УДК: 004.94; 004.02

**Алёшкин А.С., Лесько С.А., Титов В.В.**

Московский технологический университет (МИРЭА), г. Москва, Россия

# МОДЕЛИ И АЛГОРИТМЫ ОПТИМИЗАЦИИ МАРШРУТОВ В ТРАНСПОРТНОЙ СЕТИ ГОРОДА*

**Аннотация**

*В представленной работе описаны модели и алгоритмы поиска и оптимизации маршрутов в транспортной сети города, включая вопросы увеличения их производительности. В статье рассматриваются реальные, привязанные к картографическим данным характеристики и атрибуты маршрутов движения транспорта и пешеходов, что позволяет осуществить практическую реализацию поиска оптимальных маршрутов. На основе разработанных моделей и алгоритмов авторами было создано мобильное приложение под ОС Android, позволяющее реализовывать различные сценарии пользовательского поведения. Авторы работы описывают архитектуру разработанного приложения (базу данных, модули, программные интерфейсы и т.д.), его функциональные возможности и модули.*

*Кроме того, в работе представлены результаты реального тестирования моделей, алгоритмов и созданного на их основе программного обеспечения.*

**Ключевые слова**

*Транспортная сеть, поиск маршрута, оптимизация маршрута, алгоритмы поиска и оптимизации маршрута, атрибуты описания маршрутов, мобильное приложение, программное обеспечение.*

**Lesko S.A., Alyoshkin A.S., Titov V.V.**

Moscow Technological University (MIREA), Moscow, Russia

# MODELS AND ALGORITHMS OF OPTIMIZATION OF ROUTES IN THE TRANSPORT NETWORK OF THE CITY

**Abstract**

*The presented work describes models and algorithms for searching and optimizing routes in the city transport network, including questions of increasing their productivity. The article considers real characteristics and attributes of the traffic routes and pedestrians that are attached to cartographic data, which makes it possible to realize the practical search for optimal routes. Based on the developed models and algorithms the authors created a mobile application for Android OS, which allows implementumn various scenarios of user behavior. The authors describe the architecture of the developed application (database, modules, program interfaces, etc.), its functionality and modules.*

*In addition, the paper presents the results of real testing of models, algorithms and the software created on their basis.*

**Keywords**

*Transport network, route search, route optimization, route search and optimization algorithms, route description attributes, mobile application, software.*

## Введение

На сегодняшний день существует множество приложений, позволяющих автоматически прокладывать оптимальные маршруты с минимальным участием со стороны пользователей. В основе

* **Труды II Международной научной конференции «Конвергентные когнитивно-информационные технологии» (Convergent'2017), Москва, 24-26 ноября, 2017**

**Proceedings of the II International scientific conference "Convergent cognitive information technologies" (Convergent'2017), Moscow, Russia, November 24-26, 2017**

таких приложений лежат алгоритмы, которые осуществляют поиск оптимального пути между двумя пунктами. С целью обеспечения работы таких алгоритмов необходимо всю карту разделить на наборы вершин и взвешенных дуг, иными словами, преобразовать карту во взвешенный ориентированный граф. Существующие приложения Google.Maps, Яндекс.Карты, Яндекс.Метро, Rusavtobus и другие позволяют прокладывать кратчайший или оптимальный путь от одного пункта в другой, но имеют существенный недостаток. Основная проблема, с которой приходится сталкиваться в подобных программных средствах — это отсутствие возможности внесения пользователем индивидуальных данных, с которыми работает алгоритм карты. Пользовательские данные могут состоять из новых остановок (места работы, учебы, магазины, общественные места и т.д.) и путей до них. Небольшое число существующих программных средств позволяет вносить пользователям изменения, но, зачастую, это представляет собой нетривиальный процесс, требующий наличия у пользователя сторонних приложений. Именно по этой причине, на рынке могут быть востребованы программные средства, которые бы позволяли изменять карты в самих программных средствах, имея при этом доступ только к минимальному объёму информации о структуре данных, согласно которой организовано хранение данных о маршрутах; или же сторонние приложения, которые бы позволяли изменять маршруты в конечном визуальном представлении.

Проблема поиска оптимальных путей может быть решена с помощью разных алгоритмов, актуальность которых зависит от области применения и типа графа [1,2]. Для этого могут быть использованы модифицированные алгоритмы Дейкстры [3], алгоритмы поиска в ширину (Breath-First Search, BFS) [4], а также разновидности алгоритма поиска в глубину (Depth-First search, DFS) [5]. Все вышеперечисленные алгоритмы можно отнести к разряду алгоритмов поиска кратчайшего пути межу парой вершин (Single source shortest path, SSSP).

Модифицированный алгоритм Дейкстры предназначен не только для поиска кратчайшего пути между парой вершин, но и для поиска набора оптимальных путей между ними. Поиск оптимальных путей достигается посредством итерационного поиска кратчайшего пути между парой вершин при последовательном удалении вершин или дуг, через которые прошел впервые построенный кратчайший путь. Данный алгоритм обладает высокой скоростью работы, но не позволяет определить все возможные оптимальные пути в графе. В связи с этим модифицированный алгоритм Дейкстры не даст точных результатов, а только кратчайший путь и множество возможно оптимальных путей. По этой причине модифицированный алгоритм Дейкстры выполняет не всю поставленную задачу по поиску оптимальных путей в графе, а лишь её часть, и таким образом в прямом виде не может быть использован для работы на мобильных устройствах.

Модифицированный алгоритм поиска в глубину (BFS) позволяет найти все оптимальные маршруты между двумя вершинами, но требует ограничений, которые не позволят алгоритму уйти в бесконечный цикл. Для того что бы модифицированный BFS искал оптимальные маршруты, нужно перестать учитывать уже посещенные вершины и опираться на другие ограничения при поиске оптимальных путей, например, на глубину поиска, достижение каких-либо лимитов и т.д. В связи с тем, что оригинальный алгоритм BFS, так же, как и его модификация, потребляют большое количество памяти, поиск оптимальных путей может потребовать в большом графе может потребовать слишком много ресурсов, что может привести к неправильной или долгой работе алгоритма, что также делает его малопригодным для работы непосредственно на мобильных устройствах.

Разновидности алгоритма поиска в глубину, а именно алгоритмы поиска с ограничением в глубину (Depth Limited Search, DLS) [5] и поиска в глубину с итеративным углублением (Iterative Deepening Depth-First Search, IDDFS) [5], позволяют найти кратчайший путь между парой вершин опираясь на глубину поиска в графе. Алгоритм DLS и IDDFS можно эффективно использовать при работе с большими графами, так как они могут быть ограничены по глубине поиска в нём. В алгоритме DLS ограничение задаётся при запуске алгоритма и алгоритм отрабатывает единожды, а алгоритм IDDFS ищет кратчайший путь постепенно увеличивая границу глубины, что, в некоторых случаях, позволяет найти кратчайший путь быстрее чем с помощью DLS. Для выполнения задачи поиска оптимальных путей в данные алгоритмы, так же, как и в случае с модифицированным алгоритмом BFS, нужно внести изменения. Например, убрать ограничения на посещение уже посещенных вершин, но запретить посещать одну и ту же вершину на одном пути более двух раз. Так же алгоритм должен продолжать свою работу, даже если был найден кратчайший путь. Алгоритмы DLS и IDDFS, в отличии от BFS, требует гораздо меньше памяти устройства и работает быстрее на больших и сложных графах.

Для решения задачи создания мобильного приложения, позволяющего строить оптимальные маршруты, наиболее подходит алгоритм DLS, так как пользователь должен получить все оптимальные маршруты в графе, опираясь на заданную им глубину поиска. Однако и он требует существенных изменений для использования в работе мобильных приложений поиска не оптимального маршрута, а имеющего максимальное удобство для пользователя. Человеку свойственно выбирать маршруты исходя

не из критериев оптимальности, а из удобства. Перспективным направлением развития транспортной сети является создание удобных инструментов, помогающих пользователям строить не только оптимальные, но и удобные маршруты, а для этого необходимо разработать новые, более эффективные модели и алгоритмы оптимизации маршрутов в транспортной сети.

**Описание алгоритма поиска оптимальных маршрутов движения общественного транспорта**

В результате исследования проблемы поиска оптимальных маршрутов был разработан алгоритм, который основывается на базе алгоритма поиска в глубину (Depth Limited Search, DLS) [5]. Результатами работы алгоритма является набор оптимальных путей $R = \{\{L_1, C_1\}, \dots, \{L_k, C_k\}\}$, содержащих путь $L$ и критерии его оптимальности $C$.

Граф транспортной сети можно представить в виде графа: $G = (V, E, O)$, где $G$ – граф транспортной сети, $V = \{i_1, \dots, i_n\}$ – множество вершин $i$, $E$ – множество дуг $(i,j)$ с длиной $c_{ij} \geq 0$, $O$ – множество не оптимизированных вершин $\{i_1, \dots, i_m\}$.

Для нахождения оптимального пути нужно отсортировать полученный в результате работы алгоритма набор оптимальных путей по нужному критерию.

В качестве критериев оптимальности маршрута представлены три величины:

- Время, затрачиваемое на маршрут, $T$;
- Стоимость маршрута, $S$;
- Количество пересадок на маршруте, $N$.

Один из минусов данного алгоритма является его длительное время выполнения, так как происходит поиск абсолютно всех возможных путей из точки отправления в точку прибытия. По этой причине граф нужно оптимизировать, а алгоритм ограничить максимально возможным количеством пересадок. Так же нужно ввести условие, предотвращающее петли (циклы).

Для работы алгоритма требуется:

- Вершины отправления и прибытия $(i,j)$, а так же максимальное количество пересадок, $N_{max}$.
- Вершины $i$ и $j$ помечены как неоптимизируемые, $O = \{i, j\}$.
- Набор оптимальных путей, $R$.
- Оптимизированный граф $G$.
- Начальные критерии оптимальности $C = \{0,0,0\}$.
- Последняя посещенная вершина, $d = i$.

Работа алгоритма состоит из следующих шагов:

1. Если $d = j$, сохранить результаты $\{L,C\}$ в $R$ и завершить итерацию или алгоритм поиска.
2. Проверить каждую дугу, соединяющую вершины $d$ и $b$, если таковых нет, перейти на шаг 10.
3. Если вершина $b$ уже была посещена на этом маршруте $L$, перейти к шагу 2.
4. Если количество пересадок максимально, $N = N_{max}$, а вершина $b$ принадлежит отличному от типа или маршрута точки прибытия $j$, перейти к шагу 2.
5. Если вершина имеет связи с уже посещенными вершинами, перейти к шагу 2.
6. Если вершина $b$ имеет отрицательную задержку до следующего транспорта, перейти к шагу 2.
7. Дублировать путь $L$ и критерии оптимальности $C$, добавить в путь $L_2$ новую вершину $b$.
8. Обновить значения критериев оптимальности $C_2$ в соответствии с новым путём $(d,b)$ и вершиной $b$.
9. Запустить новую итерацию с $d = b$.
10. Завершить итерацию или алгоритм поиска.

После завершения алгоритма, его результаты сортируются в соответствии с нужным критерием и передаются для отображения пользователю (см. рис. 1).

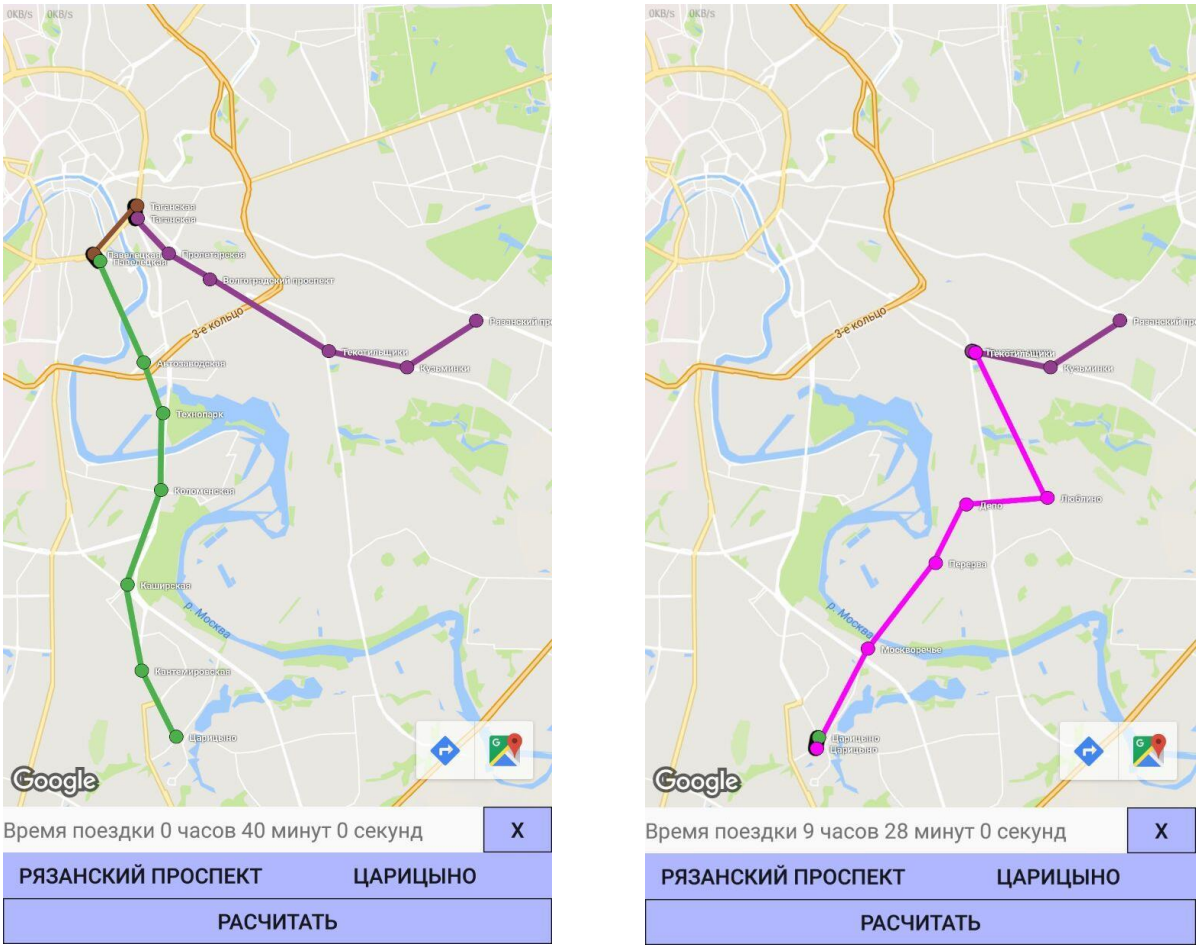


*Рисунок 1 – Результат работы алгоритма*

На рисунках 2 и 3 показаны схема последовательности выполнения разработанного алгоритма.


*Рисунок 2 – Схема подготовки алгоритма поиска в глубину*

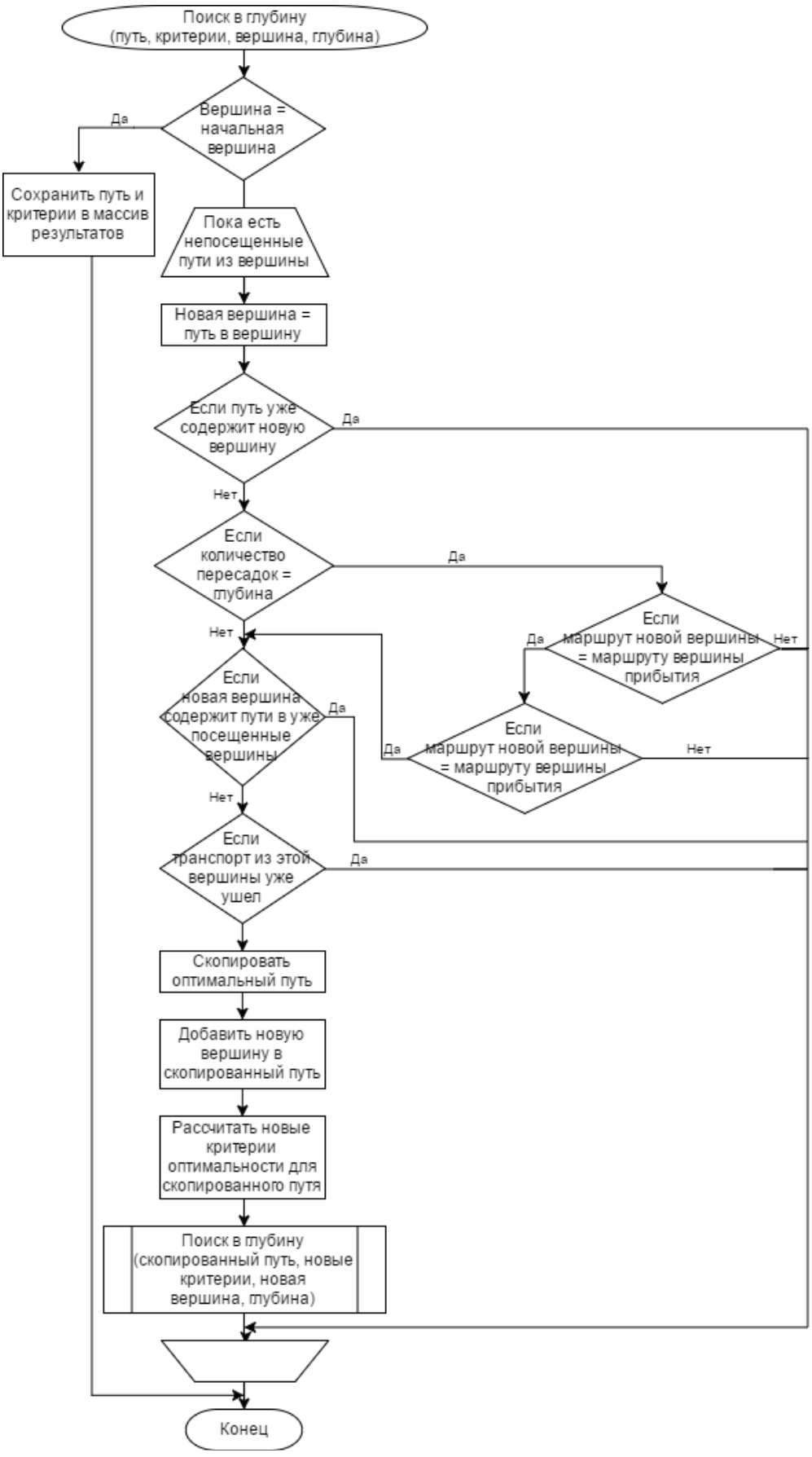


*Рисунок 3 – Блок-схема алгоритма поиска в глубину*

Так как алгоритм обхода в глубину работает медленно, то для увеличения производительности

приложения нужно прибегнуть к оптимизации графа, а именно сокращению количества вершин и путей, которые подходят под критерии оптимизации.

Оптимизация графа начинается с создания копии всего графа. Далее в цикле выполняется процедура оптимизации с условием, что оптимизация возможна.

Критериями оптимизации, под которые попадают вершины, являются:

- существуют две вершины *i* и *j*, для которых выполняется условие, $(i, j) \in E$;
- каждая из вершин *{i,j}* должна иметь не более двух дуг;
- вершины *{i,j}* принадлежат одному маршруту;
- все связанные с *{i,j}* вершины должны принадлежать одному маршруту.

При выполнении данных критериев вершины и соединяющие их дуги могут быть объединены в одну вершину, которая будет сочетать все показатели оптимальности объединенных вершин, такие как время, затраченное на их преодоление и их стоимость.

Если для работы алгоритма требуется вершина, которая находится в объединенной вершине, объединенная вершина разбивается на вершины и дуги, которые были включены в неё при оптимизации. После разбиения, нужная для работы алгоритм вершина помечается как не оптимизируемая и проводится новый цикл оптимизации.

Процедура оптимизации состоит из нахождения пары вершин, которые соответствуют критериям оптимизации. Если пара вершин, удовлетворяющих критериям, не была найдена, процедура оптимизации считается завершенной.

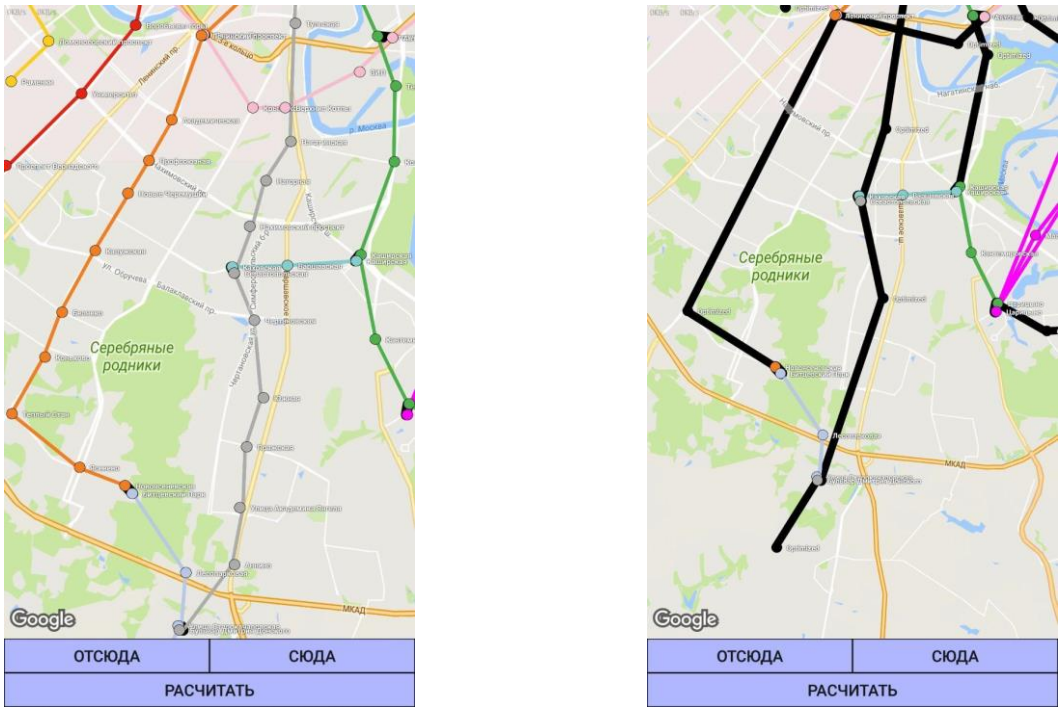


*Рисунок 4 – Схема маршрутов до (левая часть рисунка) и после (правая часть рисунка) оптимизации графа*

Если пара вершин, удовлетворяющих критериям, была найдена, осуществляется процедура изменения графа:

- Новая вершина $i_{n+1}$, которая обладает суммой критериев оптимальности вершин $\{i_1,i_2\}$, а так же дуги $(i_1,i_2)$.
- Вершина $i_{n+1}$ добавляется в граф *G*, а вершины *i* и *j* удаляются из него, $i_{n+1} \in V$, $\{i_1, i_2\} \notin V$.
- Дуги, соединяющие смежные вершины с вершинами $\{i_1,i_2\}$, меняются на дуги, соединяющие смежные вершины с вершиной $i_{n+1}$.

Если алгоритму требуется вершина, которая была сокращена при оптимизации, выполняется процедура её восстановления:

- В графе *G* производится поиск оптимизированной вершины *i,* в которую при оптимизации попала искомая вершина *j*.
- В граф *G* добавляются все вершины и дуги, включенные в оптимизированную вершину *i*.
- Все дуги, соединяющие вершину *i* и смежные с ней вершины, меняются на те, что существовали в графе *G* до оптимизации.

- Из графа $G$ исключается оптимизированная вершина $i$ и её дуги.
- Вершина $j$ помечается как неоптимизированная, $j \in O$.
- Выполняется процедура оптимизации графа.

На рисунке 4 представлен результат оптимизации графа транспортной сети, где слева – граф до оптимизации, справа – граф после оптимизации. Вершины, что попадают под критерии оптимизации, были соединены в одну, а их критерии оптимальности были объединены.

**Программная реализация моделей и алгоритмов оптимизации маршрутов движения в транспортной сети**

Для мобильной операционной системы (ОС) Android была создана программная реализация приложения, осуществляющего оптимизацию маршрутов движения в транспортной сети, обладающая необходимой компактностью и удобством использования.

*Сценарии использования*

При проектировании программного средства были выделены два основных сценария использования программы: поиск оптимальных маршрутов и изменение путей и остановок.

*Поиск оптимальных маршрутов.* Пользователь должен выбрать остановки отправления и прибытия и запустить поиск оптимальных путей (см. рис. 5).

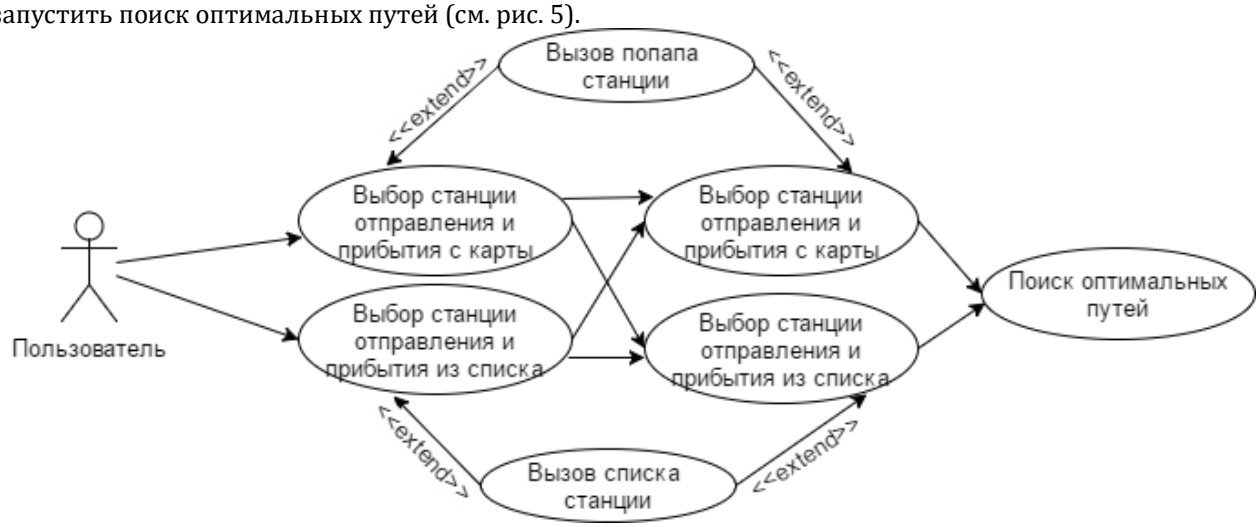


*Рисунок 5 – Поиск оптимальных маршрутов*


*Рисунок 6 – Изменение станции и её путей*

Изменение остановки или пути осуществляется пользователем через меню выбранной остановки. Если пользователь желает изменить параметры, он меняет их в соответствующих полях формы и выходит из экрана изменения станции, если пользователь хочет удалить остановку, он должен выбрать поле «Удалить» и подтвердить свой выбор.

Для изменения или удаления пути пользователь должен выбрать нужную станцию, с которой соединён этот путь, перейти на экран изменения остановки и выбрать её из списка смежных с этой остановкой путей. Если пользователь желает изменить параметры, он меняет их в соответствующих полях формы и выходит из экрана изменения пути, если пользователь хочет удалить путь, он должен выбрать поле «Удалить» и подтвердить свой выбор.

Сценарий использования операций создания, изменения и удаления остановки или пути приведён на рисунке 6.

*Модули программного комплекса*

Программный комплекс состоит из трёх модулей и базы данных (см. рис. 7).

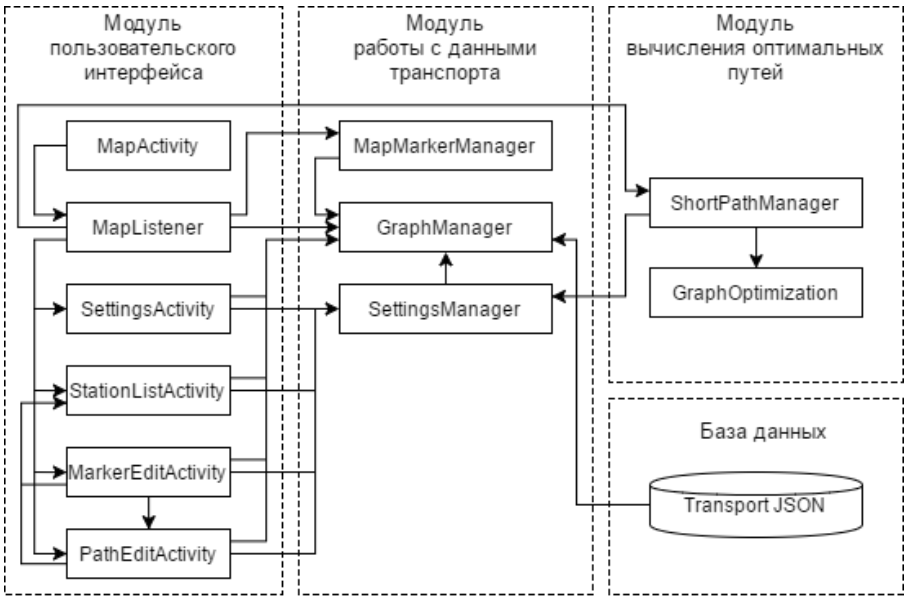


*Рисунок 7 – Модули программного комплекса и БД*

Модуль пользовательского интерфейса выполняет функции по отображению интерфейса пользователю, а также обеспечивает интерактивность интерфейса и вызов методов с других модулей.

- MapActivity – работа с Google Maps API и отслеживание пользовательских событий передаваемых в MapListener.
- MapListener – обработка пользовательских событий Google Maps API.
- SettingsActivity – настройка приложения.
- StationListActivity – список остановок.
- MarkerEditActivity – операции с остановками.
- PathEditActivity – операции с путями.

Модуль работы с данными транспорта обеспечивает сериализацию и десериализацию пользовательских данных об общественном транспорте и их изменение, обновление информации содержащейся на карте, хранение и управление настройками программного комплекса.

- MapMarkerManager – нанесение и изменение объектов общественного транспорта из пользовательских данных.
- GraphManager – сериализацию и десериализацию пользовательских данных об общественном транспорте и их изменение.
- SettingsManager – хранение и предоставление настроек приложение.

Модуль вычисления оптимальных маршрутов предоставляет поиск в графе транспортной сети и уменьшает граф общественного транспорта за счёт складывания нескольких вершин в одну:

- ShortPathManager – поиск и предоставление оптимальных маршрутов графа транспортной сети.
- GraphOptimization – оптимизация графа транспортной сети.

База данных хранит информацию о транспортной сети в формате JSON файла. База данных обрабатывается посредством сериализации или десериализации пользовательских данных в модуле работы с данными транспорта.

*Работа программного комплекса*

Инициализация программного комплекса заключается в запуске Activity карты Google Maps, десериализации объекта пользовательских данных о графе транспортной сети, расстановке маркеров остановок общественного транспорта и пеших маршрутов и централизации карты на нужных координатах (см. рис. 8).

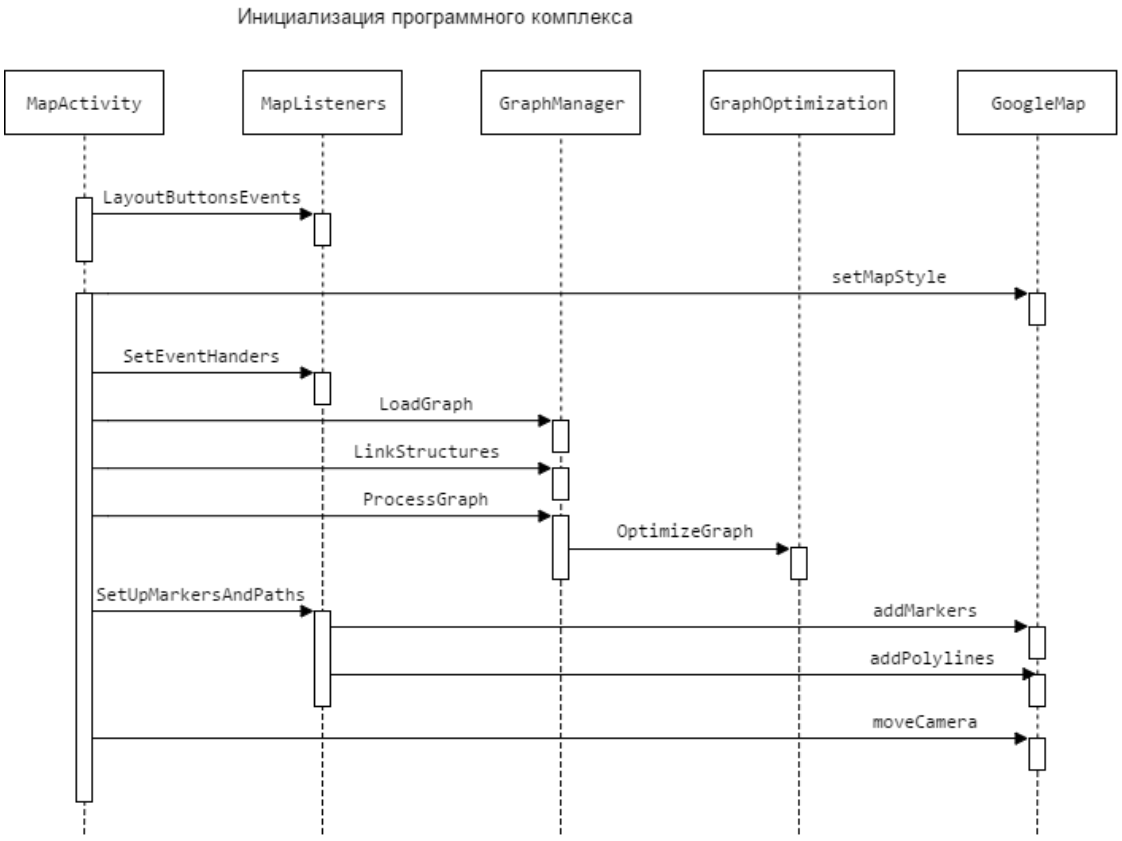


*Рисунок 8 – Инициализация программного комплекса*

Процесс поиска оптимальных маршрутов задействует 4 подмодуля. Подмодуль MapListener предоставляет графический интерфейс для выбора остановок прибытия и отбытия, а также запуска поиска оптимальных маршрутов и отображения его результатов работы. ShortPathManager производит поиск оптимальных маршрутов из графа транспортной сети в GraphManager (см. рис. 9).

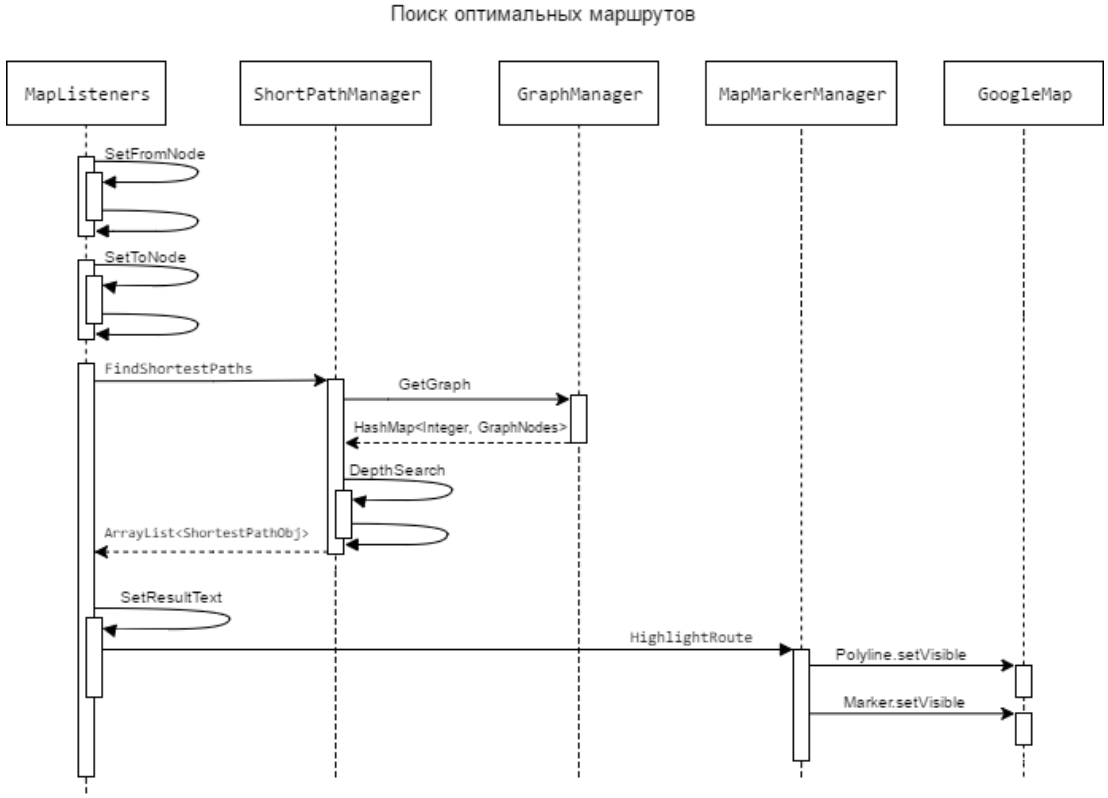


*Рисунок 9 – Поиск оптимальных маршрутов*

Процедура поиска оптимальных маршрутов заключается в переборе всех возможных маршрутов из вершины отправления в вершину прибытия и подсчёта критериев оптимальности каждого найденного маршрута. Для нахождения оптимального маршрута нужно отсортировать результаты алгоритма по критериям оптимальности маршрута в том порядке, в котором нужно пользователю.

Так же для работы алгоритма требуется ограничить возможные маршруты следующими ограничениями:

- Повторное посещение уже посещенных на текущем маршруте вершин невозможно.
- Если текущая исследуемая вершина связана с уже посещенными на этом маршруте вершинами, кроме предыдущей, путь считается недействительным.
- Если количество пересадок превысило максимум, путь считается недействительным.
- Если количество пересадок максимально, а текущая вершина находится на маршруте отличном от маршрута вершины прибытия или принадлежит другому виду транспорта, путь считается недействительным.

Запуск алгоритма поиска оптимальных путей:

- Создать массив для хранения результатов работы алгоритма.
- Восстановить вершины отправления и прибытия, которые могли быть оптимизированы.
- Создать массив хранения текущего пути и добавить в него вершину отправления.
- Создать массив хранения критериев оптимальности текущего маршрута.
- Запустить алгоритм.
- Отсортировать результаты работы алгоритма по времени маршрута.
- Вернуть результаты работы алгоритма.

```
public ArrayList<ShortestPathObj> FindShortestPaths() {
  _algorithmResult = new ArrayList<>();

  RecoverOptimizedNodes(algorithmReadyGraph, fromNodeId, toNodeId);

  GraphNode _fromNode = GraphManager.GetInstance().Nodes.get(Settings.FromStationId);
  _toNode = GraphManager.GetInstance().Nodes.get(Settings.ToStationId);
  ArrayList<Integer> path = new ArrayList<>();

  path.add(_fromNode.Id);
  int[] weight = new int[] {0, 0, 0};

  DepthSearch(path, weight, _fromNode, null, true);
  Collections.sort(_algorithmResult);
  return _algorithmResult;
}
```

Алгоритм поиска оптимальных путей:

1. Если текущая вершина – это вершина прибытия, сохранить текущий маршрут и его критерии оптимальности и перейти к шагу 10.
2. Для каждого пути из текущей исследуемой вершины. Если путей больше нет, перейти к шагу 10.
3. Если путь уже содержит вершину из нового пути или количество пересадок больше максимума, перейти на шаг 2.
4. Если количество пересадок максимально, а вершина из нового пути принадлежит типу или маршруту отличному от вершины прибытия, перейти на шаг 2.
5. Если вершина из нового пути содержит дуги в уже посещенные вершины, перейти на шаг 2.
6. Если сегодняшний поезд уже ушел, перейти на шаг 2, иначе на шаг 7.
7. Если вершина из нового пути является оптимизированной, скопировать маршрут и добавить в него все вершины что были включены в неё при оптимизации, иначе добавить только вершину из нового пути.
8. Скопировать критерии оптимальности и добавить в них критерии оптимальности нового пути и его вершины.
9. Запустить новую итерацию поиска оптимальных маршрутов с новым маршрутом и его критериями оптимальности.
10. Завершить итерацию.

```
private void DepthSearch(ArrayList<Integer> path, int[] weight, GraphNode lastNode,
                         GraphPath lastPath, boolean addDelay) {
  boolean addDelayCopy = false;
```

```
  if (lastNode.Id == _toNode.Id)
  {
    ShortestPathObj result = new ShortestPathObj(path, weight);
    _algorithmResult.add(result);
    return;
  }

  for(GraphPath gPath: lastNode.Paths) {
    if (path.contains(gPath.ToNode.Id) || weight[1] > Settings.SearchDepth)
      continue;
    if (weight[1] == Settings.SearchDepth &&
      (gPath.ToNode.Type != _toNode.Type ||
       gPath.ToNode.RouteId != _toNode.RouteId))
        continue;
    if (ContainsTransferToVisitedNode(lastNode, path))
      continue;
    if (gPath.Delay < 0)
      continue;

    ArrayList newPath = new ArrayList<>(path);
    if (gPath.ToNode.OptimizedNodes.size() > 0)
      newPath.addAll(gPath.ToNode.OptimizedNodes);
    else
      newPath.add(gPath.ToNode.Id);

    int[] newWeight = new int[]{weight[0], weight[1], weight[2]};
    newWeight[0] += gPath.Time + ((addDelay) ? gPath.Delay : 0);
    if (gPath.IsTransfer) {
      addDelayCopy = true;
      newWeight[1]++;
      newWeight[2] += gPath.Cost;
    }
    DepthSearch(newPath, newWeight, gPath.ToNode, gPath, addDelayCopy);
  }
}
```

Для увеличения производительности необходимо осуществить процедуру оптимизации графа. Процедура оптимизации графа заключается в последовательной замене всех вершин и дуг графа, которые подходят под критерии оптимизации, на оптимизированные вершины. Это позволяет значительно сократить количество вершин и дуг графа, особенно если граф имеет мало пересечений между ветвями.

В качестве критериев для оптимизации двух вершин используются следующие утверждения:

- Существуют две вершины, $Node_1$ и $Node_2$ которые соединены дугой *Path.*
- Каждая вершина $Node_n$ не должна иметь более двух дуг *Path.*
- Обе вершины $Node_1$ и $Node_2$ на дуге *Path* принадлежат одному маршруту Route.
- Все смежные с $Node_1$ и $Node_2$ вершины должны принадлежать одному пути.

```
private boolean CheckCriteria(GraphPath path) {
  // Принадлежат ли вершины Node1 и Node2 одному маршруту
  if (path.FromNode.RouteId == path.ToNode.RouteId)
  {
          // Имеет ли вершина Node1 и Node2 бульше двух дуг?
    if (path.FromNode.Paths.size() > 2 || path.ToNode.Paths.size() > 2)
      return false;

    // Принадлежат ли все вершины на пути Path одному маршруту?
    int routeId = path.ToNode.RouteId;
    for (GraphPath nPath: path.ToNode.Paths)
      if (nPath.ToNode.RouteId != routeId)
        return false;
    for (GraphPath nPath: path.FromNode.Paths)
```

```
        if (nPath.ToNode.RouteId != routeId)
          return false;

      return true;
    }
    else
      return false;
}
```

Процесс оптимизации графа состоит из следующих шагов:

1. Копировать граф.
2. Если оптимизация возможна, перейти на шаг 3.
3. Запустить цикл по каждой вершине графа, *Node*, если проверены все вершины, перейти на шаг 7.
4. Если вершина *Node* не помечена как неоптимизируемая, !*Node*.IsOptimizable, перейти на шаг 5, иначе на шаг 3.
5. Запустить цикл по каждому пути *Path* исходящему из этой вершины. Если пути из вершины существуют, перейти на шаг 6, иначе на шаг 3.
6. Если вершина *Node* и вершина, в которую идёт путь *Path* подходят под критерии для оптимальности вершин, выполнить оптимизацию, иначе взять другую вершину и повторить шаг 2.
7. Закончить оптимизацию.

```
public void OptimizeGraph(int maxId) {
  Copy();
  while (OptimizeCycle(maxId))
    maxId++;
}

private boolean OptimizeCycle(int maxId) {
  for (GraphNode node: OptimizedNodes.values())
  {
    if (!node.IsOptimizable)
      continue;

    for (GraphPath path: node.Paths) {
      if (CheckCriteria(path))
      {
        // Создание новой вершины
        // Изменение смежных с оптимизированными вершинами дуг
        // Подсчёт новых критериев оптимальности вершины
        // Исключение оптимизированных вершин и дуг из графа
        // Включение новой вершины и дуг в граф
        return true;
      }
    }
  }

  return false;
}

public boolean ContainsTransferToVisitedNode(GraphNode gNode, ArrayList<Integer> path) {
  if (path.size() > 2)
    for (GraphPath gPath: gNode.Paths) {
      if (gPath.ToNode.Id != path.get(path.size()-2))
        if(path.contains(gPath.ToNode.Id))
          return true;
    }
  return false;
}
```

*Структура хранения данных о пешеходных маршрутах и маршрутах движения транспорта*

Вершины графа можно подразделить на два вида: пешие и остановки общественного транспорта. Оба

вида несут в себе информацию о местоположении вершины, её названии. Так же вершины остановок общественного транспорта имеют задержку, которая определяет время, затрачиваемое на посадку и высадку пассажиров и принадлежность к маршруту. Структура позволяет хранить информацию о двух видах общественного транспорта, с расписанием и без него, а также пеших маршрутах.

Дуги графа можно разделить на три вида: дуги на одном маршруте, дуги, пересекающиеся с маршрутами одного вида транспорт, а и дуги, пересекающиеся с маршрутами разного вида транспорта. Данное разбиение связано с тем, что помимо прямого следования по одному маршруту существуют пересадки, которые имеют свой вес и денежную стоимость.

Дуги на одном маршруте общественного транспорта содержат ссылку на свою позицию в расписании для дальнейшего определения кратчайшего пути, зависящего от него.

Структура хранения общественного транспорта без расписания содержит информацию о путях, станциях и переходах между ними:

1) Маршруты.
   а) Название маршрута.
   б) Цвет маршрута.
   в) Задержка между транспортом в зависимости от времени суток.
2) Остановки.
   а) Название остановки.
   б) Принадлежность остановки к маршруту.
   в) Векторный объект точка с геоинформационными координатами.
3) Пути.
   а) Векторный объект полилиния с двумя геоинформационными координатами связанных станций.
   б) Время, затрачиваемое на преодоление пути.
   в) Цена пути.

Структура данных о движении общественного транспорта без расписания изображена в графическом виде на рисунке 10.

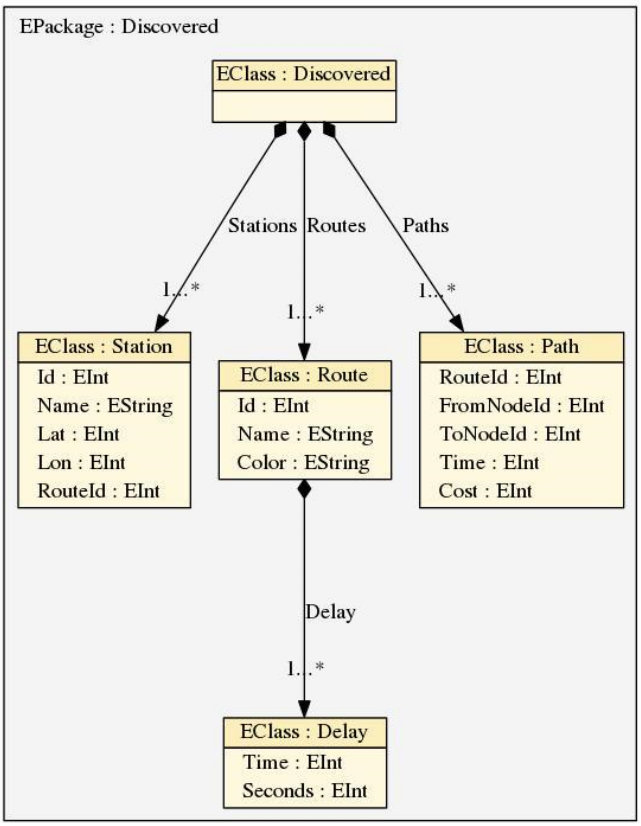


*Рисунок 10 – Структура движения общественного транспорта без расписания*

Структура хранения общественного транспорта с расписанием содержит информацию о маршрутах и остановках:

1) Маршруты.
   а) Название маршрута.

2) Календарь.
   а) Дни недели, по которым работает маршрут.
   б) Начальная и конечная даты функционирования маршрута.
3) Поездки.
   а) Маршрут, к которому принадлежит поездка.
   б) Расписание дней недели, по которому работает поездка.
4) Станции.
   а) Название станции.
   б) Векторный объект точка с геоинформационными координатами.
5) Время остановок.
   а) Остановка, к которой принадлежит время остановки.
   б) Поездка, к которой принадлежит время остановки.
   в) Порядковый номер станции в маршруте.
   г) Время прибытия и отбытия со станции.

Структура данных о движении общественного транспорта с расписанием изображена в графическом виде на рисунке 11.

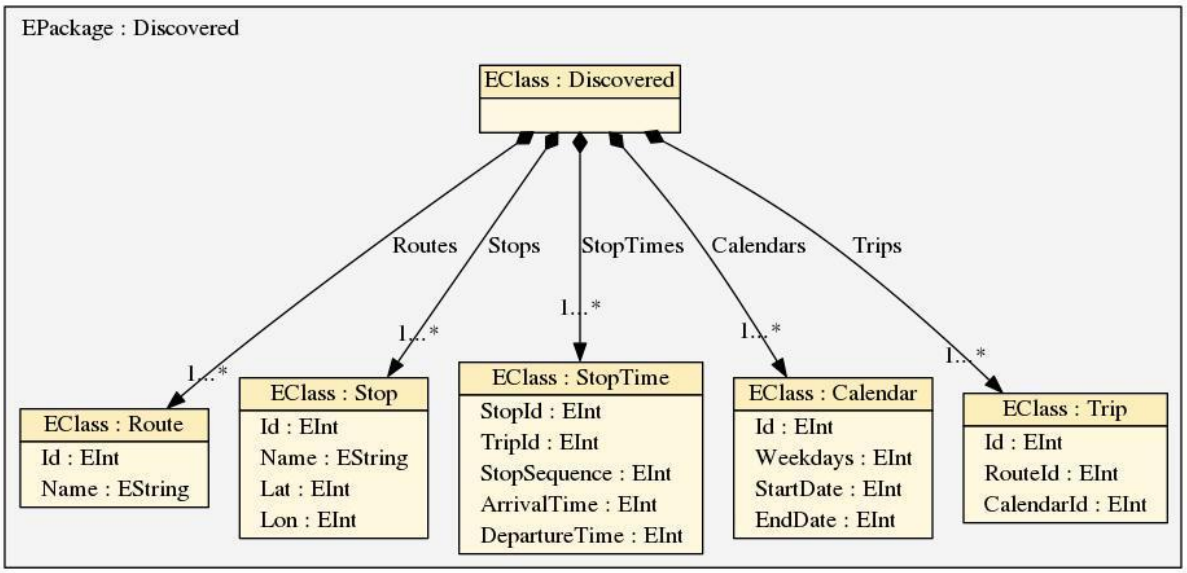


*Рисунок 11 – Структура движения общественного транспорта с расписанием*

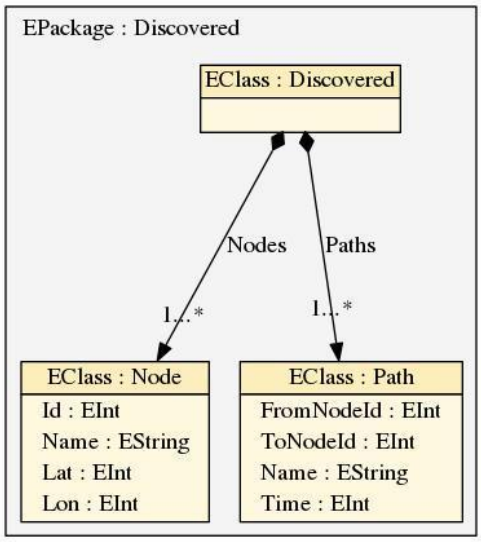


*Рисунок 12 – Структура хранения данных пеших маршрутов*

Структура хранения пеших маршрутов содержит информацию о пеших остановках и связях между ними:

1) Пешие остановки.
   а) Название остановки.
   б) Векторный объект точка с геоинформационными координатами.
2) Связи между пешими остановками.
   а) Названия пути.
   б) Время, затрачиваемое на преодоление пути.
   в) Векторный объект полилиния с двумя геоинформационными координатами связанных станций.

Структура данных пеших маршрутов изображена в графическом виде на рисунке 12.

Структура хранения пересадок содержит информацию о пересадках между разным видом транспорта:

1) Пересадка.
   а) Затрачиваемое на пересадку время.
   б) Векторный объект полилиния с двумя геоинформационными координатами связанных станций.

Структура данных пересадок изображена в графическом виде на рисунке 13.

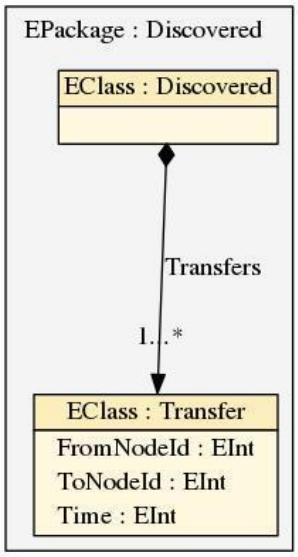


*Рисунок 13 – Структура пересадок*

**Заключение**

1. Разработаны усовершенствованные модели и алгоритмы поиска и оптимизации маршрутов в транспортной сети города и решены некоторые вопросы увеличения их производительности.
2. На основе разработанных моделей и алгоритмов создано мобильное приложение под ОС Android, позволяющее реализовывать различные сценарии пользовательского поведения. Описана архитектура разработанного приложения (база данных, программные интерфейсы и т.д.), его функциональные возможности и модули.
3. Результаты тестирования моделей, алгоритмов и созданного на их основе программного обеспечения показывают, что они являются более оптимальными и быстродействующими по сравнению с существующими.

**Благодарности**

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (РФФИ), грант № 16-37-00373 мол_а, «Разработка перколяционных и стохастических моделей балансировки потоков и управления высоконагруженными транспортными сетями».

**Acknowledge**

The work was supported by the Russian Foundation for Basic Research (RFBR), Grant No. 16-37-00373 mole_a, "Development of percolation and stochastic models of flow balancing and management of highly loaded transport networks".

## Литература

1. Liu L., (2011), "Data Model and Algorithms for Multimodal Route Planningwith Transportation Networks", Technische Universität München, pp. 9-13.
2. Cargal J. M., (1988), "Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff", chapter 9.
3. Dijkstra's Shortest Path Algorithm [Электронный ресурс]. – https://brilliant.org/wiki/dijkstras-short-path-finder/ – (дата

обращения : 14.04.2017)
4. Depth-First and Breadth-First Search [Электронный ресурс]. – https://jeremykun.com/2013/01/22/depth-and-breadth-first-search/ – (дата обращения : 20.04.2017)
5. Алгоритмы на графах: Поиск в глубину (DFS, DLS, IDDFS) [Электронный ресурс]. – http://haru-atari.com/ru/blog/17/algorithms-on-graphs-deep-first-search-dfs-dls-iddfs – (дата обращения : 23.05.2017)

## References

1. Liu L., (2011), "Data Model and Algorithms for Multimodal Route Planningwith Transportation Networks", Technische Universität München, pp. 9-13.
2. Cargal J. M., (1988), "Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff", chapter 9.
3. Dijkstra's Shortest Path Algorithm [Электронный ресурс]. – https://brilliant.org/wiki/dijkstras-short-path-finder/ – (дата обращения : 14.04.2017)
4. Depth-First and Breadth-First Search [Электронный ресурс]. – https://jeremykun.com/2013/01/22/depth-and-breadth-first-search/ – (дата обращения : 20.04.2017)
5. Algoritmy na grafah. Poisk v glubiny (DFS, DLS, IDDFS) [Electronic resource]. – http://haru-atari.com/ru/blog/17/algorithms-on-graphs-deep-first-search-dfs-dls-iddfs – (дата обращения: 23.05.2017)

**Об авторах:**

**Алёшкин Антон Сергеевич** кандидат технических наук, доцент, доцент кафедры автоматизированных систем управления института комплексной безопасности и специального приборостроения, Московский технологический университет (МИРЭА), antony@testor.ru

**Лесько Сергей Александрович** кандидат технических наук, доцент, доцент кафедры моделирования и управления систем института комплексной безопасности и специального приборостроения, Московский технологический университет (МИРЭА), sergey@testor.ru

**Титов Вячеслав Витальевич** студент кафедры моделирования и управления систем института комплексной безопасности и специального приборостроения, Московский технологический университет (МИРЭА), titov@testor.ru

**Note on the authors:**

**Alyoshkin Anton S.,** Candidate of Technical Sciences, Associate Professor of the Department of Automated Control Systems, the Institute of Complex Security and Special Instrumentation, Moscow Technological University (MIREA), antony@testor.ru

**Lesko Sergey A.**, Candidate of Technical Sciences, Associate Professor of Department of Modeling and Control Systems, Institute of Complex Security and Special Instrumentation, Moscow Technological University (MIREA), sergey@testor.ru

**Titov Vyacheslav V.,** the student of the Department of Modeling and Control Systems of the Institute of Comprehensive Security and Special Instrumentation, Moscow Technological University (MIREA), titov@testor.ru