и в результате получим среднее время работы каждого алгоритма для всех вершин.

Вначале будем сравнивать генетические алгоритмы с разными методами селекции. Для теста были выбраны следующие методы селекции: турнир и рулетка. Данные методы являются наиболее популярными в решении задач оптимизации, а также оба метода частично решают проблему преждевременной сходимости. При тестировании оба метода будут находиться в равных условиях и работать на одной машине. Результат тестирования методов был отображен в таблице 2.8.

Таблица 2.8. Сравнение двух методов селекции генетического алгоритма

Вершины/Время(сек.)	Рулетка	Турнир
5	0,844	0,96
10	1,07	2,1
25	3,9	4,7
50	4,1	5,2
75	6,8	7,6
100	10,8	12,4

Исходя из составленной таблицы можно сказать, что метод рулетки работает на 0.2 – 1.6 секунды быстрее, чем турнирный метод, поэтому для дальнейшего тестирования будет использован именно метод рулетки.

После тестирования генетических алгоритмов, протестируем МБС. После тестирования МБС была составлена таблица 2.9.

Таблица 2.9. Сравнение методов ближайшего соседа

Вершины/Время(сек.)	K = 1	K = n
5	2,1	1,6
10	2,52	1,8
25	6,03	5,04
50	8,633	6,907
75	17,33	13,869
100	31,93	25,55

Исходя из результатов таблицы 2.9 можно сделать вывод, что при увеличении количества узлов графа способ, использующий лишь одну стартовую вершину, работает приблизительно на 40 % хуже, чем способ, который использует все вершины поочередно. Поэтому в дальнейшем тестировании будет участвовать тот метод, где для каждой вершины работает МБС, а не для одной. Далее будет протестирован МВиГ, результат которого в таблице 2.10.

Таблица 2.10. Тестирование МВиГ

Вершины	Время
5	0,001
10	0,02
15	0,5
20	2,6
25	12,2
30	32,889

Согласно таблице 2.10 МВиГ можно использовать для решения задачи коммивояжера, только для вершин $n \leq 25$, в противном случае алгоритм будет работать долго, особенно если вершин будет 100.

Руководствуясь результатами предыдущих трех тестов, в итоговом тестировании будут участвовать: алгоритм Прима-Эйлера, генетический алгоритм с селекцией рулетки, метод имитации отжига и модифицированный метод ближайшего соседа. Результат работы всех алгоритмов представлен в таблице 2.11 ниже.

Талица 2.11. Сравнение всех алгоритмов

Вершины/ Время(сек.)	Генетический(рулетка)	Отжиг	Ближайший сосед(K=n)	Прим-Эйлер
10	1,07	2	1,8	3,4
25	3,9	6,5	5,04	6,2
50	4,1	17,2	6,907	16,4
75	6,8	43,1	13,869	38,6
100	10,8	102,3	25,55	94,3

Согласно таблице 2.11 наиболее эффективными оказались 2 алгоритма: генетический и метод ближайшего соседа, работающий для всех вершин. В целом отлично подходят для решения задачи коммивояжера с заданными требованиями. Однако, если сравнить два этих алгоритма, то можно увидеть, что ГА работает в ≈1.5 раза быстрее, чем МБС. Остальные алгоритмы, а это Прим-Эйлер и имитация отжига на вершинах от 50 и выше работают значительно дольше, по сравнению с двумя другими.

Глава 3

Реализация генетического алгоритма и интерфейса веб-сервиса

3.1 Выбор средств разработки

Для реализации веб-сервиса построения маршрутов был выбран язык программирования JavaScript (JS). Такой выбор обусловлен тем, что сам сервис будет храниться в сети интернет, а JS легко позволяет создать скрипты и сценарии. Вся структура сервиса, а именно текст, формы и слои, будут описаны при помощи языка гипертекстовой разметки (HTML), именно его будет отображать и обрабатывать браузер в первую очередь. Также в данной разработке присутствуют каскадные таблицы стилей (CSS). В данной разработке они нужны не только для того чтобы придать сервису индивидуальность и красивый внешний вид, но, и чтобы можно было избавить HTML-документ от лишних нагрузок и строчек кода. Это увеличит гибкость документа и браузеру будет легче его обрабатывать, а сами таблицы можно будет повторно использовать для других документов.

Для того чтобы воспользоваться веб-сервисом, пользователю понадобится доступ в интернет и любой браузер. Сделать запрос к серверу при помощи доменного имени и получить сервис для построения маршрутов внутри зданий. Данная процедура изображена на рис. 3.1.

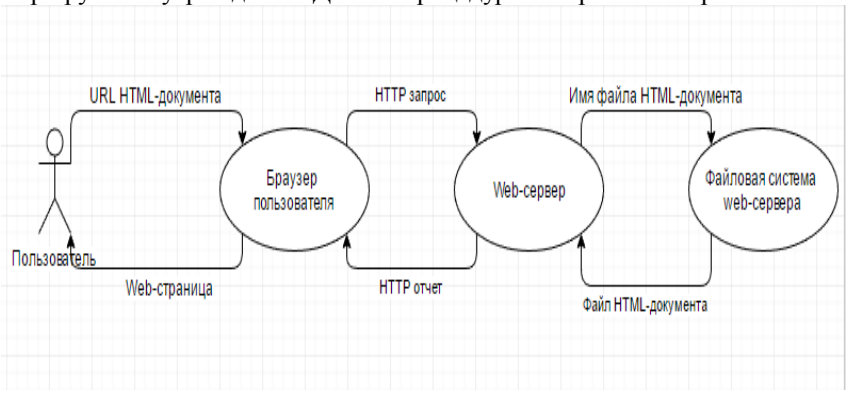


Рис. 3.1. Взаимодействие пользователя, браузера и веб-сервера

3.2 Особенности реализации сервиса

Файловую систему и основную структуру проекта можно представить в виде диаграмм на рис.3.2 и рис. 3.3 соответственно.

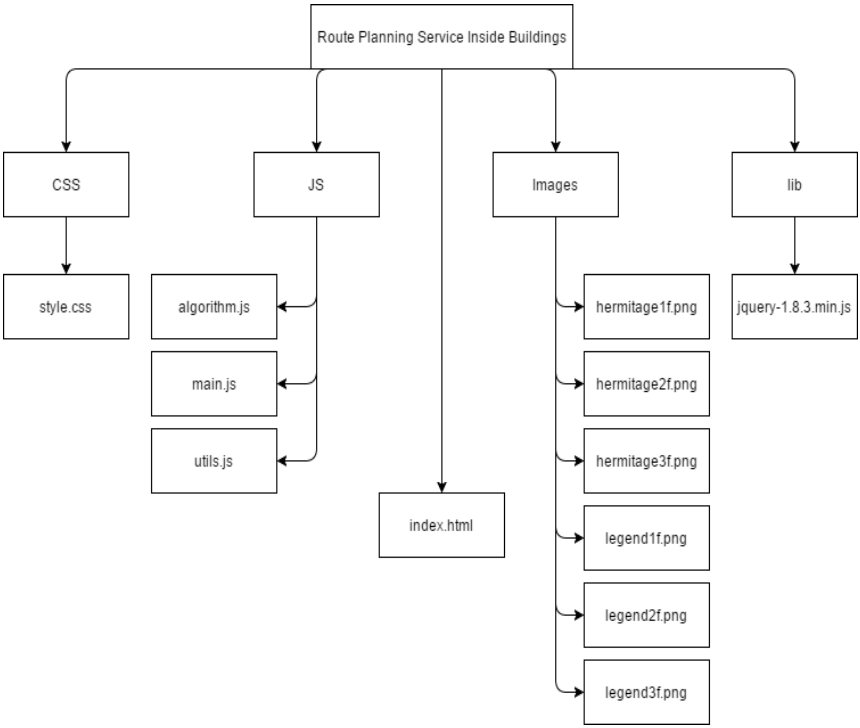


Рис. 3.2. Файловая система проекта

Как видно из этой диаграммы, проект содержит 4 каталога и один главный файл index.html. Файл содержит основную структуру веб-страницы, а также подключает все библиотеки и CSS-таблицы. В каталоге Images хранятся карты помещений и условные обозначения. В CSS каталоге содержатся все стили веб-страницы. Папка lib хранит всего одну библиотеку — jQuery, которая легко обеспечивает доступ ко всем элементам и атрибутам объектно-ориентированной модели (ОМД). В папке JS хранятся все сценарии, алгоритмы и скрипты, обеспечивающие основной функционал сервиса.

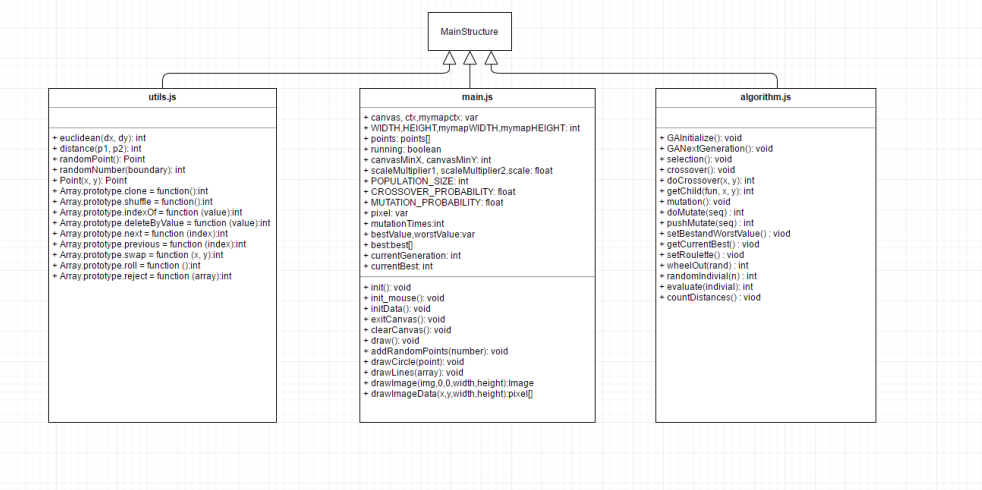


Рис. 3.3. Диаграмма классов проекта

Файл main.js содержит все сценарии, скрипты, события и данные.

Основные Поля:

var canvas — холст, на котором будет отображена карта и точки обхода.

var ctx — контекст рисования на холсте для точек обхода.

var mymapctx — контекст рисования на холсте для карты.

int WIDTH — ширина холста.

int HEIGHT — длина холста.

bool running — состояние работы алгоритма.

int currentBest — переменная, которая хранит какое-то время найденное оптимальное расстояние, пока не найдет еще лучше.

int currentGeneration — номер текущего поколения.

int bestValue — лучшее расстояние, которое было найдено за всю работу алгоритма.

int worstValue — худшее расстояние, которое было найдено за всю работу алгоритма.

best[] — массив всех найденных оптимальных решений для будущего отбора.

int mutationTimes — количество мутаций алгоритма

float scale — масштабируемость карты

int POPULATION_SIZE — количество популяции (маршрутов)

float MUTATION_PROBABILITY — вероятность мутации

float CROSSOVER_PROBABILITY — вероятность селекции

Методы:

init() — загрузка начального холста

init_mouse() — метод, в котором описаны все события и скрипты мыши

initData() — начальная инициализация всех переменных

addRandomPoints(number) — добавляет случайно 10 вершин на карту.

drawCircle(point) — рисует вершины графа.

drawLines(array) — рисует ребра графа.

draw() — метод, который рисует все объекты и карту, а также всю информацию о работе алгоритма.

clearCanvas() — метод, который очищает весь холст от вершин и путей маршрута.

exitCanvas() — закрывает веб-страницу сервиса.

getImageData(x,y, WIDTH,HEIGHT) — метод, который определяет цвет пикселя.

drawImage(img, 0, 0,width, height) — метод, который рисует карту

Файл utils.js содержит метод, который создает точки, основные операции с исходным массивом и метод, вычисляющий расстояние между точками.

Методы:

distance(p1, p2) — вычисляет расстояние между двумя точками, используя метод Евклида.

randomPoint() — создает случайно вершину внутри холста

Point(x, y) — метод создает точки и дает им координаты x,y.

Прототипы для массива:

Array.prototype.clone — прототип клонирует исходный массив.

Array.prototype.shuffle — прототип перемешивает массив.

Array.prototype.indexOf — прототип возвращает первый (наименьший) индекс элемента внутри массива. Если элемент не найден, то возврат значения -1.

Array.prototype.deleteByValue — удаление элемента массива по значению.

Array.prototype.next — следующий элемент массива.

Array.prototype.previous — предыдущий элемент массива.

Array.prototype.swap — обмен значениями двух элементов. массива.

Array.prototype.roll — прототип получает случайное количество вершин и добавляет в исходный массив.

В файле algorithm.js реализован генетический алгоритм с селекцией рулетки.

Основные методы:

getCurrentBest() — метод, который вычисляет наиболее оптимальный маршрут.

getCurrentWorst() — метод, который вычисляет самый невыгодный (случайный) маршрут.

setBestandWorstValue() — метод, который устанавливает самый оптимальный маршрут и самый худший маршрут.

GAInitialize() — начальное поколение, которое создает случайный (неоптимальный) маршрут.

randomIndivial(n) — метод, который путем тасования элементов массива, участвует в создании первого случайного поколения.

GANextGeneration() — создание нового поколения при помощи отбора, скрещивания и мутации.

selection() — метод селекции (отбора).

crossover() — метод скрещивания (размножение).

mutation() — метод мутации.

setRoulette() — метод рулетки.

countDistances() — расчет расстояния между точками при помощи метода евклида.

evaluate(indivial) — оценка всего пути.

Помимо заданных требований в подразделе 1.4, в сервисе есть одно условие, которое было сделано для того, чтобы сервис мог строить оптимальный маршрут не только для заданной карты, но и для других планов зданий.

Для стабильной работы сервиса необходимо загружать изображения плана этажа только на черном фоне. Места, которые имеют черный цвет сервис воспринимает как препятствия, в которые нельзя поставить точку или провести маршрут, например, двери, окна, стены и все, что находится снаружи здания. Благодаря этому условию программа будет строить все маршруты только внутри здания. Сервис рассчитан так и для цветных карт, так и для черно-белых. Данное условие реализовано при помощи метода getImageData(), который проходит все изображение. Если на каком-то месте метод определяет черный цвет, то в данном месте метод запрещает ставить точки.

3.3 Реализация генетического алгоритма

Исходя из таблиц, полученных в подразделе 2.3, для решения задачи коммивояжера был выбран генетический алгоритм, основные шаги которого можно увидеть на следующей схеме (см. рис. 2.7). Начальными параметрами будут являться:

- вершины графа от 2 до 100
- количество популяций 10
- максимальное количество поколений 1000

На первом этапе происходит создание начальной популяции. На этом этапе нам не важно, как создавать начальную популяцию, так как генетический алгоритм все равно будет решать задачу оптимизации независимо от того является ли данная популяция жизнеспособной или нет. Поэтому здесь достаточно написать функцию, которая будет принимать количество вершин графа, случайно их перемешивать и записывать, в качестве результата для первой популяции, оценку всего маршрута. Реализация данной функции представлена на рис. 3.4 ниже.

```
function GAInitialize() {
  countDistances();
  for(var i=0; i<POPULATION_SIZE; i++) {
    population.push(randomIndivial(points.length));
  }
  setBestandWorstValue();
}
```

Рис. 3.4. Функция генерации начального поколения

3.3.1 Операция селекции (отбор)

Следующий этап ГА — операция селекции. Существуют различные методы отбора популяции, тесты которых представлены в подразделе 2.3 (см. рис. 2.8). Для операции селекции генетического алгоритма был реализован метод рулетки, реализация которого изображена на рис. 3.5 и рис. 3.6.

```
function selection() {
  var parents = new Array();
  var initnum = 4;
  parents.push(population[currentBest.bestPosition]);
  parents.push(doMutate(best.clone()));
  parents.push(pushMutate(best.clone()));
  parents.push(best.clone());
  setRoulette();
  for(var i=initnum; i<POPULATION_SIZE; i++) {
    parents.push(population[wheelOut(Math.random())]);
  }
  population = parents;
}
```

Рис. 3.5. Операция селекции с использованием метода рулетки

```
function setRoulette() {
  // расчет все функций приспособленности
  for(var i=0; i<values.length; i++) { fitnessValues[i] = 1.0/values[i]; }
  //устанавливаем рулетку
var sum = 0;
for(var i=0; i<fitnessValues.length; i++) { sum += fitnessValues[i]; }
  for(var i=0; i<roulette.length; i++) { roulette[i] = fitnessValues[i]/sum; }
    for(var i=1; i<roulette.length; i++) { roulette[i] += roulette[i-1]; }
  }
```

Рис. 3.6. Метод рулетки

Селекция отбирает тех двух родителей (x, y), у которых наибольшая функция приспособленности fitnessValues.

3.3.2 Операция скрещивания

После отбора двух родителей наступает следующий этап — скрещивание (размножение). Двое «лучших» родителей участвуют в создании нового потомства (маршрута). Всего они произведут не менее 9. Весь процесс на рис. 3.7.

```
  function crossover() {
    var queue = new Array();
    for(var i=0; i<POPULATION_SIZE; i++) {
      if( Math.random() < CROSSOVER_PROBABILITY ) {
        queue.push(i);
      }
    }
    queue.shuffle();
    for(var i=0, j=queue.length-1; i<j; i+=2) {
      doCrossover(queue[i], queue[i+1]);
  }
}
function doCrossover(x, y) {
  child1 = getChild('next', x, y);
  child2 = getChild('previous', x, y);
  population[x] = child1;
  population[y] = child2;
}
function getChild(fun, x, y) {
  solution = new Array();
  var px = population[x].clone();
  var py = population[y].clone();
  var dx,dy;
  var c = px[randomNumber(px.length)];
  solution.push(c);
  while(px.length > 1) {
    dx = px[fun](px.indexOf(c));
    dy = py[fun](py.indexOf(c));
    px.deleteByValue(c);
    py.deleteByValue(c);
    c = dis[c][dx] < dis[c][dy] ? dx : dy;
    solution.push(c);
  }
  return solution;
}
```

Рис. 3.7. Операция скрещивания

3.3.3 Операция мутации

Мутация вносит изменения для некоторых особей с некоторой вероятностью (как и для скрещивания) для получения новой информации о популяции. Реализация на рис. 3.8.

```
function mutation() {
  for(var i=0; i<POPULATION_SIZE; i++) {
    if(Math.random() < MUTATION_PROBABILITY) {
      if(Math.random() > 0.5) {
        population[i] = pushMutate(population[i]);
      } else {
        population[i] = doMutate(population[i]);
      }
      i--;
    }
  }
}

function doMutate(seq) {
  mutationTimes++;
  // m and n refers to the actual index in the array
  // m range from 0 to length-2, n range from 2...length-m
  do {
    m = randomNumber(seq.length - 2);
    n = randomNumber(seq.length);
  } while (m>=n)

  for(var i=0, j=(n-m+1)>>1; i<j; i++) {
    seq.swap(m+i, n-i);
  }
  return seq;
}
function pushMutate(seq) {
  mutationTimes++;
  var m,n;
  do {
    m = randomNumber(seq.length>>1);
    n = randomNumber(seq.length);
  } while (m>=n)

  var s1 = seq.slice(0,m);
  var s2 = seq.slice(m,n)
  var s3 = seq.slice(n,seq.length);
  return s2.concat(s1).concat(s3).clone();
}
```

Рис. 3.8. Операция мутации

Так как генетический алгоритм моделирует эволюционный процесс, который может продолжаться до бесконечности, то необходим такой критерий, который бы останавливал данный процесс, поэтому последним шагом будет являться критерий останова.

Помимо достижения максимального количества поколений, в программе есть условие, которое останавливает генетический алгоритм еще до достижения значения в 1000 поколений. Если следующие 100 поколений не дают особи лучше тех, которые есть, то алгоритм останавливается и выводится лучшая особь, то есть маршрут.

3.4 Интерфейс пользователя

После запуска веб-приложения, при помощи любого браузера, мы видим весь интерфейс (рис. 3.9), который состоит из нескольких частей. Главное окно занимает большую часть экрана, в котором будет отображаться карта этажа здания (рис. 3.10). Рядом с ним есть меню карты (рис. 3.11), при помощи которого мы можем выбрать здание, этаж, загрузить свою карту, изменить масштаб карты, отобразить план этажа здания на главном окне и отобразить описание карты на панели управления (рис.3.12). Панель управления находится справа от главного окна (рис. 3.13). При помощи неё мы можем добавить точки обхода так и кликом мыши, так и нажатием клавиши «Добавить точки» (рис.3.14), проложить маршрут, показать лучший или худший путь, очистить карту и закрыть приложение.

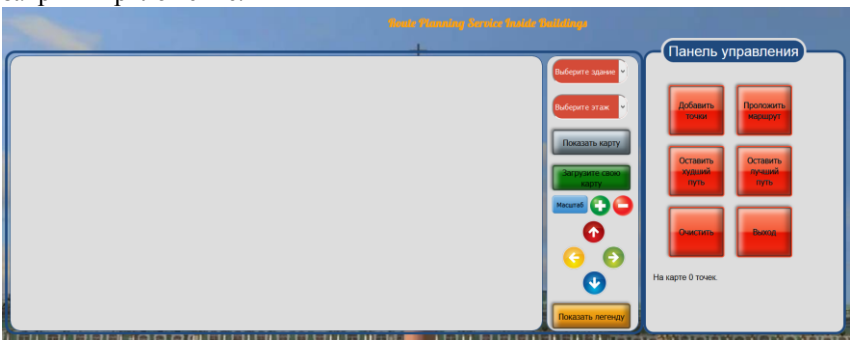
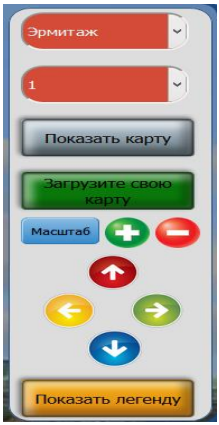
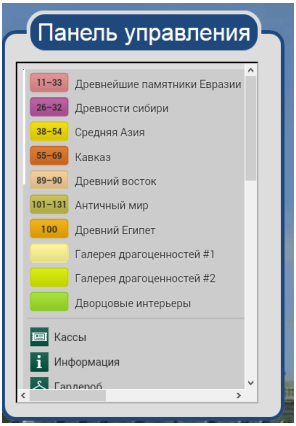


Рис. 3.9. Интерфейс пользователя

Рис. 3.10. Главное окно





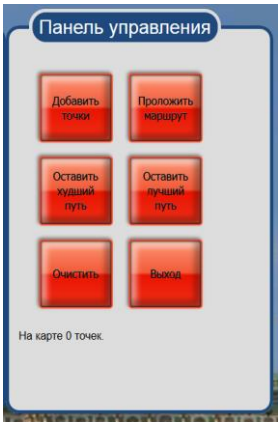


Рис. 3.11. Меню Рис. 3.12. Указатель Рис. 3.13. Панель управления

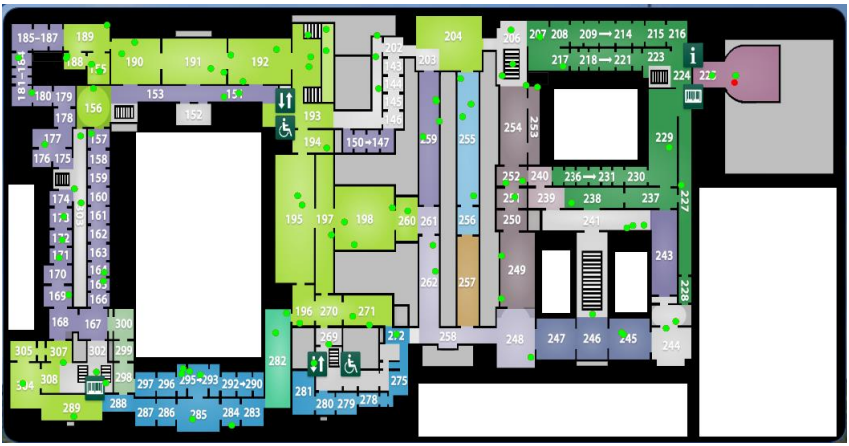


Рис. 3.14. Расстановка точек на карте первого этажа Эрмитажа

Красная точка является началом маршрута, а все остальные зеленые точки являются местами, которые хочет посетить человек. Все эти точки можно поставить на любой участок карты, кроме тех мест, которые по сути являются обычным фоном и не представляют интереса, пример подобных участков показан на рисунке выше, и отмечены они белыми прямоугольниками. (см. рис. 3.14).

Глава 4

Тестирование генетического алгоритма и оценка результатов

4.1 Тестирование интерфейса сервиса

Перед тем как загружать веб-приложение на веб-сервер, необходимо его протестировать и проверить корректность результатов. Вначале стоит посмотреть при каких сценариях приложение не будет работать или будет работать некорректно, а заодно проверить выполнение заданных требований, приведенные в подразделе 3.3. Если попытаться добавить на карту больше, чем 100 точек, то сервис выдаст сообщение о превышении лимита точек (рис 4.1).

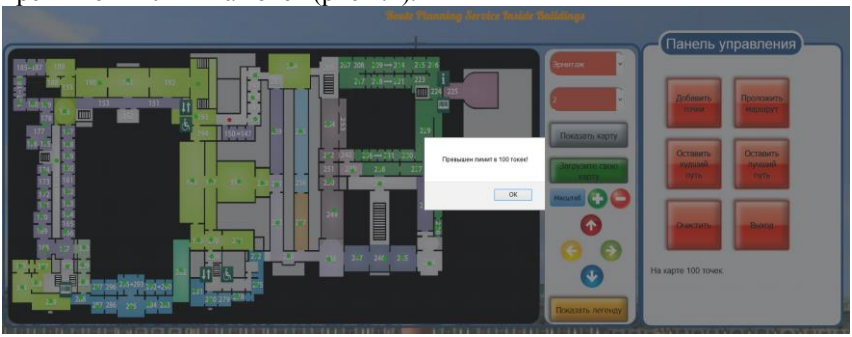


Рис. 4.1. Попытка добавить на карту больше 100 точек

Если попытаться построить маршрут для одной точки, то сервис выдаст сообщение, что для построения маршрута необходимо больше, чем одна точка (рис. 4.2).

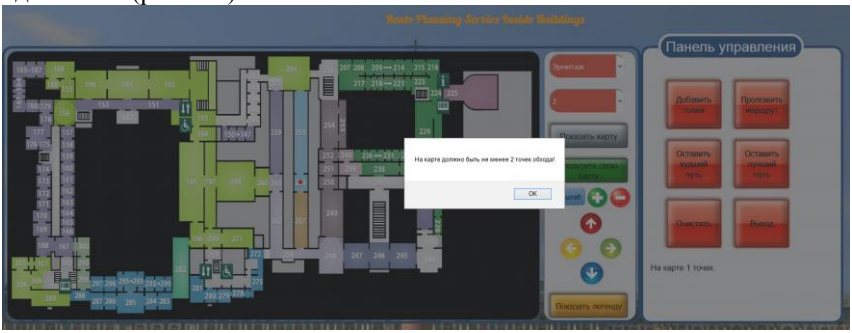


Рис. 4.2. Попытка построить маршрут для одной точки

Как видно по рисункам 4.1 и 4.2, заявленные требования выполняются. Также обработаны сценарии, направленные на улучшение взаимодействия пользователя с веб-приложением. Например, нельзя добавить точки на пустом окне без карты. При попытке отобразить карту, не выбирая этажа и здания, сервис выдаст соответствующее сообщение (рис. 4.3). Можно было бы обойтись без этого сценария, если бы сервис работал только для единственного плана этажа, однако в данном сервисе есть возможность загружать свои карты и менять этажи. Тот же самый сценарий работает и для легенды карты.

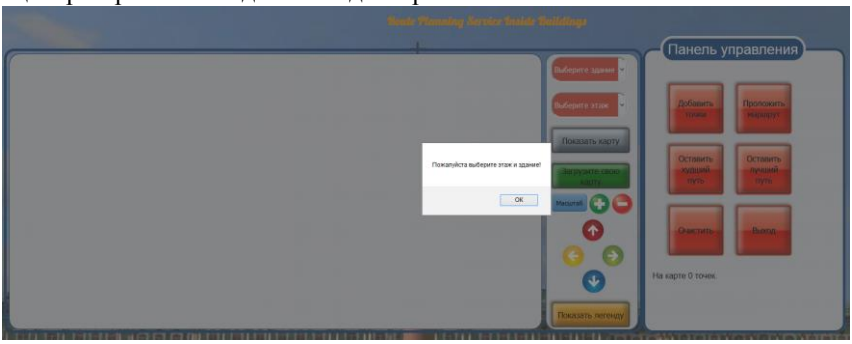


Рис. 4.3. Попытка отобразить карту и её легенду, не выбирая этаж и здание

Последний сценарии сделан для того, чтобы пользователь, который случайно нажал на кнопку «Выход» или на крестик на вкладке веб-страницы, мог всегда отменить свое действие и продолжить работу, не теряя свой маршрут (рис. 4.4).

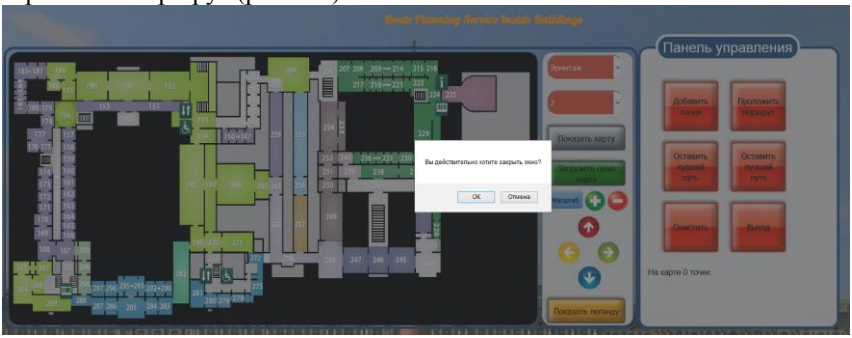
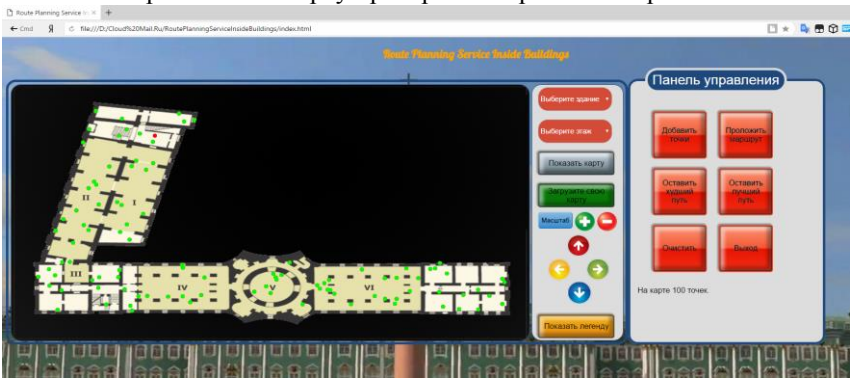


Рис. 4.4. Реакция сервиса при попытке закрыть приложение

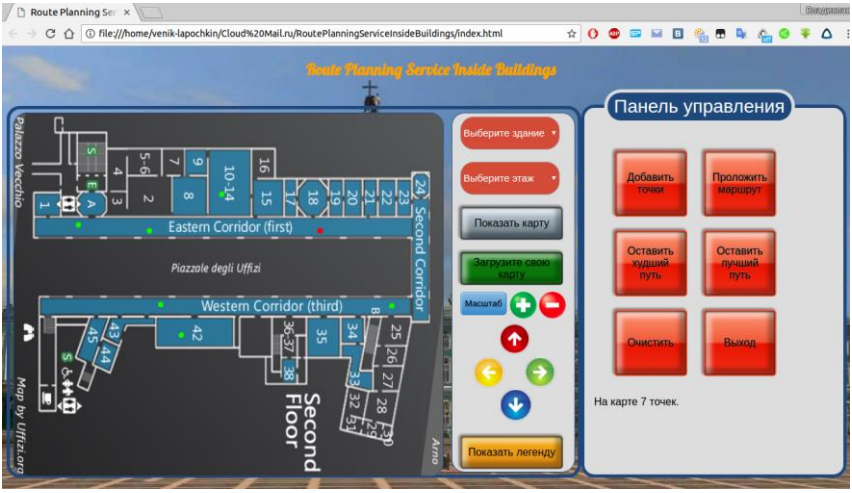
4.2 Тестирование основных функций сервиса

Проверив, как работает интерфейс, переходим к тестированию отображения веб-сервиса на различных браузерах. Так как люди могут использовать разные браузеры, которые обладают своими особенностями (самый простой пример — это то, что у каждого окна браузера есть свои размеры окна, которые еще и меняются в зависимости от разрешения монитора), необходимо убедиться в корректности отображения приложения для разных браузеров. Из существующих браузеров для тестирования было выбрано 4 популярных: Mozilla Firefox, Яндекс браузер, Google Chrome и Opera (рис. 4.5, рис. 4.6, рис. 4.7) [19]. Результат отображения приложения в браузере Mozilla Firefox можно посмотреть в подразделе 3.4 (см. рис. 3.9 и 3.10).

Рис. 4.5. Приложение в браузере Opera. Разрешение экрана 1920x1080



4.6 Приложение в Яндекс браузере. Разрешение экрана 1920x1080



4.7. Приложение в Google Chrome. Разрешение экрана 1366x768

Проверив отображение веб-сервиса на различных браузерах с разным разрешением, переходим к тестированию основного функционала сервиса, а именно построение маршрута внутри здания.

Для начала у нас есть выбор либо воспользоваться теми картами, которые есть в наличии (доступны пока что только 3 этажа Эрмитажа), либо загрузить свою (см. рис. 4.6 и рис. 4.7). Далее мы ставим точки, которые хотим посетить и точку начала маршрута. Начальная точка будет красного цвета, а те точки, которые хотим посетить будут зеленые. Сами точки мы можем добавлять при помощи двойного клика мыши по карте, также мы можем при помощи панели управления рекурсивно добавить несколько случайных точек. Проставив все необходимые точки, нажимаем на кнопку «Проложить маршрут». После того, как алгоритм закончил свою работу, можно посмотреть на главном окне построенные два маршрута: лучший и худший (рис. 4.5), а также всю информацию о них. В полученной информации, которая доступна внизу панели управления, можно узнать количество точек на карте, включая стартовую, время и расстояние худшего и лучшего пути, а также на каком поколении алгоритм закончил свою работу. Результат выполнения работы сервиса приведен в таблице 4.1, а результат работы ГА в таблице 4.2.

Рис. 4.5. Вывод лучшего (белый) и худшего (черный) маршрута для первого этажа Кунсткамеры

Для того чтобы оставить единственный маршрут, например, лучший, то необходимо нажать на соответствующую кнопку. Результат на рис. 4.6.

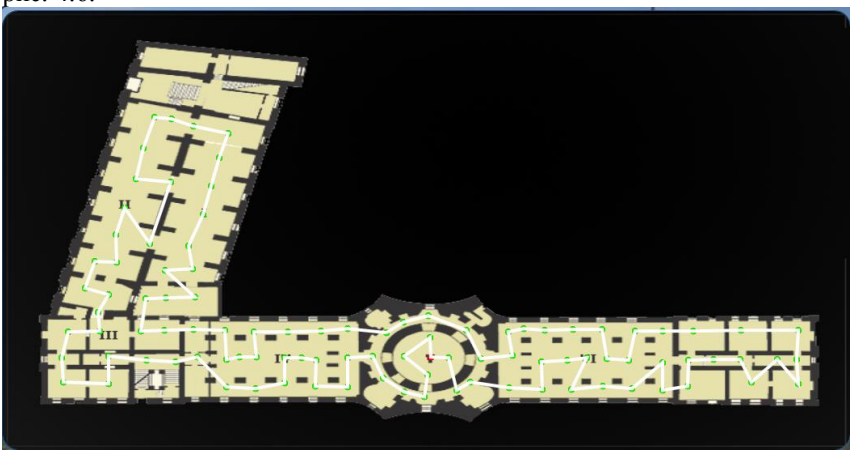


Рис. 4.6. Только лучший путь

Таблица 4.1. Результат обхода 100 точек первого этажа Кунсткамеры

	Оптимальный путь	Худший путь
Время	42 мин.	383 мин.
Весь путь	3526 м.	31952 м.
Количество вершин	100	
Время работы программы	7.6 сек.	

Таблица 4.2. Результат работы ГА для 100 вершин

Поколение	Лучший путь	Кандидаты на селекцию	Лучшие кандидаты для скрещивания
1	31952	31952,35807,35510,33035,36663, 34895,33230,32765,33144,32589	x: 31952 y: 32589
2	22687	22687,25624,31406,32340,24053, 33144,25448,26035,25282,33144	x: 22687 y: 24053
3	18605	18605,22594,19152,22523,25282, 18647,22233,21328,19999,25282	x: 18605 y: 18647
4	14882	17997,18236,18605,16675,14882, 15522,22233,14951,16855,18605	x: 14882 y: 14951
5	13207	13207,14695,14882,14882,17997, 13512,15319,14800,14951,18236	x: 13207 y: 13512
6	11892	13282,14085,11960,11892,12987, 15319,12748,12376,12594,14882	x: 11892 y: 11960
7	10707	11261,12293,11196,11244,10707, 12179,12747,11494,11277,11342	x: 10707 y: 11196
8	9236	9441,10433,10417,10304,9790,10 234,9511,9236,10864,11508	x: 9236 y: 9511
9	8944	9269,9585,9175,10040,10743,962 8,8944,9729,10234,9790	x: 8944 y: 9175
10	8801	9131,9767,8801,9533,10234,9403, 10814,9614,9947,8944	x: 8801 y: 8944
221	3527	3527,4381,3582,3527,3527,3527,3 576,3567,3527,3527	x: 3527 y: 3567
222	3527	3527,3546,3707,3527,3527,3527,3 527,3527,3527,3527	x: 3529 y: 3556
223	3529	3581,4858,4987,3581,3527,3527,3 546,3526,3611,3527	x: 3526 y: 3527
323	3529	3526,3617,4634,3526,4610,3526,3 572,4834,3526,3526	x: 3529 y: 3536

Как видно в таблице 4.2 после 223 поколения, следующие 100, не улучшают качественную характеристику маршрута и поэтому можно считать, что оптимальное решение найдено, и алгоритм может быть остановлен еще до достижения максимального значения поколений. Данные результаты, полученные после работы сервиса, были отображены в браузере Google Chrome. Приложение было также протестировано и для других зданий (рис. 4.7)

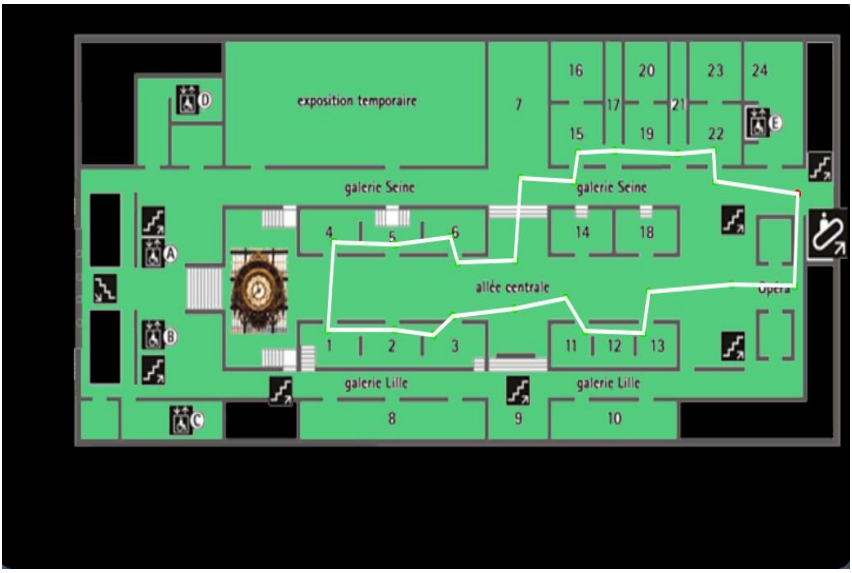


Рис. 4.7. Второй этаж музея д Орсе.

Заключение

В результате выполнения выпускной работы были решены следующие задачи:

- Проведено исследование алгоритмов решения задачи коммивояжера. На основании этих исследований были составлены таблицы, по которым был сделан вывод о том, что алгоритм Прима-Эйлера и метод отжига можно применять в решении задачи коммивояжера, только если количество точек обхода не будет превышать 50 вершин. Если количество вершин не превышает 25, то можно использовать МВиГ. ГА и МБС за короткое время находят оптимальный маршрут так и для малого количества вершин, так и для максимального количества вершин, равное значению 100, однако, сравнивая ГА и МБС, можно увидеть, что ГА работает в ≈1.5 раза быстрее, чем МБС.
- Был выбран генетический алгоритм с селекцией рулетки. Генетический алгоритм за короткое время находит оптимальный маршрут независимо от количества вершин (от двух до 100). Данный результат можно объяснить тем, что в алгоритме эффективно реализован критерий останова, который отслеживает момент, когда качественная характеристика маршрута не меняется на протяжении 100 поколений, что позволяет алгоритму остановиться, не доходя до максимального значения, которое равно 1000 поколений, вывести значения минимального маршрута и построить замкнутую кривую на карте.
- Был разработан веб-сервис, который позволяет строить маршруты внутри зданий, основанный на генетическом алгоритме.
- Разработанный сервис был протестирован для ряда зданий.
- Результаты тестирования показали, что сервис соответствует сформулированным требованиям и ограничениям.

Список литературы

1. На Google Maps появились схемы российских зданий [Электронный ресурс]. — Электронные новости. — Режим доступа: https://lenta.ru/news/2013/09/18/plans/
2. Тест навигации внутри помещений [Электронный ресурс]. — Электронные новости. — Режим доступа: http://ichip.ru/test-navigacii-vnutri-pomeschenii.html
3. Официальный блог Google Россия. Схемы зданий на Google Картах приходят в Россию [Электронный ресурс]. — Электронный блог. — Режим доступа: http://ichip.ru/test-navigacii-vnutri-pomeschenii.html
4. GateGuru A Trip Advisor Company. — [Электронный ресурс]. — Режим доступа: http://gateguru.com/about.html
5. Системы позиционирования внутри зданий для мобильных сервисов [Электронный ресурс]. — Статья. — Режим доступа: https://habrahabr.ru/post/126410/
6. Гмарь Д.В., Дюльдина К.И. Навигация внутри зданий с использованием беспроводной сети (на примере кампуса ВГУЭС) [Электронный ресурс]: публикация в сборнике материалов конференций, форумов, симпозиумов, семинаров. — Электронные данные. — Режим доступа: http://ecampus.vvsu.ru/publication/science_publications/details/material/14847/navigaciya_vnutri_zdaniy_s
7. Навигация по зданию университета ГУАП [Электронный ресурс]. — Электронные данные. — Режим доступа: http://purecreation.ru/suainav/about.html
8. Борознов В. О. Исследование решения задачи коммивояжера [Электронный ресурс] // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2009. - № 2. – Режим доступа: http://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera
9. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. — Нижний Новгород, 2007, 85 с.
10. Левитин А.В. Алгоритмы: введение в разработку и анализ. М.: Вильямс, 2006. 576 с.

11. Костюк Ю. Л. Эффективная реализация алгоритма решения задачи коммивояжера методом ветвей и границ. Прикладная дискретная математика. Вычислительные методы в дискретной математике, 2010. 78-90 с.
12. Окулов С.М. Программирование в алгоритмах. — М.:Бином. Лаборатория знаний, 2002. 341 с.
13. Асанов М.О., Баранский В.А., Расин В.В. Дискретная математика: графы, матроиды, алгоритмы. Ижевск: НИЦ «РХД», 2001. 288 с.
14. Дискретная математика. Методы решения трудоемких задач. [Электронный ресурс]. — Материалы лекций. — http://rain.ifmo.ru/cat/view.php/theory/unsorted/approx-2004
15. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы: Учебное пособие. — 2-е изд. — М:Физматлит, 2006. 320 с.
16. Крук Е.А., Овчинников А.А. Методы программирования и прикладные алгоритмы: Учебное пособие. ГУАП. — СПб., 2007. 166 с.
17. Вишняков П.О. Планирование маршрутов с использованием модифицированного метода «ближайшего соседа» // Математические методы в технике и технологиях. ММТТ,2014. 63-76 с.
18. Елизаров С. Введение в оптимизацию. Имитация отжига [Электронный ресурс]. — Статья. — Режим доступа: https://habrahabr.ru/post/209610/
19. Софт каталог. Выбираем лучший браузер для Windows [Электронный ресурс]. — Электронные данные. — Режим доступа: http://softcatalog.info/ru/obzor/vybiraem-luchshiy-brauzer