Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
Петрозаводский государственный университет (ПетрГУ)
Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

(подпись соискателя)

Прохоров Илья Сергеевич

Магистерская диссертация

Анализ проблем адаптации алгоритмов упрощения ломаных на мобильных устройствах (на территории г. Петрозаводска)

Направление 01.04.02 — Прикладная математика и информатика

Научный руководитель:
к.т.н., доцент О.Ю. Богоявленская

(подпись руководителя)

Петрозаводск — 2020

Оглавление

Введение

В настоящее время мобильные устройства используются не только как средства связи, но и как инструмент взаимодействия с прикладными программами. Также с каждым годом увеличивается число пользователей, имеющих доступ к сети Интернет с переносных устройств посредством мобильных сетей, что позволяет вести ~~обоюдный~~ обмен информацией между пользователем и приложением в большем промежутке времени.

С увеличением способов получения данных вышеназванными сетями, благодаря улучшению качества сигнала и увеличению площади покрытия, соответственно улучшается и качество этих данных, в частности, сведений о местоположении. Подобная информация является конфиденциальной, однако ее использование приложениями представляет собой огромную пользу в первую очередь для самих пользователей. Так в условиях сильной загруженности городских дорог важно правильно подбирать способы ~~для~~ достижения необходимых ~~мест~~, используя для этого, как можно меньшее количество трафика при отображении маршрута.

Как было сказано выше, при большом количестве данных о передвижениях возникает проблема, как хранения, так и их демонстрации. Для устранения указанных проблем важно сохранять информацию таким образом, чтобы при ее демонстрации пользователю, он мог без труда определить основные точки своего маршрута.

Таким образом, ~~в ходе работы были проведены~~ исследования по упрощению и выделению наиболее важных данных, получаемых на основе изменения дисклокации пользователя, с возможностью их демонстрации и дальнейшего использования в других исследованиях.

Целью работы является разработка алгоритма упрощения ломаных, который также позволит уменьшить количество GPS данных ~~полученных~~ с погрешностью, ~~с~~ интеграци~~ей~~ его в мобильное приложение.

Задачи:

1. Рассмотрение способов получения данных о дислокации пользователя.
2. Разработка оригинального алгоритма упрощения кривой.
3. Интеграция алгоритма в мобильное приложение с возможностью демонстрации результатов в виде разницы между полученными и упрощенными данными.

Глава 1

Мотивация и проблематика

1.1 Погрешность GPS

Погрешность при получении данных о дислокации пользователя посредством GPS зависит от множества факторов. Помимо точности сигнала передаваемого спутниками GPS на погрешность влияют и такие факторы как расположение GPS спутников, блокировки сигнала, атмосферные условия, особенности конструкции устройства, качество приемника и т.д.

Правительство США, предоставляющее возможность работы с GPS гражданским лицам, обязалось транслировать GPS сигнал со средней ошибкой пользовательского диапазона (URE) $\leq$7,8 м с вероятностью 95%, по данным на 11 мая 2016 года данный показатель достиг 0,715 м в 95% случаев [1]. Однако, как было описано выше, получать данные посредством GPS датчиков на мобильных устройствах с данной точностью редко предоставляется возможным. Существует множество причин, по которым данные о дислокации могут поступать с большей погрешностью, например, ошибки связанные с системой GPS:

1. **Ошибки распространения:** ошибки, замедляющие распространение сигнала, когда он проходит через ионосферу и тропосферу.

2. **Ошибки многолучевого распространения:** ошибки, возникающие при отражении сигнала от географических объектов, та-

ких как здания, скалы и т.д. Поскольку отраженный сигнал проходит большее расстояние от спутника до приемника, то его путь занимает дополнительное время, что сказывается на погрешности сигнала, получаемого устройством при его отражении.

3. **Ошибки часов приемника:** ошибки, возникающие при несовпадении часов приемника и спутника.

4. **Ошибки орбиты спутников GPS:** отклонения спутника от точной орбиты, указанной в данных эфемерид, влекут за собой ошибки эфемерид, влияющие на точность определения местоположения приемника.

5. **Количество видимых спутников:** уровень достоверности данных о местоположении пользователя напрямую связан с количеством спутников, находящихся в области его видимости, таким образом, чем меньше спутников видит приемник, тем больше погрешность в определении дислокации.

6. **Геометрия расположения спутников:** углы между находящимися рядом спутниками также влияют на точность данных, так при больших углах между ними точность данных возрастает, а при меньших – убывает.

Также иногда возникают ситуации, при которых передача GPS сигнала происходит без сбоев со стороны транслятора, однако средства для отображения дислокации (сервисы карт) неисправны или имеют неточности, например, неправильно зафиксированные данные о местоположении дорог, зданий, достопримечательностей и т.д.

К тому же большинство современных гражданских устройств использует только одну частоту GPS, в то время как, например, военные приемники используют две, либо взаимодействуют с системами аугментации. Использование двух частот GPS позволяет корректировать искажения сигнала, вызываемые земной атмосферой, что позволяет улучшить получаемые данные о местоположении. А системы аугментации, как например, - WAAS (Wide Area Augmentation System), охва-

тывающая только территорию США, использует наземные станции для внесения корректировок в данные, получаемые по одной GPS частоте.

Разницу в получаемой погрешности между гражданскими и военными приемниками можно увидеть на двух примерах проведенных исследований. Так по данным FAA (Federal Aviation Administration) на 2016 год их одночастотные приемники позволили получать данные о местоположении с погрешностью $\leq$1,891 м в 95% случаев [2]. При исследовании точности гражданских устройств, проведенном в рамках GPS MOOC в ION (Institute of Navigation) в 2015 году, в котором участвовало более 31 000 человек из 192 стран, погрешность данных со смартфонов с поддержкой GPS достигла 4,9 м под открытым небом с ее увеличением вблизи зданий, мостов, деревьев и т.д. [3]

1.2 Объем данных

Совокупный объем производимой и хранимой информации достиг в 2017 году порядка 26 ЗБ [4]. Очевидно, что с течением времени объем таких данных будет только расти. В качестве примера рассмотрим данные о GPS траекториях, создаваемые транспортными средствами в рамках дорог общего пользования и шоссе.

Согласно данным, опубликованным Министерством транспорта США, общее количество пройденных миль за 2018 год в США составило $3,225 * 10^{12}$ миль [5]. Для приблизительной оценки объема данных, необходимых для хранения всего объема этих траекторий, предположим, что ограничение скорости на всех участках дорог составляло 60 миль в час, а частота дискретизации GPS данных была равна 1/60 Гц. Также будем считать, что один замер представляет собой набор из двух координат типа float, исключая символы пробела и переноса строки. На основе названных предположений получим, что в США за 2018 год было сгенерировано не менее 25,8 ТБ данных по GPS траекториям. Очевидно, что подобный результат является лишь грубым предположением, и реальные объемы данных могут быть зна-

чительно ~~больше~~. Исходя из этого, уменьшение объема сохраняемых данных за счет упрощения GPS траекторий, является крайне важной задачей, поскольку позволяет значительно уменьшить расходы на хранение и передачу данных подобного типа.

Помимо задачи упрощения GPS траекторий также важна и проблема избытка информации по ним. Так вышеназванные данные были приведены для дорог общего пользования, понятно, что подобные маршруты являются ~~во многом схожими, поскольку являются~~ частями строго ограниченной дорожной сети, что говорит о нецелесообразности их хранения в полном объеме. Так по данным Федерального управления автомобильных дорог США общая протяженность публичной дорожной сети составила $4,165 * 10^6$ миль в 2017 году [6], а, как уже говорилось ранее, транспортные средства в США преодолели $3,225 * 10^{12}$ миль за 2018 год. Таким образом, получается, что каждая дорога в течение года в среднем посещалась более 774 000 раз, что еще раз подтверждает значимость задачи сжатия данных о маршрутах передвижения. Однако в данной работе сосредоточимся на задаче упрощения GPS траекторий.

Глава 2

Изученные материалы

2.1 API android.location

Данный интерфейс программирования приложений содержит Framework API классы, которые определяют Android сервисы для определения местоположения объекта [7]. При работе над получением координат используется три разных провайдера:

LocationManager.GPS_PROVIDER – провайдер, определяющий дислокацию пользователя на основе данных со спутников. Данный провайдер предоставляет информацию в зависимости от условий, в которых находится объект, запрашиваемый данные, что в некоторых ситуациях может занимать значительные промежутки времени.

LocationManager.NETWORK_PROVIDER – данный провайдер определяет местоположение пользователя на основе данных от вышек сотовой связи и точек доступа Wi-Fi. Результаты предоставляются на основе выполненного сетевого поиска объекта, к тому же затрачивая на получение данных меньшее количество времени, чем вышеназванный провайдер.

LocationManager.PASSIVE_PROVIDER – специальный провайдер для получения геоданных без инициирования средств для исправления местоположения, то есть предоставляющий лишь приблизительные значения. В основном возвращает значения местоположения, полученные с помощью других провайдеров.

Основная работа с определением местоположения происходит посредством взаимодействия с классами Location [8] и LocationManager

[9]. Первый класс предоставляет данные о географическом положении пользователя, помимо демонстрации показателей долготы и широты располагает данными о времени получения координат, скорости движения, расположении объекта на некоторой возвышенности и другие. Вся информация о дислокации создается в соответствии с LocationManager'ом.

Второй класс - LocationManager обеспечивает доступ к сервисам системы определения местоположения. Данная служба позволяет приложениям обновлять данные о нахождении пользователя с некоторой периодичностью или запускать некоторое действие со стороны устройства при приближении к некоторой заданной географической области.

Так же необходимо использование интерфейса LocationListener [10] для получения уведомлений от LocationManager при изменении местоположения. Вызов методов данного интерфейса происходит, если LocationListener был зарегистрирован в службе диспетчера местоположений с использованием метода requestLocationUpdates.

Согласно официальной документации с сайта для Android-разработчиков данный API не является рекомендуемым для использования при получении данных о местоположении.

2.2 Google Location Services API

Данный API является частью служб Google Play, которые построены поверх Android API. Подобные API предоставляют доступ к «Fused Location Provider» [11] – «Провайдер плавного местоположения» вместо названных в API android.location провайдеров. Данный способ определения местоположения позволяет провайдеру автоматически выбирать способ определения дислокации – на основе данных со спутников или мобильных вышек, исходя из точности данных, перемещений пользователя, нагрузки на аккумулятор и т.д. В большинстве случаев подобный подход дает более высокую производительность с точки зрения нагрузки на батарею устройства и большую

точность при определении местоположения объекта. К тому же увеличивается и скорость обмена данными, так как используется единая служба, управляющая постоянным обновлением информации. Также подобный подход предоставляет возможности для использования более сложных функций при работе с данными дислокации, например, geofencing - геозонирование.

Для использования служб геолокации компании Google приложение должно быть подключено к соответствующим службам GooglePlayServicesClient. Для реализации данного взаимодействия между пользователем и приложением определены два интерфейса: GoogleApiClient.ConnectionCallbacks [12] и GoogleApiClient.OnConnectionFailedListener [13], отвечающие за проверку корректного подключения или отключения сервисов и обработки непредвиденных ситуаций обрыва соединения с ними. После обработки подключения к сервисам необходимо указать, что приложению необходимо получать данные при изменении местоположения. Работа с классом LocationManager и интерфейсом LocationListener была описана в предыдущем параграфе API android.location, поэтому данный этап работ здесь будет опущен.

Для взаимодействия и настройки самого FusedLocationProviderApi используется класс LocationRequest [14], содержащий параметры обслуживания запросов данным API. Объекты LocationRequest используются для создания запросов по качеству обслуживания при обновлении местоположения. Так, к примеру, можно использовать запросы о дислокации с высокой точностью - PRIORITY_HIGH_ACCURACY и соответственно задать как можно меньший интервал между получениями координат, что подойдет для отображения передвижения пользователя в режиме реального времени. В случае же, когда необходимо не столько точное определение данных о нахождении пользователя, сколько экономное расходование аккумулятора можно инициировать запросы о местонахождении со значением - PRIORITY_NO_POWER. Для сбалансированного режима нагрузки на батарею устройства и получения данных о дислокации использу-

ется - PRIORITY_BALANCED_POWER_ACCURACY.

Чтобы получать периодические обновления об изменениях в нахождении пользователя, отправляется запрос с использованием клиента местоположения. В зависимости от формы запроса службы определения местонахождения могут вызывать метод обратного вызова, передачу объекта местоположения либо вызов метода, обрабатывающего данные о местоположении. Для данных нужд можно использовать метод onLocationChanged, вызываемый при изменении дислокации. Данный метод содержит объект Location, посредством которого можно производить необходимые действия с информацией о дислокации пользователя, например, демонстрировать пользователю его местоположение или записывать эти данные в файл для дальнейшего использования.

Технология geofencing [15] предоставляет возможность объединения данных о текущей дислокации пользователя с данными о местах, которые могут представлять повышенный интерес. При создании области указывается необходимый участок вокруг интересующего пользователя местоположения, посредством задания координат долготы и широты и добавления к ним радиуса. В результате внесения названных данных создается круговая область, которая позволяет отслеживать пересечения ее границ пользователем, что может быть полезно в некоторых случаях: определение местоположения детей и пожилых людей, ограничение использования огнестрельного оружия в недопустимых зонах, мониторинг присутствия сотрудников в здании компании и т.д. Google сервисы предоставляют возможность создания сразу нескольких активных объектов зонирования для отдельного пользователя, и для каждого из них есть возможность создания индивидуальных настроек генерации оповещений при пересечении границ геозонирования. Пользователь может получать уведомления при совершении входа и выхода из созданных зон повышенного интереса или указать промежуток времени, отслеживание которого важно для данной геозоны.

Таким образом, Google Location Services API дает более обширные

возможности в работе с данными о местонахождении пользователя, посредством улучшенных способов получения данных через обращения к службам Google Play и использования этих способов в зависимости от нужд пользователя.

2.3 Google Maps Android API

Интерфейсы Google Maps Android API доступны в службах Google Play и позволяют добавить в Android приложение карты. API предоставляет возможность просмотра 3D-карт, смену режима их отображения (снимки со спутника, упрощенный режим отображения в виде макетов), просмотр планов зданий, эффективное кэширование данных, удобное использование векторной графики и добавление пользовательских настроек при отображении карт.

В приложении были использованы два основных класса, предоставляющих возможность работы с главными возможностями карты - GoogleMap [16] и MapFragment [17]. GoogleMap является основным классом API Google Maps, фактически он является «точкой входа» для всех остальных методов, позволяющих производить различные действия с картой. Создание экземпляра GoogleMap напрямую невозможно, поэтому для работы с ним использовался метод getMapAsync для получения экземпляра MapFragment, который был добавлен в xml-файл приложения. Так как данный класс является фрагментом, то добавление его компонента - fragment в файл макета активности не создает никаких проблем, обеспечивая при этом все необходимые жизненные циклы для работы.

Для создания связи между GoogleMap и MapFragment используется интерфейс обратного вызова OnMapReadyCallback [18], установленный на необходимый фрагмент карты. Его запуск происходит в случае готовности карты для использования, что позволяет получить непустой экземпляр GoogleMap. Данный интерфейс, как было сказано выше, связан с сервисами Google Play, поэтому в случае их отсутствия на устройстве пользователя будет создано уведомление с воз-

можностью их установки. Также класс GoogleMap помимо возможностей по изменению внешнего вида отображаемой карты, изменения ее положения и т.д. предоставляет возможности по внесению изменений на карту. В частности в приложении использовался метод addPolyline для добавления объекта Polyline, который предназначен в данном случае для построения маршрутов передвижений пользователя.

Сам объект Polyline [19] является классом и содержит разного рода настройки отображения кривой на карте, благодаря чему при демонстрации изначальной и упрощенной ломаных передвижений не возникает проблем с их визуальным распознаванием. Для отображения нынешнего положения пользователя на карте использовался метод addMarker для размещения объекта Marker [20] на экземпляре карты. Данный объект также является классом и содержит собственные настройки отображения, которые также использовались при создании приложения.

2.4 Средства Android

android.app.AlertDialog - подкласс класса Dialog, предоставляющий возможность отображения текстовой строки и создания кнопок для предоставления пользователю возможности выбора между некоторыми действиями. Использовался для вывода предупреждения о необходимости включения определения местоположения и взаимодействия с отображением не упрощенного маршрута передвижения.

android.content.Intent – абстрактное описание для операции, которая должна быть выполнена. В приложении операция используется для перехода к меню включения определения местоположения на телефоне и чтения данных из файла.

android.content.pm.PackageManager - класс, предоставляющий информацию о том, какие пакеты приложений доступны в данный момент на устройстве пользователя. В приложении использовался для проверки разрешений на доступ к данным геолокации.

android.net.Uri - абстрактный класс, создающий и анализирующий URL-ссылки, соответствующие RFC 2396. Использовался для создания связи между файлом с данными о передвижениях пользователя и приложениями на мобильном устройстве, способными взаимодействовать с текстовыми документами.

android.os.Environment – класс, обеспечивающий доступ к переменным окружения. getExternalStorageState() – метод, возвращающий текущее состояние внешнего носителя по заданному пути. Используется для проверки доступа к SD-карте. getExternalStorageDirectory() – метод для получения пути к каталогу с файлом о данных местоположения пользователя.

android.widget.Toast - класс, использующийся для вывода небольшого сообщения пользователю, демонстрирующего, что его действия в приложении были корректно обработаны. В приложении было необходимо для создания оповещений об изменении местонахождения, начале или прекращении записи данных геолокации и некоторых предупреждений.

2.5 Средства языка Java

java.text.SimpleDateFormat – класс, использующийся для форматирования и разложения даты в локально-чувствительной манере.

java.util.Scanner - простой текстовый сканер с возможностью анализа примитивных типов и строк посредством использования регулярных выражений. Использовался для считывания данных из файла о перемещении пользователя.

java.io.BufferedWriter - класс для записи текста в поток вывода с обеспечением эффективной записи символов. Использовался для создания связи между путем к файлу записи и его последующим заполнением данными.

java.io.File - абстрактное представление для имен файлов и каталогов хранения данных. Использовался в приложении при создания пути для последующей записи данных.

java.io.FileInputStream - класс, предоставляющий возможность получения входных данных из некоторого файла в файловой системе пользователя. Необходим в приложении для создания взаимосвязи между файлами с данными о дислокации пользователя и сканером для дальнейшей обработки этой информации.

java.io.FileWriter - класс для записи потока символов в файл. Конструкторы данного класса не содержат проверки на допустимость используемой кодировки и размера файла, поэтому для ручного определения этих значений лучше использовать OutputStreamWriter в FileOutputStream.

java.io.IOException - общий класс для обработки исключений при проведении операций ввода-вывода, которые могут быть вызваны при неудаче или прерывании данных операций.

java.util.Locale - класс, предоставляющий возможность для определения необходимой географической, политической или культурной области при форматировании данных, демонстрируемых пользователю.

Глава 3

Обзор методов упрощения ломаных

3.1 Алгоритмы упрощения ломаной на основе расстояния

3.1.1 Радиальное расстояние

Радиальное расстояние – грубый алгоритм упрощения полилинии. Сложность алгоритма $O(n)$. Суть его заключается в уменьшении количества исходных точек, путем отсеивания тех, которые находятся слишком близко к некоторой вершине, называемой «ключом». Данные «ключи» и образуют упрощенную кривую.

Первая и последняя точки исходного маршрута считаются частью упрощенного маршрута, поэтому изначально считаются «ключами». Начиная с первого «ключа» (первой точки) алгоритм проходит по всем последующим, удаляя те вершины, которые попадают в некоторое заданное расстояние. Первая точка, не попадающая в заданную область, считается новым «ключом». Описанный выше процесс повторяется для всех последующих точек, начиная с нового «ключа», до тех пор, пока не дойдет до конечного «ключа» (последней вершины) [22].

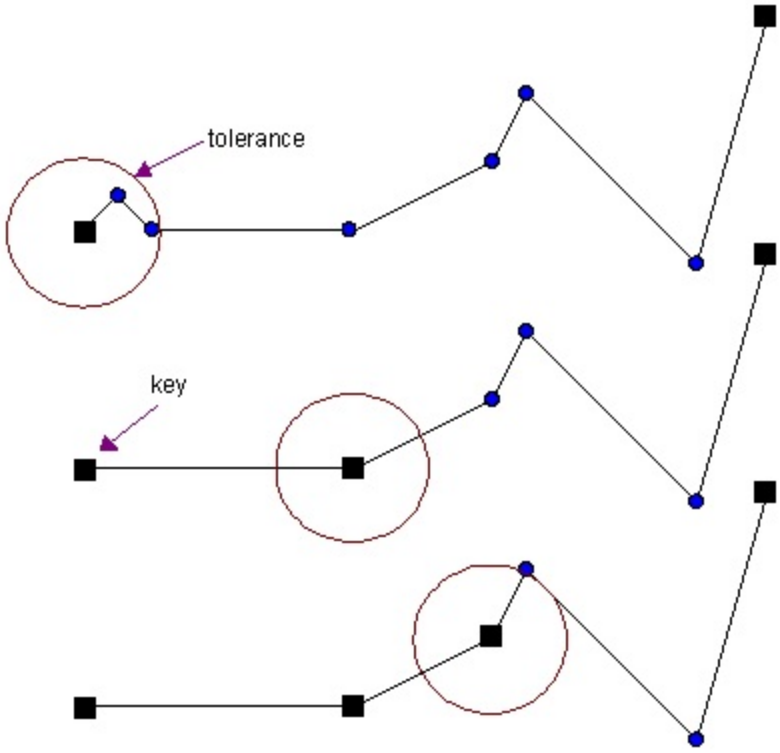


Рис. 1. Радиальное расстояние.

3.1.2 Алгоритм Реуманна-Виткама

Алгоритм Реуманна-Виткама вместо использования радиального расстояния в качестве критерия отклонения применяет перпендикулярный. Сложность алгоритма $O(n)$. Алгоритм начинает работу с определения линии, проходящей через две первые точки исходного набора. Для каждой последующей вершины вычисляется перпендикулярное расстояние от нее до этой линии. Новый «ключ» получается в случае, если расстояние до новой вершины превышает некоторый заданное допустимое значение. Новая линия строится уже между второй и третьей точками для получения последующих точек, которые будут входить в упрощенный маршрут, и процесс повторяется до конечной вершины, которая изначально считается «ключом» [23].

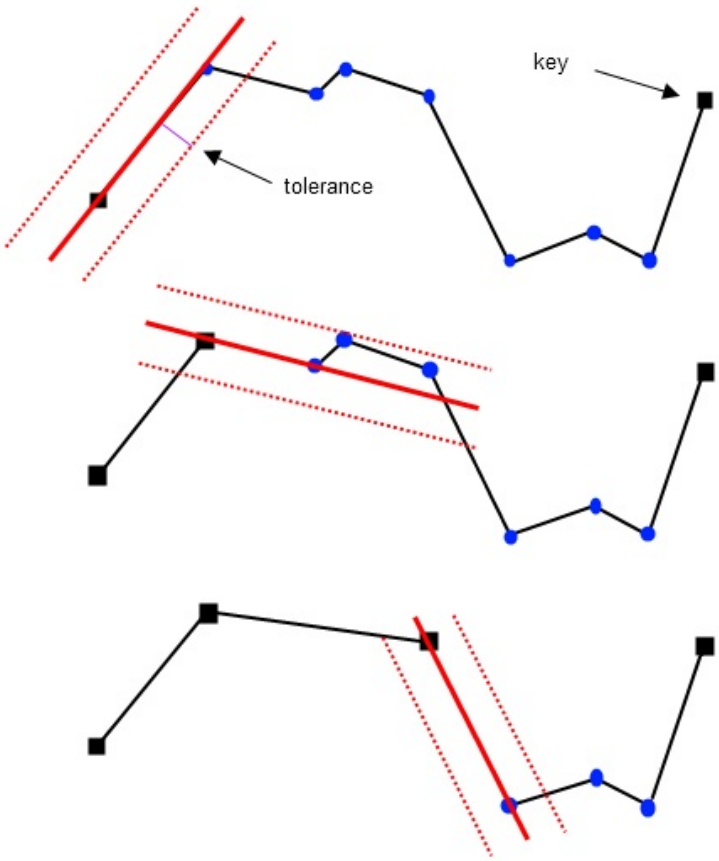


Рис. 2. Алгоритм Реуманна-Виткама.

3.1.3 Алгоритм Ланга

Алгоритм Ланга – алгоритм упрощения исходной кривой, определяющий фиксированный размер поисковой области с последующим вычислением расстояний между этой областью и точками, находящимися между точками, входящими в ее крайние границы. Первая и последняя точка данной области дают линию для вычисления расстояний для каждой промежуточной точки. Если любое вычисленное расстояние больше указанного допустимого значения, то область поиска уменьшается путем исключения из нее последней точки. Процесс продолжается до тех пор, пока все вычисляемые расстояния не будут ниже допустимого или пока не останется промежуточных точек. Все точки, попадающие в допустимую область, отбрасываются, и создается новая поисковая область, начиная с последней точки исходной. Процесс повторяется до тех пор, пока не достигнет конечной точки [24].

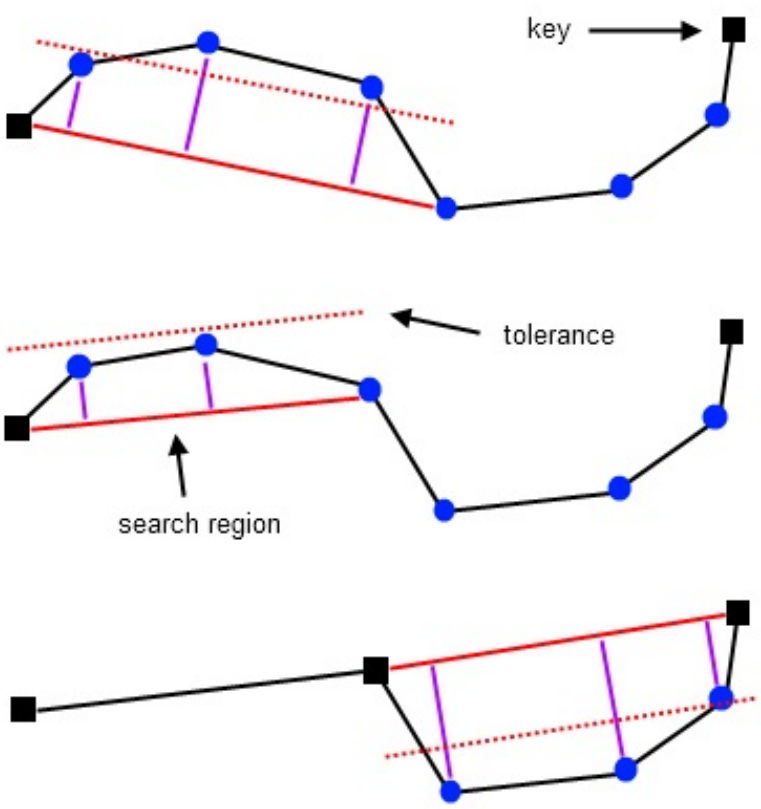


Рис. 3. Алгоритм Ланга.

3.1.4 Алгоритм Дугласа-Пекера

Алгоритм Дугласа-Пекера – алгоритм аппроксимации количества точек кривой, путем измерения расстояние между отрезком, проходящим через первую и последнюю точки, и вершинами, находящимися между ними. Сложность алгоритма составляет $O(n * log(n))$ в среднем и $O(n^2)$ в худшем случае. Алгоритм начинается с грубого упрощения, создавая единственную линию, проходящую через первую и последнюю точки. В дальнейшем вычисляются расстояния для всех промежуточных вершин этой линии. Вершина, наиболее отдаленная от исходной линии и превышающая некоторое заданное допустимое расстояние, будет помечена как «ключ» и добавлена в упрощенный маршрут. В дальнейшем алгоритм рекурсивно вызывает себя на промежутках от начальной до новой точки и от новой до конечной. После всех рекурсивных вызовов строится упрощенная линия, состоящая только из «ключей» [25].

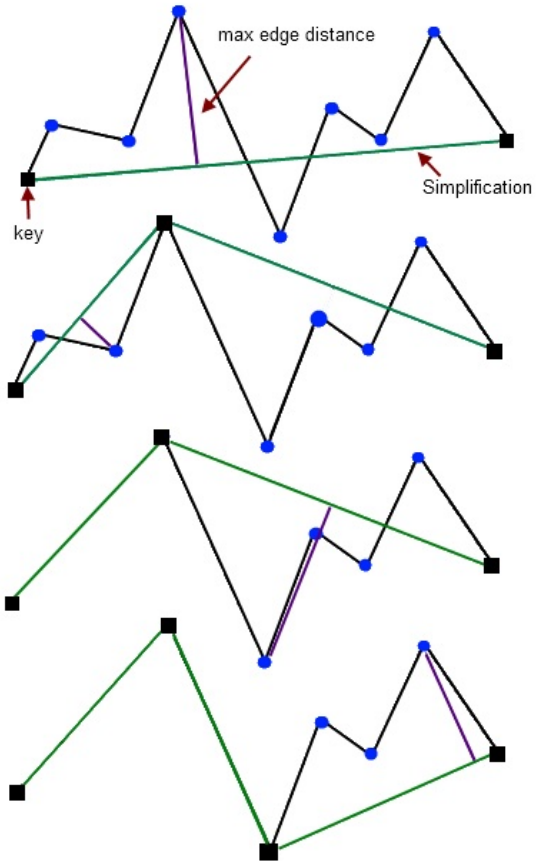


Рис. 4. Алгоритм Дугласа-Пекера.

3.2 Алгоритмы упрощения ломаной на основе скорости

3.2.1 Top-Down Speed-Based ~~Algorithm~~

Данный алгоритм является модификацией алгоритма Дугласа-Пекера, полученной посредством использования пространственно-временной информации, содержащейся в промежутках исходной выборки. Подобную информацию можно получить путем анализа производных скоростей на сегментах полилинии. Большая разница в скорости между двумя концами рассматриваемого промежутка является критерием, на основе которого принимается решение о сохранении точки в упрощенном маршруте. Для этого вводится некоторый порог скорости. Сначала находится точка, в которой разница скорости является наибольшей. После чего эта разница сравнивается с пороговым значением, и, в случае превышения заданного порога, точка записывается в упрощенный маршрут [26].

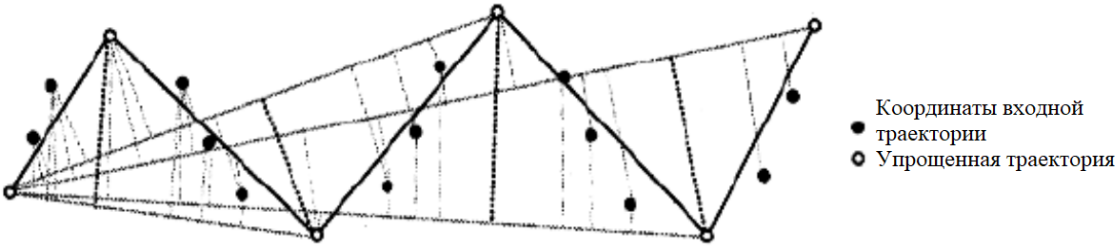


Рис. 5. Top-Down Speed-Based Algorithm.

3.2.2 Dead-Reckoning

Данный алгоритм является интерактивным, т.е. упрощение маршрута происходит в режиме реального времени. Данные, используемые в ходе его работы, приходят от пользователя в традиционном представлении в Moving Objects Databases (MOD) - (location, time, velocity). Основная идея алгоритма заключается в том, что между движущимся объектом и некоторым сервером MOD заключается соглашение, в соответствии с которым связь между ними может быть уменьшена. Другими словами пользователь указывает пороговое значение E, являющееся некоторой областью, в которой пользователь не сообщает данные о своем местоположении.

При отправке данных на сервер MOD пользователь помимо стандартных данных о своем местоположении также указывает информацию о предполагаемой скорости. Исходя из этих данных, сервер создает предполагаемую траекторию движения данного пользователя – бесконечный луч, исходящий из указанных координат дислокации, получаемый путем экстраполяции из вектора скорости. Пользователь, зная о своем фактическом местоположении (например, посредством GPS мобильного устройства), не передает новых сообщений серверу, пока не отклоняется от предполагаемой траектории больше, чем на E. В случае отклонения от траектории больше, чем на E, пользователь информирует сервер MOD о своем текущем местоположении, и сервер вновь строит допустимую траекторию его движения [27].

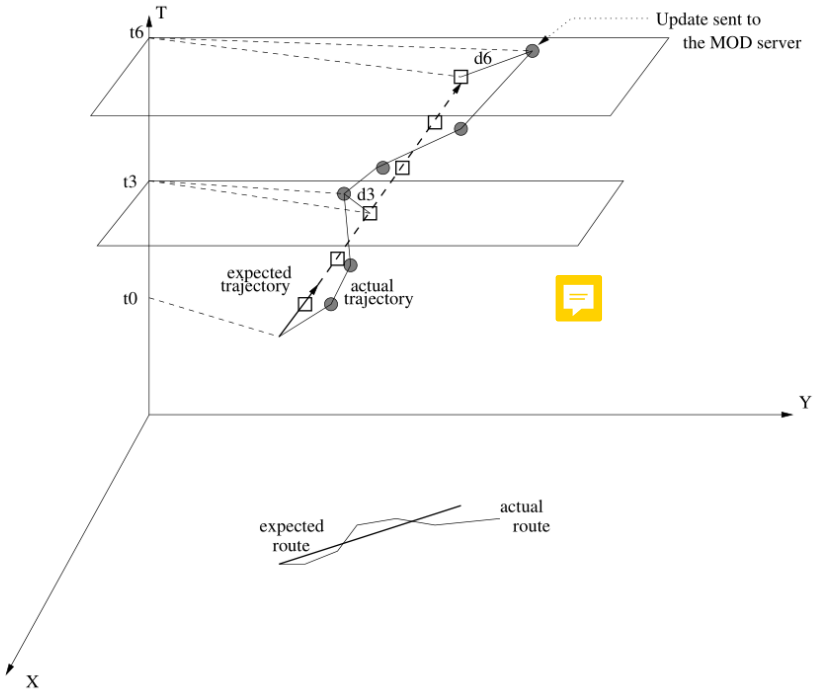


Рис. 6. Dead-Reckoning.

3.3 Алгоритмы упрощения ломаной на основе семантики дорожной сети

Определим траекторию T как путь движения объекта в пространстве с функцией времени, т.е. последовательность, содержащую как пространственную, так и временную информацию. Традиционным подходом представления подобной траектории является набор из n троек вида: $\{(x_1, y_1, t_1), (x_2, y_2, t_2), ..(x_n, y_n, t_n)\}$, где (x_i, y_i) – позиция объекта в двумерном евклидовом пространстве в момент времени t_i. Однако подобное представление траектории будет не слишком удобным в случае, если поступит, например, запрос о средней скорости движения объекта, т.к. тройки (x_i, y_i, t_i) и (x_j, y_j, t_j) не содержат информацию о расстоянии между i и j. Поэтому для представления информации о траектории целесообразнее будет использовать кортеж: (d_i, t_i), где d_i – информация о расстоянии, пройденном объектом.

Исходя из вышесказанного, введем следующие определения. Дорожная сеть - это ориентированный граф G = (V, E), где V – множество вершин (перекрестков) дороги, а E – множество ребер (отрезков) дороги. Вес ребра e_i, обозначим как $w(e_i)$, может быть физическим

расстоянием, временем в пути или другими затратами в соответствии с контекстом приложения.

Пространственный путь для траектории в дорожной сети является последовательностью последовательных ребер. Траектория, представленная на рис. 7, последовательно проходит через ребра e_1, e_2, e_3, e_4 и e_5, соответственно ее пространственный путь с учетом весов ребер может быть представлен в виде $\{w(e_1), w(e_2), w(e_3), w(e_4), w(e_5)\}$.

Временной путь хранит информацию о нахождении объекта в конкретный момент времени. Например, тройка из традиционного представления траектории (x_i, y_i, t_i) говорит о том, что местоположение объекта в момент времени t_i было (x_i, y_i). Однако вследствие перехода от традиционного представления траектории к кортежам заменим данное представление на $(w(e_i), t_i)$.

Таким образом, совокупность пространственного и временного путей рассматриваемой траектории будет выглядеть следующим образом: $\{(0, t_1), (w(e_1), t_2), (w(e_1) + w(e_2), t_3), (w(e_1) + w(e_2) + w(e_3), t_4), (w(e_1)+w(e_2)+w(e_3)+w(e_4), t_5), (w(e_1)+w(e_2)+w(e_3)+w(e_4)+w(e_5), t_6)\}$.

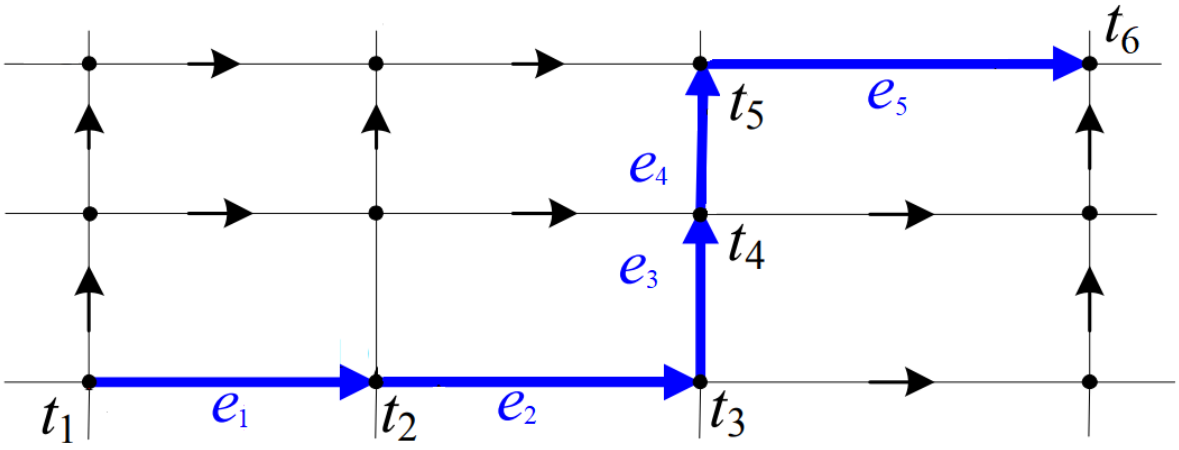


Рис. 7. Траектория движения объекта.

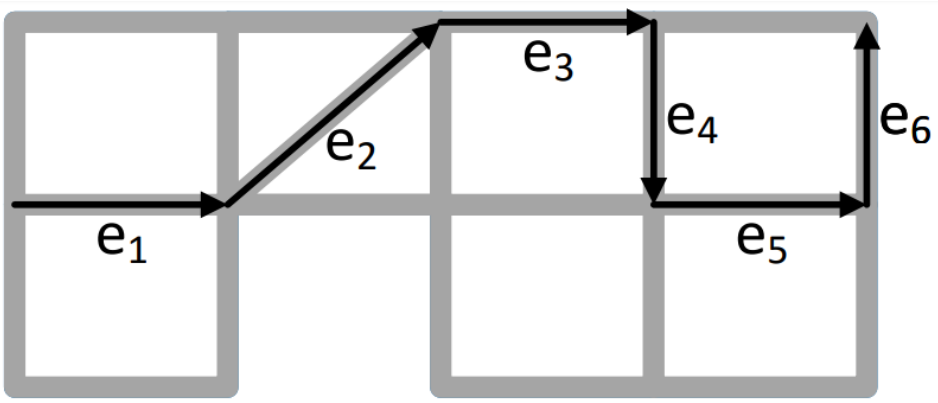
3.3.1 Shortest Path Compression

Упрощение исходного пути в SPC [28] основывается на предположении, что движущийся объект всегда стремится двигаться по кратчайшему пути для достижения пункта назначения. Алгоритм удаляет подтраекторию из исходной траектории в случае ее совпадения с кратчайшим путем.

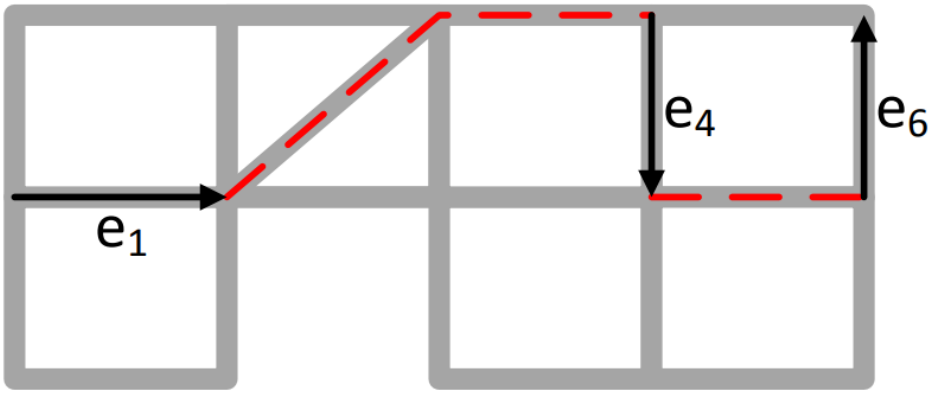
~~Изначально~~ предполагается, что информация о кратчайших путях доступна посредством предварительной обработки дорожной сети. Получение данной информации может быть достигнуто посредством любого из алгоритмов поиска кратчайшего пути. В случае наличия нескольких кратчайших путей между парой рассматриваемых ребер записывается лишь один из них для устранения неопределенности во время упрощения.

Первый – e_1 и последний - e_n сегменты рассматриваемой траектории однозначно попадают в конечный набор упрощенной траектории. После запоминания первого сегмента SPC проверяет соответствие подтраектории между первым и третьим сегментами дороги кратчайшему пути: $\{e_2\} = shortest\_path\_1$. При нахождении совпадения в подтраекторию добавляется следующее ребро дороги до появления несоответствия кратчайшему пути: $\{e_2, e_3\} = shortest\_path\_2$. При расхождении подтраектории с кратчайшим путем SPC удаляет ее и заносит в набор упрощенной траектории сегмент, который вызвал данное расхождение: $\{e_2, e_3, e_4\} \neq shortest\_path\_3$. Процесс повторяется до достижения конечного сегмента рассматриваемой траектории.

Таким образом упрощенная траектория будет представлена, как: $\{e_1, e_4, e_6\}$. При декомпрессии SPC просто восстанавливает кратчайшие пути между несвязанными участками дороги. Сложность алгоритма составляет $O(|T|)$, где $|T|$ – количество ребер входной траектории.



(а) Исходная траектория



(б) Упрощенная траектория

Рис. 8. Shortest Path Compression.

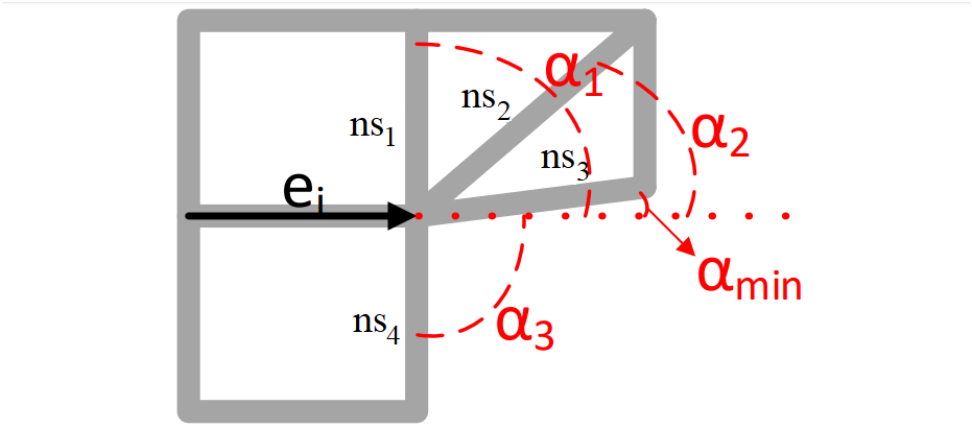
3.3.2 Following Path Routing ~~Algorithm~~

FPRA [28] упрощает исходный путь, исходя из предположения, что движущийся объект стремится двигаться по пути с наименьшим отклонением от нынешней траектории при движении к конечному пункту. Проще говоря, алгоритм удаляет пути, находящиеся на прямой или незначительно отклоняющиеся от нее.

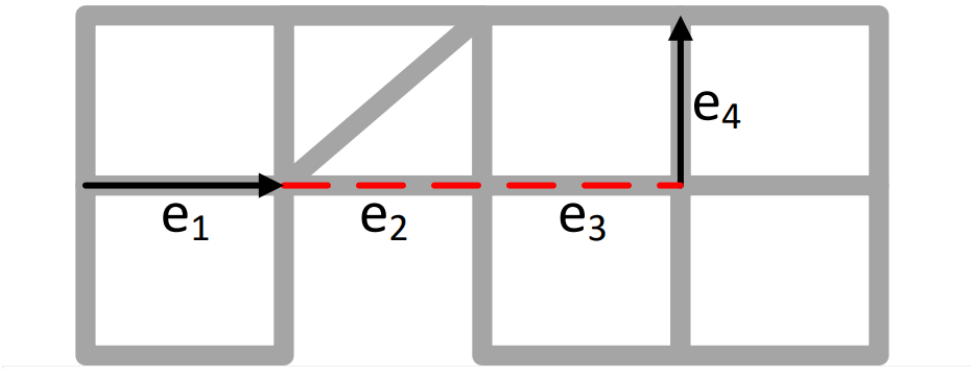
Каждый сегмент дороги имеет некоторое множество возможных путей, идущих за ним, обозначим его, как $NS = \{ns_1, ns_2, ..\}$, т.е. предполагается, что следующий за e_i сегмент дороги e_{i+1} будет соответствовать одному из элементов NS. Между рассматриваемым сегментом и его множеством возможных следующих путей существует угловой коэффициент, обозначим его за α_k. Также существует неко-

торое допустимое угловое отклонение, обозначим за α_{min}.

Внесение первого и последнего сегмента в упрощенную траекторию аналогично SPC. Далее FPRA вычисляет угловой коэффициент между первым – e_1 и вторым - e_2 сегментами, если вычисленный коэффициент соответствует: $\alpha_1 \leq \alpha_{min}$, то он обозначается как «сжигаемый», в противном случае e_2 будет записан в набор упрощенной траектории. На следующем этапе происходит сравнение углового коэффициента между e_2 и e_3 с α_{min}. Процесс продолжается до конечного сегмента рассматриваемой траектории. Все сегменты, помеченные как «сжигаемые», не попадают в набор упрощенной траектории. Упрощенная траектория: $\{e_1, e_4\}$. Декомпрессия и сложность FPRA аналогичны SPC.



(а) Угловые коэффициенты для возможных путей



(б) Упрощенная траектория

Рис. 9. Following Path Routing Algorithm.

3.3.3 Map-matched Trajectory Compression

В MMTC [29] используется сразу два типа алгоритмов - алгоритмы сопоставления данных с картой и алгоритмы упрощения траектории с потерями. Также он использует отдельную функцию оценки, именуемую «траектория сходства», которая гарантирует идентичность упрощенной траектории исходной.

После проведения этапа сопоставления с картой предполагается, что все точки входной траектории точно спроецированы на вершины дорожной сети, а кратчайшие пути между вершинами дорожной сети могут быть вычислены. Исходя из данных предположений, MMTC заменяет определенные части исходной траектории кратчайшими путями в дорожной сети для уменьшения стоимости хранения данных.

Для проведения этапа упрощения MMTC предлагает интерактивный и автономный алгоритмы, которые заменяют определенные подтраектории на кратчайшие пути, если последние содержат меньшее количество вершин: $length(\{e_4, e_5, ..e_{14}\}) = 11$, $length(shortest\_path) = 7$. Следовательно, упрощенная траектория: $\{e_1, e_2, e_3, e_{15}\}$. Автономный алгоритм вычисляет все кратчайшие пути для каждой пары точек, и его сложность составляет $O(n^2 * \log(n))$. В то время как интерактивный алгоритм вычисляет кратчайшие пути только между r точками и его сложность составляет $O(r * n * \log(r))$, где r – заданное количество точек. На выполнение упрощения посредством автономного алгоритма тратится большее количество времени, однако интерактивный алгоритм не так эффективен с точки зрения упрощения данных. К тому же предположение о точной проекции точек исходной траектории на вершины дорожной сети приводит к отклонению от заданной траектории, т.к. большинство точек после проекции попадают не на вершины, а на ребра сети. Таким образом, упрощенная траектория может сильно отличаться от исходной.

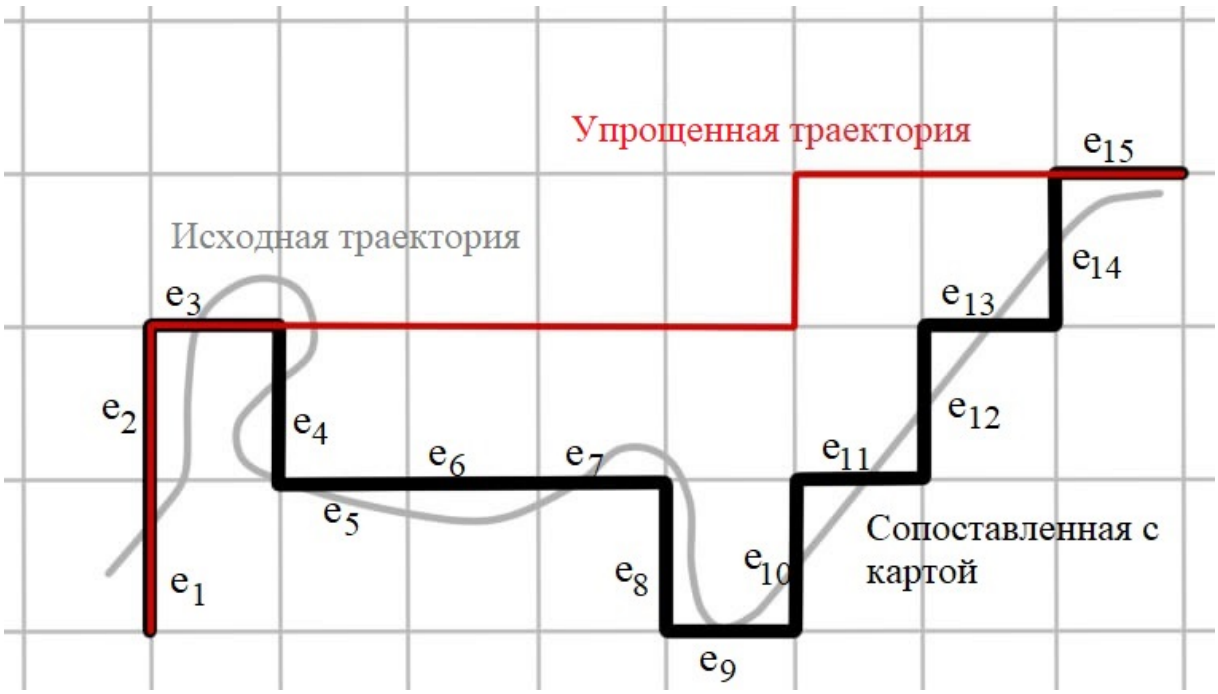


Рис. 10. Map-matched Trajectory Compression.

3.3.4 PRESS

Упрощение траектории в фреймворке PRESS [30] происходит посредством пяти компонентов: «сопоставитель» с картой, средство реформатирования траектории, пространственный и временной компрессоры и процессор запросов (см. рис. 11). На начальном этапе «сопоставитель» с картой принимает исходную GPS траекторию и отображает каждую ее подтраекторию в последовательность ребер дорожной сети. На этапе реформатирования полученная последовательность разделяется на пространственную траекторию и временной ряд. Пространственный путь упрощается с помощью соответствующего компрессора, который работает на основе Hybrid Spatial Compression (HSC) алгоритма. Временной ряд сжимается своим компрессором, работающим на основе алгоритма Bounded Temporal Compression (BTC). Работа обоих компрессоров происходит параллельно. На финальном этапе упрощенные траектория и ряд передаются в процессор запросов для дальнейшего их использования в нуждах приложения.

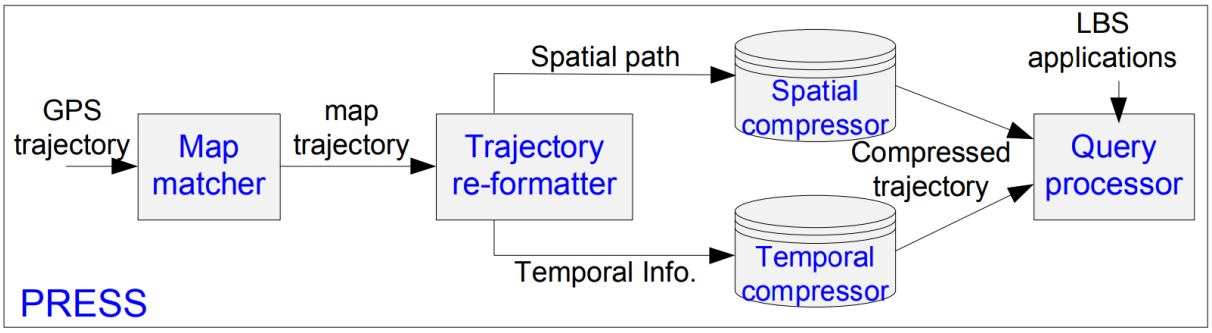


Рис. 11. Структура фреймворка PRESS.

Как правило, основная проблема упрощения состоит в том, что чем выше степень упрощения, тем ниже качество упрощенной траектории для дальнейшего использования. Авторы фреймворка PRESS считают, что их разработка решает данную проблему с трех разных точек зрения. Во-первых, PRESS считает, что пространственная и временная информация о траекториях имеют разные особенности, следовательно, их разделение является стратегически важным этапом. Во-вторых, используемый алгоритм упрощения пространственной траектории без потерь HSC эффективно упрощает данный путь, используя значительно меньше места, не теряя какой-либо важной информации об исходном пути. Работа HSC разделена на два этапа. На первом этапе происходит упрощение, основанное на поиске кратчайшего пути, т.е. в случае, если рассматриваемая подтраектория от ребра e_i до e_j совпадает с кратчайшим путем, то она заменяется им. Второй этап основан на frequent sub-trajectory (FST) кодировании. Основная его идея состоит в разложении траектории на последовательность FST, каждый элемент которой представлен уникальным кодом (например, с помощью кодов Хаффмана). Чем «популярнее» FST, тем короче его код и тем больше экономия места. Используемый для временного упрощения алгоритм BTC с ограниченными ошибками является очень гибким, т.к. может упрощать временную информацию на основе допустимой степени ошибки, являющуюся разной для различных приложений. Таким образом, оба используемых алгоритма упрощения гарантируют высокое качество упрощенной траектории. И, в-третьих, PRESS поддерживает многие популярные пространственно-

временные запросы, используемые в сервисах определения местоположения.

3.4 Другие

Названные выше подходы к упрощению ломаной являются наиболее распространенными и изученными, однако, не единственными. Помимо них существуют и другие алгоритмы упрощения, которые не попадают под выделенные категории.

Одним из примеров подобных алгоритмов является способ упрощения траектории на основе очереди с приоритетами. При данном подходе выбирается наилучшее подмножество точек в исходном маршруте. Процесс выбора такого подмножества происходит посредством удаления из него избыточных и несущественных точек, для чего используются стратегии локальной оптимизации. Упрощение производится до выполнения некоторого условия остановки.

Данный способ упрощения применяется в SQUISH (The Spatial QUalIty Simplification Heuristic), предложенном Макеллом и др.[31] Позднее была представлена улучшенная версия данного алгоритма SQUISH-E (Spatial QUalIty Simplification Heuristic-Extended)[32], в которой была добавлена настройка упрощения на основе степеней сжатия и ошибки.

Другим примером подобных алгоритмов является алгоритм сохранения направления, применяемый в фреймворке DPTS (Direction-Preserving Trajectory Simplification), разработанном Лонгом и др.[33] Данный алгоритм разрабатывался как альтернатива традиционному способу упрощения на основе сохранения позиции, поэтому базируется на идее сохранения направления. Его суть состоит в том, что угловая разница в каждый момент времени между направлением движения, представленном в исходной траектории, и направлении движения, полученном после упрощения, не превосходит заданной степени ошибки.

3.5 Достоинства и недостатки алгоритмов упрощения

Алгоритмы упрощения траектории на основе расстояния являются быстрыми, простыми и эффективными с точки зрения ресурсозатратности. К тому же они легко реализуются, т.к. не требуют сложных вычислительных действий.

Однако очевидным недостатком данных алгоритмов является игнорирование особенностей входной траектории, поскольку их логика базируется лишь на взаимодействии с расстоянием, что может привести к появлению серьезных ошибок, особенно при работе с GPS данными в рамках дорожной сети. Так важные точки маршрута передвижений пользователя, например, точки поворота, могут не попасть в упрощенный маршрут, т.к. расстояние до них не превышало допустимого значения. В то же время точки, которые не несут важной информации о передвижениях, напротив, могут попасть в упрощенный маршрут, поскольку расстояние до них превысило допустимое значение.

Алгоритмы упрощения траектории на основе скорости обладают достоинствами в плане эффективности и недостатками, связанными с возможностью нарушения геометрических особенностей маршрута, ч[illegible] и алгоритмы на основе расстояния.

Очевидно, что схожесть недостатков данных алгоритмов с недостатками алгоритмов на основе расстояния связана с минимальными отличиями в их логике, т.к. вместо расстояния в большинстве из них используется значение скорости. Однако при использовании других характеристик скорости, как например, ее направление или ускорение, аналогичных проблем можно было бы избежать.

Алгоритмы упрощения траектории на основе семантики дорожной сети предоставляют достаточно эффективные методы взаимодействия с GPS траекторией в рассматриваемой задаче, т.к. учитывают геометрические свойства дорог в городской среде. Вследствие чего упрощенная траектория получается более достоверной и

приближенной к реальным условиям.

Однако трудозатратность использования данных алгоритмов весьма велика, поскольку для их работы необходимо либо проведение предварительной обработки входной траектории посредством сопоставления с картой, либо наличие характеристических данных дороги на рассматриваемом участке. К тому же не все алгоритмы гарантируют идентичность упрощенного маршрута исходному, что может привести к осложнениям при идентификации реальных передвижений пользователя.

Глава 4

Разработка специализированного алгоритма упрощения ломаной

4.1 Обозначения

«Выброс» - точка, не являющаяся значимой для данного маршрута передвижения, полученная в результате сбоя в системе определения местоположения, неточности данных на рассматриваемом промежутке дороги, либо других воздействий, блокирующих точное определение дислокации, представляет собой поворот в маршруте следования пользователя, при отсутствии оного.

«Выброс направления» - точка, полученная аналогичным образом, что и обычный выброс, представляющая собой смену направления движения пользователя на противоположное, при отсутствии оного.

«Главная» точка – точка, являющаяся либо началом, либо концом маршрута, либо точкой изменения направления движения пользователя. Не является «выбросом» или «выбросом направления».

Незначимая точка – Незначимая точка – координаты передвижения пользователя из исходного маршрута, не принадлежащие ни к одной из вышеперечисленных категорий выбросов и не являющиеся «главными».

«Упрощенный» маршрут – упрощенная кривая передвижений пользователя, содержащая лишь важные точки маршрута.

4.2 Постановка проблем определения местоположения

В основе алгоритма лежит ~~идея~~ упрощения кривой ~~посредством~~ определения направления движения пользователя и фикс~~ирования лишь тех~~ точек, в которых это направление изменилось. Упрощение кривой передвижений пользователя происходит без изменений входных данных, таким образом все точки, получаемые в результате работы алгоритма, ~~являются теми же точками, что были предоставлены пользователем изначально.~~

Перейдем к описанию проблем определения местоположения.

1. При считывании изменений дислокации пользователя были выявлены некоторые координаты, которые не являлись точками маршрута, а были лишь «выбросами», полученными в результате неточности систем распознавания местоположения.

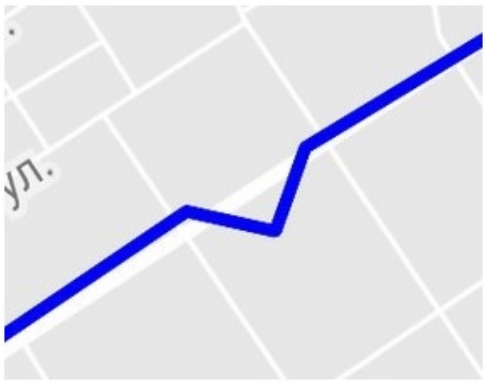


Рис. 12. «Выброс».

2. При анализе данных передвижений были выявлены «выбросы направления», которые показывали, что направление движения пользователя сменилось на противоположное, хотя подобного не происходило.

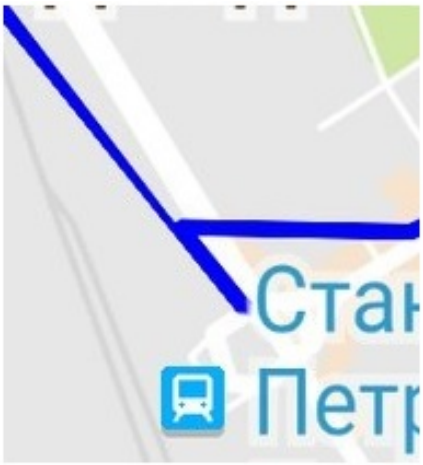


Рис. 13. «Выброс направления».

Как видно из описания данных проблем, добиться их исключения на этапе получения данных невозможно, т.к. на этапе проведения замеров подобные точки теоретически могли являться сменой направления движения пользователя. Вследствие чего устранение подобных ошибок определения местоположения необходимо производить на этапе упрощения маршрута.

Помимо ликвидации вышеназванных проблем на этапе упрощения также было решено отсеивать точки, располагающиеся близко друг к другу, т.е. полученные, например, в результате перемещения пользователя на территории остановки общественного транспорта. Данное решение мотивируется тем, что количество подобных точек может быть достаточно большим, однако ~~ценность~~ их ~~наличия~~ для идентификации передвижений пользователя наоборот мала, несмотря на то, что среди них могут иметься точки, предшествующие смене направления движения.

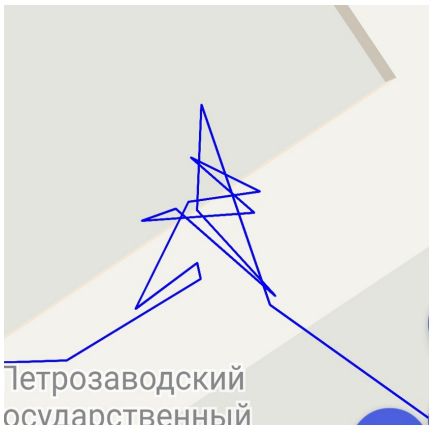


Рис. 14. Незначимые точки.

При получении данных о дислокации была выявлена следующая ситуация: координаты местоположения пользователя в данный момент совпадали с его местоположением при предыдущем получении данных, что могло быть связано с нахождением пользователя в одной точке, либо «выбросом». Для исключения подобных ситуаций было решено производить проверку на несовпадение данных о дислокации пользователя в данный и предыдущий моменты на этапе получения данных о передвижениях, т.к. очевидно, что в подобных данных нет необходимости, и их отсев можно произвести еще на этапе формирования маршрута.

Таким образом, устранение всех названных выше проблем, кроме последней, является одной из задач разработанного алгоритма, что в совокупности с задачей упрощения позволяет построить «упрощенный» маршрут, в котором отсутствуют «выбросы», «выбросы направления» и незначимые точки. Далее в этом разделе дадим определение поставленной задачи, приведем описание разработанных алгоритмов и представим полученные результаты.

4.3 Постановка задачи

Определим исходный маршрут передвижений пользователя как множество вида: $P = (p_1, p_2, \ldots, p_n)$, где n - количество точек в первоначальном маршруте. Элемент p_i представляет собой пару (x_i, y_i), где x_i и y_i - широта и долгота местоположения пользователя в момент i соответственно. Будем называть P' - «упрощенным» маршрутом передвижений, который образуется упорядоченной последовательностью вершин из P так, что P' является «схожей» с P. Полное определение P' дадим позднее.

Введем множество «главных» точек P*.

«Выбросы», «выбросы направления» и незначимые точки обозначим как множество P^-, а их количество - $count_{P^-}(P)$.

Определим угол поворота между сегментами hp_i и p_ip_{i+1}, где $h \subset P'$ или $h = p_{i-1}$, следующим образом. Угол поворота на сегменте hp_i

обозначим как $angle\_cur = atan2(y_i - y_h, x_i - x_h)$, а на сегменте p_ip_{i+1} – $angle\_next = atan2(y_{i+1} - y_i, x_{i+1} - x_h)$, тогда угол поворота между сегментами hp_i и p_ip_{i+1}: $angle(hp_i|p_ip_{i+1}) = |angle\_next - angle\_cur|$.

Расстояние между парой точек p_i и p_{i+1} обозначим как $dist(p_i|p_{i+1})$ и для его расчета будем использовать формулу haversine:

$R = 6371 * 10^3$ - радиус Земли в метрах

$\Delta\varphi = x_{i+1} - x_i$

$\Delta\lambda = y_{i+1} - y_i$

$a = \sin^2(\Delta\varphi/2) + \cos x_i * \cos x_{i+1} * \sin^2(\Delta\lambda/2)$

$c = 2 * atan2(\sqrt{a}, \sqrt{1-a})$

$dist(p_i|p_{i+1}) = R * c$

Текущим направлением движения на сегменте p_ip_{i+1} будем называть пару $(dir\_x(p_ip_{i+1}), dir\_y(p_ip_{i+1}))$ такую, что $dir\_x(p_ip_{i+1}) = sign(x_{i+1} - x_i), dir\_y(p_ip_{i+1}) = sign(y_{i+1} - y_i)$.

Исходя из введенных обозначений, дадим определение «упрощенного» маршрута.

«Упрощенным» маршрутом называется подмножество P' множества P, такое что $P' = (p_1 = p'_1, p'_2, \ldots, p'_m = p_n)$, где m < n, элементы которого удовлетворяют следующим условиям:

1. Угол поворота для двух последовательных сегментов hp_i и p_ip_{i+1} не меньше α, т.е. $angle(hp_i|p_ip_{i+1}) > \alpha$, где $\alpha > 0$ - минимально допустимый угол поворота.

2. Расстояние между элементами p_i и p_{i+1} из P не меньше ε, т.е. $dist(p_i|p_{i+1}) > \varepsilon$, где $\varepsilon > 0$ - средняя ширина дороги.

3. Отношение направлений движения между сегментами $p_{i-1}p_i$ и p_ip_{i+1} из P может быть описано, как $(dir\_x(p_{i-1}p_i) \neq dir\_x(p_ip_{i+1}))XOR(dir\_y(p_{i-1}p_i) \neq dir\_y(p_ip_{i+1}))$.

4. $count_{p^-}(P') \to 0$.

5. Элементы множества $P' : p' \in P^*$.

4.4 Описание алгоритма версии 1

~~Разрабатываемый~~ алгоритм работает по принципу нахождения отклонений от направления движения пользователя, создающегося на основе «главной» точки и первой идущей за ней. Отклонение определяется ~~посредством~~ сравнения модуля разности начального направления движения с последующими, то есть получаемых на основе «главной» точки и идущих после нее через один шаг и далее. В случаях, когда значение модуля не превосходит константы α – радиуса перекрестка, алгоритм переходит к рассмотрению следующего направления движения, тем самым отбрасывая точки, находящиеся примерно на одной прямой.

В случаях, когда направление движения меняется, в «упрощенный» маршрут заносится точка предшествующая смене направления маршрута. Для отсечения точек, являющихся «выбросами», занесение в «упрощенный» маршрут происходит не сразу, а только после сравнения еще одного направления движения.

Если следующее направление соответствует тому, которое предшествовало предполагаемой смене направления, то точка отбрасывается.

Для изъятия «выбросов направления» производится проверка направления посредством взятия сигнума от него. Так если сигнум меняется по одной из координат на одном шаге, но потом возвращается к исходному значению либо меняется по обеим координатам, то данная точка также отбрасывается, как и обычный «выброс».

Для отсечения незначимых точек производится проверка на расстояние между координатами. В случаях, когда расстояние не превышает константы ε – ширины дороги, точка не вносится в «упрощенный» маршрут.

Таким образом, посредством всего одного прохождения алгоритма по массиву, содержащему маршрут передвижений пользователя, получается «упрощенный» маршрут, соответствующий поставленной задаче.

4.5 ~~Словесное~~ описание алгоритма версии 1

Обозначения:

$main\_point$ - координаты «главной» точки на данном участке дороги (точки поворота или начальной).

$prev\_point$ - координаты $n-1$-ой точки.

now_point - координаты n-ой точки.

$next\_point$ - координаты $n+1$-ой точки.

$angle$ - угол между «главной» точкой и $n+1$-ой точкой, пересчитывается на каждом шаге алгоритма.

$sign\_main$ - сигнум направления движения пользователя, вычисляемый между «главной» и следующей за ней точками, пересчитывается в случае изменения «главной» точки.

$sign\_now$ - сигнум направления движения пользователя, вычисляемый между $n+1$-ой и n-ой точками, пересчитывается на каждом шаге алгоритма.

$dist$ - расстояние между $n-1$-ой и n-ой точками, пересчитывается на каждом шаге алгоритма.

ε - наибольшее допустимое расстояние между точками, полученное опытным путем.

$angle\_err\_max$ - верхняя граница модуля недопустимой разности между «главным» углом и нынешним.

$angle\_err\_min$ - нижняя граница модуля недопустимой разности между «главным» углом и нынешним.

Алгоритм:

1. Получить координаты первой и второй точек.
2. Внести координаты первой точки в массив с «упрощенным маршрутом».
3. Определить координаты первой точки как x_main, y_main – $main\_point$.
4. Определить и запомнить угол между первой и второй точками - $angle$.

5. Определить и запомнить сигнум между $main\_point$ и второй точкой – $sign\_main$.

6. Если координат больше нет, то перейти к 11.

7. Получить координаты $n + 1$-ой точки.

8. Определить переменные $prev\_point$ = координаты $n - 1$ точки, now_point = координаты n точки, $next\_point$ = координаты $n+1$ точки.

9. Если модуль разницы между $angle$ и углом между $next\_point$ и $main\_point$ находится в промежутке $(0; angle\_err\_min)$ или больше $angle\_err\_max$, то:

   (a) Вычислить сигнум между $next\_point$ и now_point – $sign\_now$.

   (b) Вычислить расстояние между $prev\_point$ и now_point - $dist$.

   (c) Если расстояние $dist$ превышает ε, то:

      i. Проверить $sign\_now$ на соответствие с $sign\_main$: если $sign\_now$!= $sign\_main$ только по одной из координат x или y, то:

         A. Записать в массив с «упрощенным маршрутом» $prev\_point$, присвоить $main\_point$ и $sign\_main$ значения $prev\_point$ и $sign\_now$ соответственно.

      ii. Иначе записать в массив с «упрощенным маршрутом» now_point, присвоить $main\_point$ и $sign\_main$ значения now_point и $sign\_now$ соответственно.

10. Присвоить $angle$ значение угла между $next\_point$ и $main\_point$, перейти к 6.

11. Внести координаты последней точки в массив с «упрощенным маршрутом».

12. Конец.

4.6 Определение погрешности

Для предоставления данных о погрешности разработанного алгоритма определим, каким образом выявляется эта погрешность.

Точка, входящая в «упрощенный» маршрут, является ошибочно попавшей в него, если:

1. Точка является «выбросом» или «выбросом направления».
2. На рассматриваемом участке дороги точка теоретически являющаяся «главной» была заменена ложной, что подтверждается визуальным представлением «упрощенного» маршрута. Не является «выбросом» или «выбросом направления».
3. Точка является незначимой для данного маршрута, однако была добавлена в «упрощенный» маршрут.

Таким образом, для определения погрешности алгоритма будем использовать четыре переменные, по одной на каждый пункт из представленного выше списка и одна дополнительная переменная для демонстрации совокупности неправильных данных в «упрощенном» маршруте. Подобное разделение при определении погрешности необходимо для более точной оценки дефектов алгоритма и их последующей ликвидации.

Обозначим переменные для вычисления погрешности как $error_i$, где $i \in [1, 4]$. Количество точек, соответствующих одному из пунктов вышеприведенного списка, обозначим как err_count_j, где $j \in [1, 3]$. Общее количество точек в «упрощенном» маршруте обозначим - $simple\_count$.

Численное значение $error_i$ будем вычислять следующим образом:

1. $error_1 = err\_count_1/(simple\_count - (err\_count_2 + err\_count_3)) * 100\%$
2. $error_2 = err\_count_2/(simple\_count - (err\_count_1 + err\_count_3)) * 100\%$

3. $error_3 = err\_count_3/(simple\_count - (err\_count_1 + err\_count_2)) * 100\%$
4. $error_4 = \sum_{i=1}^{3} err\_count_i/simple\_count * 100\%$

После демонстрации числовых характеристик названных переменных также укажем количество точек, которые теоретически должны были являться «главными», однако не попали в «упрощенный» маршрут.

4.7 Вычисление погрешности для алгоритма версии 1

В ходе тестирования алгоритмов на двух основных маршрутах передвижения пользователя была получена следующая погрешность:

Первый маршрут:

1. $error_1 = 33\%$
2. $error_2 = 0\%$
3. $error_3 = 0\%$
4. $error_4 = 33\%$

Количество «главных» точек, не попавших в «упрощенный» маршрут - 4.

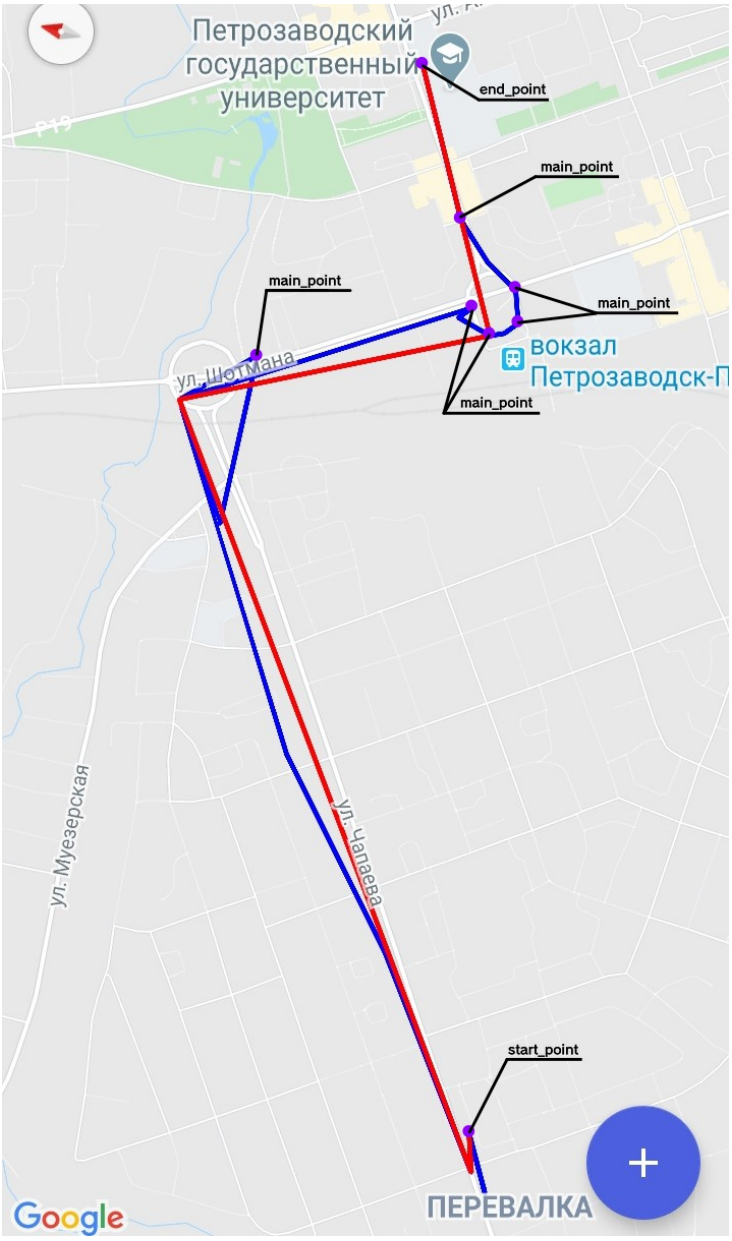


Рис. 15. Погрешность первой версии алгоритма на первом маршруте.

Второй маршрут:

1. $error_1 = 0\%$
2. $error_2 = 60\%$
3. $error_3 = 71\%$
4. $error_4 = 80\%$

Количество «главных» точек, не попавших в «упрощенный» маршрут - 10.

Рис. 16. Погрешность первой версии алгоритма на втором маршруте.

Как видно из приведенных выше данных погрешность алгоритма на обоих маршрутах оказалась достаточно высокой, несмотря на то, что визуальное представление «упрощенного» маршрута в целом соответствует исходным передвижениям пользователя. Для уменьшения степени погрешности было принято решение о переработке алгоритма упрощения. Разработанные улучшения представлены в следующем параграфе.

4.8 Описание алгоритма версии 2

Вторая версия алгоритма представляет собой улучшение логики алгоритма, описанной ранее, поэтому основные операции над данными из входного маршрута остались теми же, однако их последовательность и приоритетность были изменены.

Новая версия алгоритма базируется на идее сравнения данных для двух последовательных сегментов маршрута, вместо фиксирования главной точки и сравнения ее значений с последующими.

В первую очередь происходит определение направления движения на текущем сегменте путем расчета сигнума от разницы между координатами, которые его определяют. Данное значение сравнивается с направлением движения в следующем сегменте, и в случае его изменения по одной координате алгоритм записывает в «упрощенный» маршрут точку, являющуюся началом сегмента, в котором данное изменение было найдено.

Подобная проверка позволяет достаточно просто определять повороты в маршруте, угловое значение которых приближено к 90 градусам, однако данной меры недостаточно в случае работы с дорогами, имеющими округлую форму, или при движении по диагонали. Для определения случаев подобного движения используется проверка значения модуля разности углов между двумя сегментами.

При превышении заданного значения α - радиуса перекрестка, точка, являющаяся началом сегмента, на котором превышение было выявлено, заносится в «упрощенный» маршрут. Вследствие чего точки, характеризующие движение по диагонали либо кольцевой дороге, также попадают в «упрощенный» маршрут, что позволяет более точно передать изначальные передвижения, не жертвуя при этом степенью упрощения. Также данная проверка является опциональной и ее включение в алгоритм зависит от выбора пользователя.

Использование описанных проверок позволяет не допустить попадания в «упрощенный» маршрут «выбросов» и «выбросов направления».

Для отсеивания незначимых точек используется проверка на расстояние, как и в первой версии алгоритма, поэтому повторно описывать ее здесь не будем. Из-за внесения изменений в логику алгоритма его сложность не изменилась.

4.9 Словесное описание алгоритма версии 2

При описании алгоритма будет рассмотрен случай, когда пользователь активировал опциональную проверку.

Обозначения:

$prev\_point$ - координаты $n-1$-ой точки.

now_point - координаты n-ой точки.

$next\_point$ - координаты $n+1$-ой точки.

cur_sign - сигнум направления движения пользователя, вычисляемый между $n-1$-ой и n-ой точками, пересчитывается на каждом шаге алгоритма.

$next\_sign$ - сигнум направления движения пользователя, вычисляемый между $n+1$-ой и n-ой точками, пересчитывается на каждом шаге алгоритма.

$dist$ - расстояние между n-ой и $n+1$-ой точками, пересчитывается на каждом шаге алгоритма.

ε - наибольшее допустимое расстояние между точками, полученное опытным путем (ширина дороги).

cur_angle - угол между $n-1$-ой и n-ой точками, пересчитывается на каждом шаге алгоритма.

$next\_angle$ - угол между n-ой и $n+1$-ой точками, пересчитывается на каждом шаге алгоритма.

$angle$ - модуль разности углов $next\_angle$ и cur_angle.

α - минимально допустимое значение angle (радиуса перекрестка).

Алгоритм:

1. Записать координаты первой точки в массив с «упрощенным» маршрутом.

2. Присвоить переменной $prev\_point$ координаты первой точки.

3. Если количество координат меньше 3, то перейти к 10.

4. Получить координаты n-ой и $n + 1$-ой точек.

5. Определить переменные $cur\_sign = sign(now\_point - prev\_point)$, $next\_sign = sign(next\_point - now\_point)$.

6. Вычислить расстояние $dist = next\_point - now\_point$.

7. Если $cur\_sign \neq next\_sign$ только по одной координате и $dist > \varepsilon$, то записать now_point в массив с «упрощенным» маршрутом.

8. Иначе, если $cur\_sign == next\_sign$ по обеим координатам и $dist > \varepsilon$:

   (a) Определить переменные $cur\_angle = atan2(now\_point - prev\_point)$, $next\_angle = atan2(next\_point - now\_point)$.

   (b) Определить $angle = |next\_angle - cur\_angle|$.

   (c) Если $angle > \alpha$, то записать now_point в массив с «упрощенным» маршрутом.

9. Определить переменную $prev\_point$ = координаты $n - 1$-ой точки. Перейти к 2.

10. Записать координаты последней точки в массив с «упрощенным» маршрутом.

11. Конец.

4.10 Вычисление погрешности для алгоритма версии 2

В ходе тестирования алгоритмов на двух основных маршрутах передвижения пользователя была получена следующая погрешность:

Первый маршрут:

1. $error_1 = 14\%$

2. $error_2 = 0\%$

3. $error_3 = 0\%$

4. $error_4 = 14\%$

Количество «главных» точек, не попавших в «упрощенный» маршрут - 2.

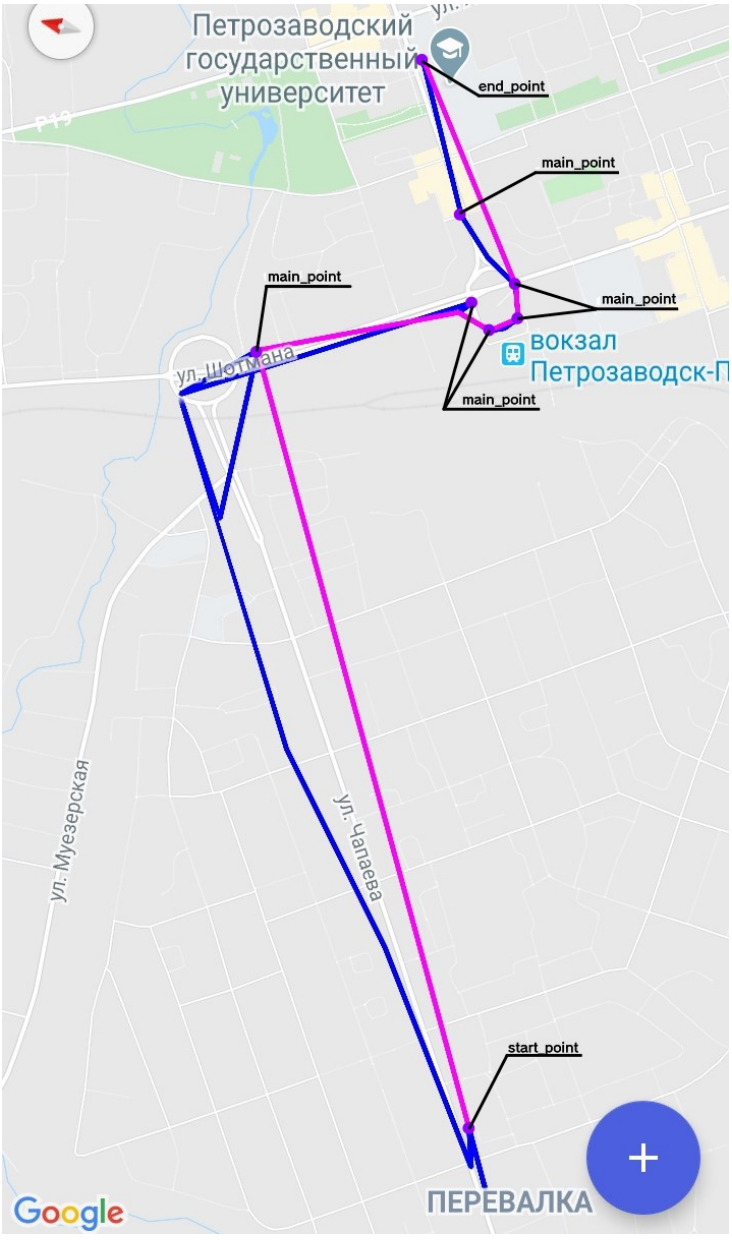


Рис. 17. Погрешность второй версии алгоритма на первом маршруте.

Второй маршрут:

1. $error_1 = 0\%$
2. $error_2 = 0\%$
3. $error_3 = 0\%$
4. $error_4 = 0\%$

Количество «главных» точек, не попавших в «упрощенный» маршрут - 0.

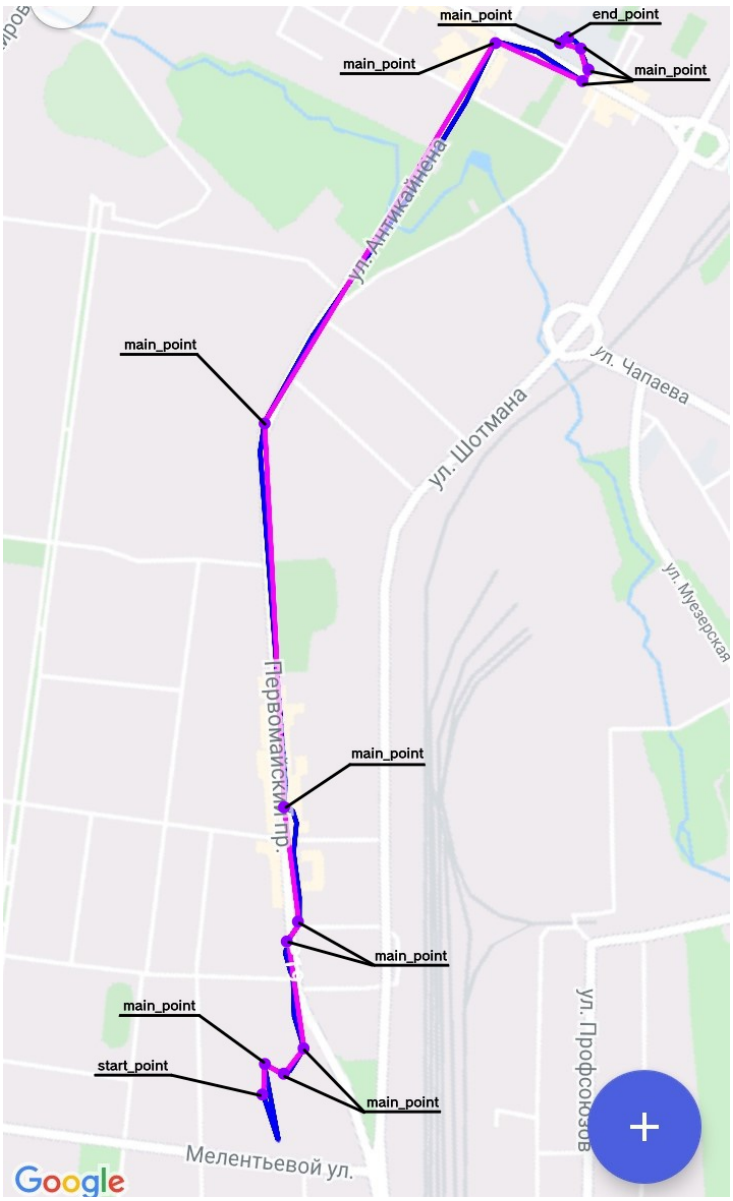


Рис. 18. Погрешность второй версии алгоритма на втором маршруте.

Таким образом, погрешность для второй версии алгоритма имеет значительно меньшие показатели, чем для первой. Общий процент погрешности для первого маршрута уменьшился с 33% до 14%, а количество «главных» точек, не попавших в маршрут, с 4 до 2, что в целом является допустимой погрешностью и не сказывается на идентификации передвижений пользователя. На втором маршруте погрешностей в работе алгоритма обнаружено не было. Вследствие чего мы делаем вывод, что вторая версия алгоритма справилась с поставленной задачей.

Глава 5

Приложение

5.1 Разработка прототипа приложения

В ходе работы над магистерской диссертацией реализовано корректно работающее приложение, учитывающее все проблемы при определении местоположения, описанные в предыдущем параграфе.

Описание возможностей приложения:

Нахождение.

Приложение получает данные о местонахождении пользователя на основе работающего GPS, вышек сотовой связи либо Wi-Fi.

Запись.

При нажатии в окне приложения кнопки «Start Write» выводится диалоговое окно, предоставляющее пользователю возможность выбора между обычной записью изменений его дислокации и записью с отображением маршрута на карте. Вне зависимости от выборанного способа учета изменений местоположения данные заносятся в файл. Каждая последующая запись добавляется при изменении координат согластно условиям, описанным в Google Location Services API, до момента повторного нажатия на кнопку «Start Write». Данные записываются в файл «GPS_Coordinates.txt» с указанием времени записи, координат долготы и широты нахождения пользователя. Также создается файл «GPS_Points.txt» с численными данными о дислокации пользователя.

Чтение.

При нажатии на кнопку «Read File» пользователь открывает файл

«GPS_Coor
dinates.txt», в котором указаны данные, описанные выше. Открытие файла происходит с помощью встроенной программы для чтения текстов.

Демонстрация.

Пользователь имеет возможность моментального получения данных о своей дислокации в виде маркера на карте с помощью нажатия на кнопку «Show Location». При нажатии на кнопку «Start Write» и выборе пункта в контекстном меню о желании увидеть свои передвижения на карте пользователь получает ломаную своих перемещений на экране устройства.

Упрощение.

При нажатии на кнопку «Simplify» пользователь получает файл «GPS_Simple.txt» с упрощенным маршрутом своих передвижений и «упрощенную» ломаную передвижений в приложении, отображенную на карте.

5.2 Тесты.

Ниже представлены изображения, демонстрирующие основные функции приложения.

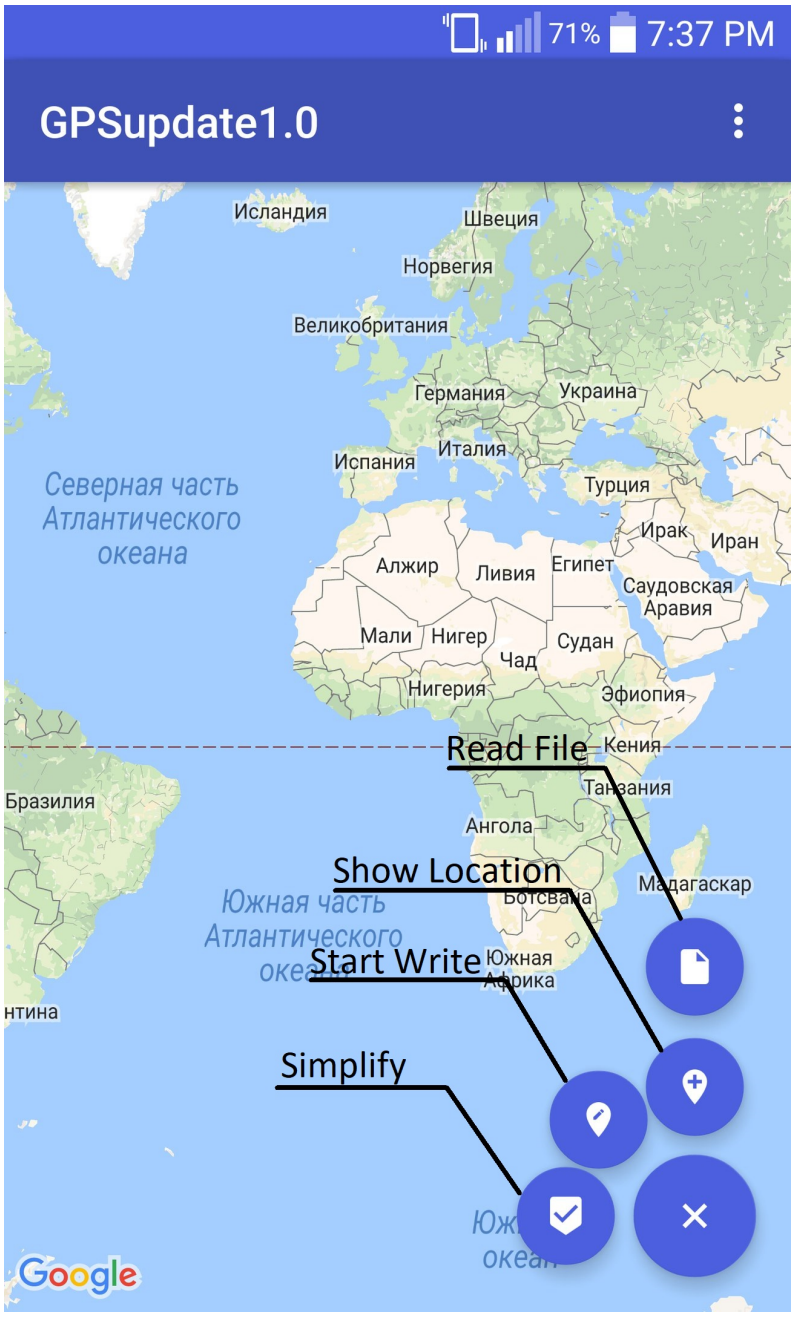


Рис. 19. Главное окно приложения.

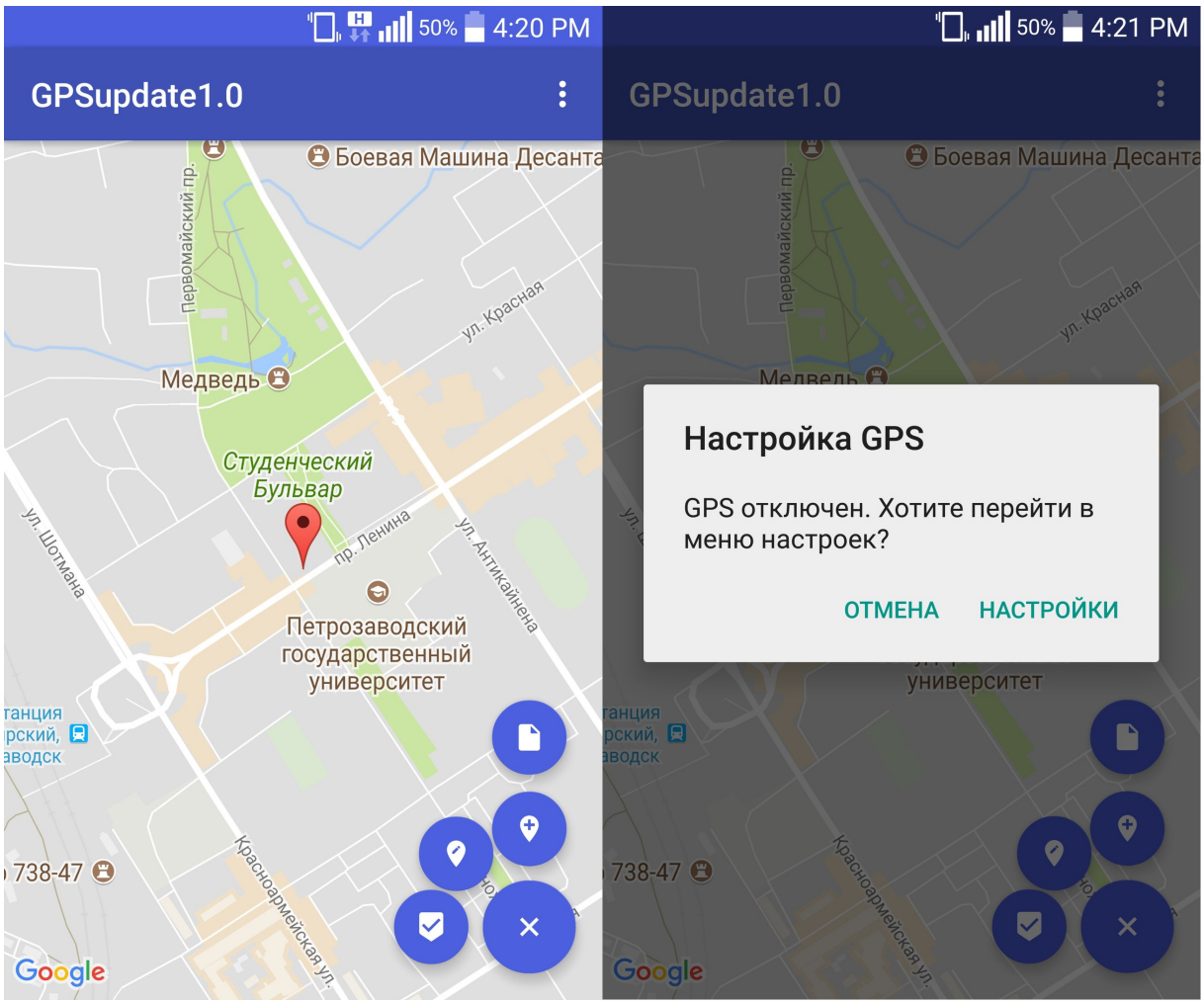


Рис. 20. Нажатие кнопки «Show Location»: отображение дислокации пользователя или вывод диалогового окна для включения функции учета местоположения, если она отключена.

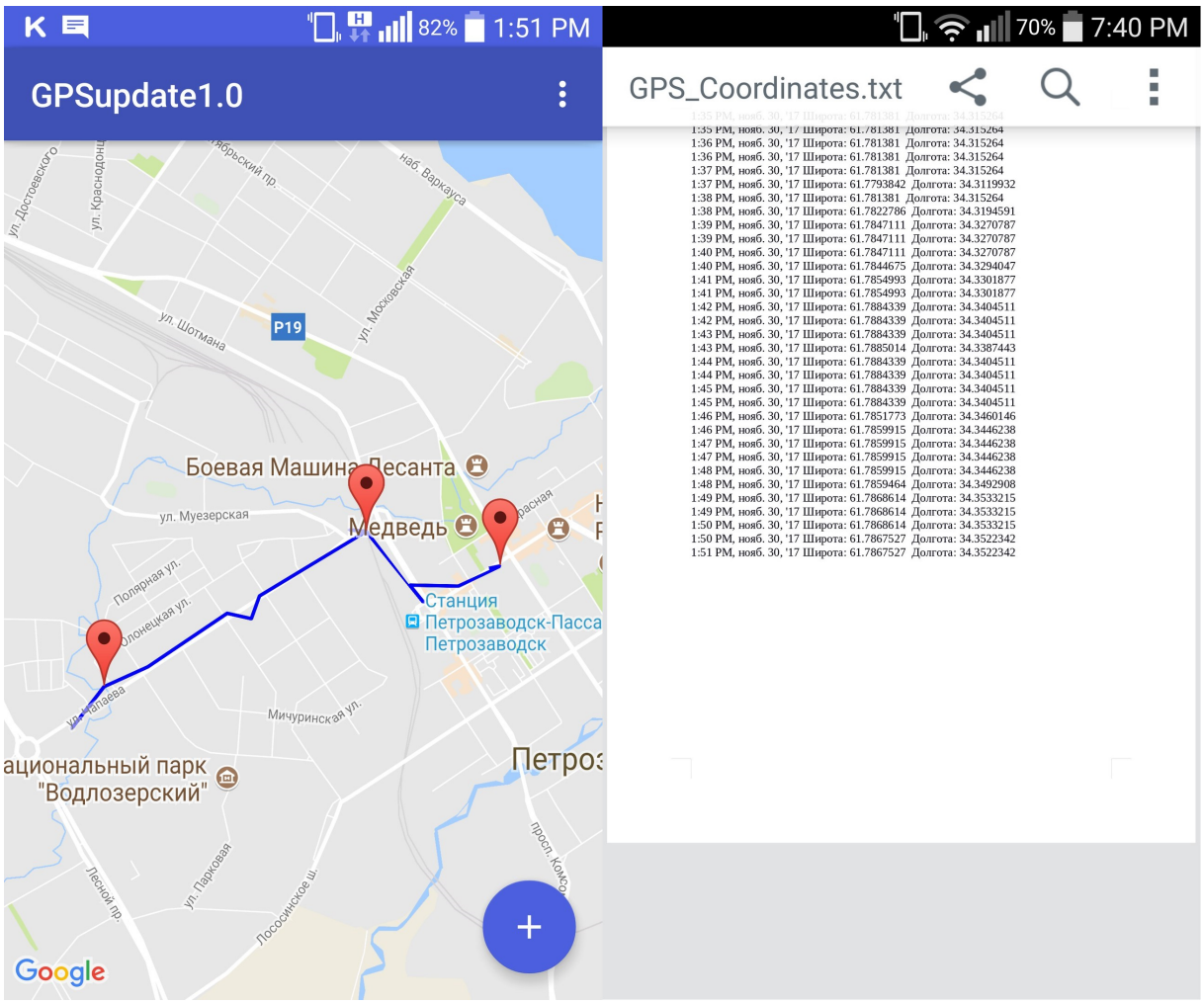


Рис. 21. Кривая, демонстрирующая передвижения пользователя, и файл с аналогичными данными(кнопки «Start Write» и «Read File»).

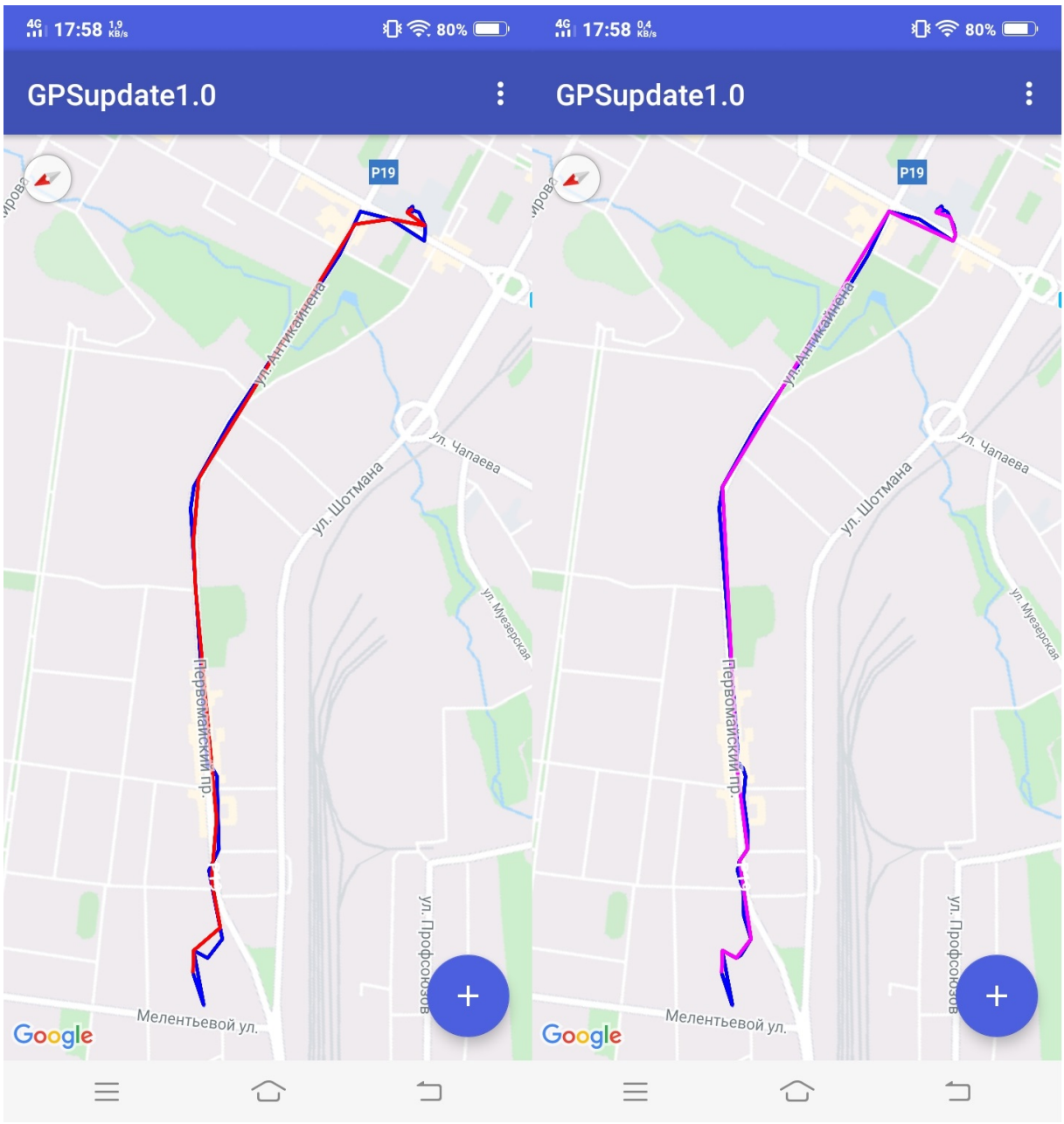


Рис. 22. Исходная полилиния передвижений пользователя и ее упрощенная версия(кнопка «Simplify»). Слева первая версия алгоритма, справа вторая.

Заключение

Настоящая работа посвящена анализу проблем адаптации алгоритмов упрощения ломаных на мобильных устройствах и реализации оригинальной версии алгоритма упрощения.

В работе рассмотрены несколько способов получения данных о дислокации пользователя и общие проблемы при получении информации о местоположении на территории г. Петрозаводска. Представлены описания уже имеющихся алгоритмов упрощения ломаной и описаны две версии собственного алгоритма.

В ходе работы разработан оригинальный алгоритм упрощения ломаной с интеграцией его в мобильное приложение. Получение данных о дислокации пользователя производится с помощью Google Location Services API. Приложение предоставляет возможность демонстрации маршрута передвижений пользователя и результатов работы алгоритма упрощения на карте для возможности визуального представления разницы между исходными и упрощенными данными.

Библиографический список использованной литературы

1. gps.gov [Электронный ресурс]: Global Positioning System. Standart Positioning Service. Performance Standart. 4th Edition. — Электрон. ст. — [USA]. — URL:https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf

2. nstb.tc.faa.gov [Электронный ресурс]: Global Positioning System. Standart Positioning Service. Performance Analysis Report. Report №96. — Электрон. ст. — [USA]. — URL:https://www.nstb.tc.faa.gov/reports/PAN96_0117.pdf

3. ion.org [Электронный ресурс]: Сайт Института Навигации (Institute of Navigation). The World's first GPS MOOC and Worldwide Laboratory using Smartphones. — Электрон. ст. — [USA]. — URL:https://www.ion.org/publications/abstract.cfm?articleID=13079

4. statista.com [Электронный ресурс]: Сайт с информацией об объеме мировых данных. — Электрон. ст. — [Germany]. — URL:https://www.statista.com/statistics/871513/worldwide-data-created/

5. fhwa.dot.gov [Электронный ресурс]: The U.S. Department of Transportation's Federal Highway Administration - Press Release. — Электрон. ст. — [USA]. — URL:https://www.fhwa.dot.gov/pressroom/fhwa1905.cfm

6. fhwa.dot.gov [Электронный ресурс]: The U.S. Department of Transportation's Federal Highway Administration - Highway Statistics. — Электрон. ст. — [USA]. — URL:https://www.fhwa.dot.gov/policyinformation/statistics/2017/hm220.cfm

7. developer.android.com [Электронный ресурс]: Сайт с информацией о API android.location. — Электрон. ст. — [USA]. — URL:https://developer.android.com/reference/android/location/package-summary.html

8. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Класс Location. — Электрон. ст. — [USA]. — URL:https://developer.android.com/reference/android/location/Location

9. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Класс LocationManager. — Электрон. ст. — [USA]. — URL:https://developer.android.com/reference/android/location/LocationManager

10. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Интерфейс LocationListener. — Электрон. ст. — [USA]. — URL:https://developer.android.com/reference/android/location/LocationListener

11. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Интерфейс FusedLocationProviderApi. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi

12. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Интерфейс

GoogleApiClient.ConnectionCallbacks. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.ConnectionCallbacks

13. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Интерфейс GoogleApiClient.OnConnectionFailedListener. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener

14. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Класс LocationRequest. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest

15. developer.android.com [Электронный ресурс]: Сайт с информацией для Android-разработчиков. Geofencing. — Электрон. ст. — [USA]. — URL:https://developer.android.com/training/location/geofencing

16. developers.google.com [Электронный ресурс]: Сайт с информацией о Google Maps API. Класс GoogleMap. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/maps/GoogleMap

17. developers.google.com [Электронный ресурс]: Сайт с информацией о Google Maps API. Класс MapFragment. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/maps/MapFragment

18. developers.google.com [Электронный ресурс]: Сайт с информацией о Google Maps API. Интерфейс OnMapReadyCallback. — Электрон. ст. — [USA]. —

URL:https://developers.google.com/android/reference/com/google/android/gms/maps/OnMapReadyCallback

19. developers.google.com [Электронный ресурс]: Сайт с информацией о Google Maps API. Класс Polyline. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/maps/model/Polyline

20. developers.google.com [Электронный ресурс]: Сайт с информацией о Google Maps API. Класс Marker. — Электрон. ст. — [USA]. — URL:https://developers.google.com/android/reference/com/google/android/gms/maps/model/Marker

21. docs.oracle.com [Электронный ресурс]: Сайт с информацией для Java-разработчиков. — Электрон. ст. — [USA]. — URL:http://docs.oracle.com/javase/7/docs/api/index.html

22. psimpl.sourceforge.net [Электронный ресурс]: Сайт с информацией об алгоритме упрощения полилинии - радиальное расстояние. — Электрон. ст. — [USA]. — URL:http://psimpl.sourceforge.net/radial-distance.html

23. psimpl.sourceforge.net [Электронный ресурс]: Сайт с информацией об алгоритме упрощения полилинии - Реуманна-Виткама. — Электрон. ст. — [USA]. — URL:http://psimpl.sourceforge.net/reumann-witkam.html

24. psimpl.sourceforge.net [Электронный ресурс]: Сайт с информацией об алгоритме упрощения полилинии - Ланга. — Электрон. ст. — [USA]. — URL:http://psimpl.sourceforge.net/lang.html

25. psimpl.sourceforge.net [Электронный ресурс]: Сайт с информацией об алгоритме упрощения полилинии - Дугласа-Пекера. — Электрон. ст. — [USA]. — URL:http://psimpl.sourceforge.net/douglas-peucker.html

26. N. Meratnia and A. Rolf, «Spatiotemporal compression techniques for moving point objects», in Advances in Database Technology—EDBT

2004, vol. 2992 of Lecture Notes in Computer Science, pp. 765–782, Springer, Berlin, Germany, 2004.

27. G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro, «On-line data reduction and the quality of history in moving objects databases», in Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '06), pp. 19–26, ACM, 2006.

28. P. M. Lerin, D. Yamamoto, and N. Takahashi, «Encoding travel traces by using road networks and routing algorithms», in Intelligent Interactive Multimedia: Systems and Services. Springer, 2012, pp. 233–243.

29. G. Kellaris, N. Pelekis, and Y. Theodoridis, «Map-matched trajectory compression», Journal of Systems and Software, vol. 86, no. 6, pp. 1566–1579, 2013.

30. R. Song, W. Sun, B. Zheng, and Y. Zheng, «PRESS: a novel framework of trajectory compression in road networks», Proceedings of the VLDB Endowment, vol. 7, no. 9, pp. 661–672, 2014.

31. J. Muckell, J. H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi, «SQUISH: an online approach for GPS trajectory compression», in Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications, article 13, Washington, DC, USA, May 2011.

32. J. Muckell, P. W. Olsen Jr., J.-H. Hwang, C. T. Lawson, and S. S. Ravi, «Compression of trajectory data: a comprehensive evaluation and new approach», GeoInformatica, vol. 18, no. 3, pp. 435–460, 2014.

33. C. Long, R. C. W. Wong, and H. V. Jagadish, «Direction-preserving trajectory simplification», Proceedings of the VLDB Endowment, vol. 6, no. 10, pp. 949–960, 2013.