1. Аннотация

Целью данного обзора является рассмотрение существующих классических подходов, решающих проблему маршрутизации транспортных средств с некоторыми ограничениями. В дальнейшем планируется использовать полученные результаты для сравнения с результатами, полученными моделью машинного обучения, которая решает ту же самую проблему с теми же самыми ограничениями.

В работе проведены экспериментальные исследования на одинаковых наборах данных, на которых алгоритмы оптимизируют конкретные задачи. Также приведены сравнения времени работы и оптимальности решений всех алгоритмов, описанных в данной работе.

2. Введение

2.1 Задача VRP

Задача маршрутизации транспортных средств (Vehicle Routing Problem) – это класс задач комбинаторной оптимизации, суть которой в построении оптимального пути между депо и клиентами, зачастую с дополнительными ограничениями на время и объем товаров. VRP легко представим в виде графа с N + 1 вершиной, где N вершин это клиенты, одно депо и каждое ребро отвечает за путь, пройденный одним или несколькими транспортными средствами, наша задача – уменьшить некую заданную стоимость этого пути (пройденное расстояние, время или количество транспортных средств и т.д.). В целом данная задача может быть сформулирована следующим образом:

«Каков оптимальный набор маршрутов для транспортных средств, чтобы доставить продукт определенной группе клиентов с минимальными затратами?»

В такой постановке можно заметить, что задача является модификацией задачи Коммивояжера, а значит является NP-трудной, что накладывает свои ограничения.

2.2 Актуальность

Проблемы маршрутизации транспортных средств, хотя и являются общим и слабо определенным классом проблем, имеют большое практическое значение. Например, распределение заказов такси, все возможные логистические задачи распределения товаров, курьерская доставка и еще множество других. Сейчас существует множество открытых и коммерческих полностью готовых инструментов, способных решать задачу VRP, и при этом демонстрировать хорошие результаты.

Однако основная проблема начинается, когда размер задач становится большим.

Большие размерности

Под большими размерностями задачи подразумевается большое количество клиентов. Прежде доставка происходила только с помощью грузовиков, однако сейчас это могут быть различные курьерские службы, беспилотные транспортные средства, роботы-доставщики и т.п. Особенно в условиях пандемии, когда большинство заказов делается из дома, размерность задач только растет.

Готовые алгоритмы, которые решают данные задачи, требуют слишком много времени для получения решения. И стоит отметить, что чаще всего задача построения маршрутов является динамической и стохастической:

Динамическая постановка VRP (Dynamic VRP)

Данная проблема является одним из важных вариантов задачи VRP. Его цель состоит в построении оптимальных маршрутов для транспортных средств, необходимых для обслуживания определенного набора клиентов, пока поступают новые заказы от клиентов во время выполнения доставки заказов. Таким образом, маршруты должны быть перестроены динамически. То есть в течение доставки заказов маршрут может меняться, например:

- отменяются или добавляются заказы;
- существуют ограничения по времени на доставку заказов;
- во время доставки могут возникать пробки, т.е. например загруженность в часпик одна, а в повседневное время другая.

Стохастическая постановка VRP (Stochastic VRP)

Цель данной проблемы состоит в построении оптимальных наборов маршрутов для транспортных средств, необходимых для обслуживания определенного набора клиентов, но один или несколько параметров рассматриваются как случайные или неизвестные в течение планирования маршрута. Случайными параметрами могут быть

- количество клиентов;
- характер потребительского спроса в данном местоположении;
- время, такое как время обслуживания и время в пути;
- возникновение пробок, например загруженность отдельных магистралей с течением времени может изменяться.

В современных задачах различных ограничений достаточно много. Например, одни из самых распространенных это:

Временные окна(VRP with Time Windows)

Этим занимается задача “Проблема маршрутизации транспортных с временными окнами”.

Транспортное средство затрачивает время на перемещение и на обслуживание клиентов. Каждому клиенту сопоставляется некий промежуток времени состоящий из двух моментов - это время наиболее ранней допустимой доставки и время наиболее поздней. Таким образом, задача заключается в построении оптимального маршрута для транспортных средств, чтобы посетить как можно большее число клиентов, не нарушая, выделенные временные рамки для каждого клиента.

Удовлетворение спроса (Capacity VRP)

Этим занимается задача “*Проблема маршрутизации транспортных средств с ограниченной вместимостью”*.

Формально спрос представляет собой некоторую неотрицательную величину, характеризующую объем товара запрашиваемого клиентом. Для удовлетворения спроса необходима перевозка заданного объема товара из места хранения к клиенту транспортным средством, имеющим определенную вместимость. Таким образом, на маршрут накладывается следующее ограничение: вдоль пути от депо и обратно возможно посетить лишь то число клиентов, суммарный спрос которых не превышает вместимость транспортного средства.

3 Постановка задачи

Задачей данного обзора является исследование существующих алгоритмов для решения проблемы маршрутизации транспортных средств.

На основе наиболее перспективных подходов требуется разработать Framework в виде устанавливаемого пакета setup.py. В данном пакете должны содержаться все реализации алгоритмов рассмотренные в данном обзоре. Работа с пакетом осуществляется следующим образом:

- пользователь выбирает один или несколько подходов, а также выбирает задачу, которую необходимо решить;
- пользователь подает свой набор набор данных;
- в качестве результата выводится длина длина оптимизированного маршрута и время затраченное на оптимизацию.

Данный Framework в дальнейшем будет улучшаться и в конечном итоге он должен выдавать решение точнее и быстрее , чем это делает OR-Tools от компании Google.

4 Обзор существующих решений

4.1 Цели обзора

Целями данного обзора являются:

1. изучение алгоритмов «*Simulated Annealing»*, «*Branch-and-Cut»*, «*LKH»*.

2. сравнение алгоритмов по оптимальности решения и времени работы при решении задач *CVRP*, *CVRPTW*;
3. поиск открытых наборов данных для проведения собственных исследований;
4. формулировка направлений дальнейших исследований.

4.2 Виды VRP алгоритмов

Эвристические алгоритмы являются наиболее распространенными алгоритмами для таких задач и могут быть грубо классифицированы как *конструктивные эвристики* и *метаэвристики*.

Конструктивные эвристические алгоритмы - это алгоритмы, которые строят действительное решение VRP, обычно во времени, полиномиальном по размеру входных данных и без каких-либо гарантий с точки зрения качества решения, т.е. алгоритмы, которые шаг за шагом выстраивают решение, учитывая общую стоимость, которая получается в ходе решения.

Наиболее распространенным подходом для области VRP являются **метаэвристики —** это общие эвристики, позволяющие находить близкие к оптимальным решениям, решения различных задач оптимизации за приемлемое время.

Следующий вид алгоритмов это **MIP(Mixed-Integer Programming) алгоритмы**, т.е. алгоритмы смешанного целочисленного программирования. Данные алгоритмы являются точными методами.

MIP алгоритмы предлагают вычислить все возможные решения, пока не будет достигнут лучший из них.

Отличие MIP алгоритмов от метаэвристик в том, что точные методы найдут оптимальное решение за конечное время, хотя часто это время может быть очень большим. Поэтому были разработаны метаэвристики специально для того, чтобы найти решение за меньшее время и, которое приближено к оптимальному. Специально для сложных проблем, метаэвристики часто могут предложить лучший компромисс между качеством решения и вычислительным временем. Более того, метаэвристики более гибкие, чем точные методы.

Поскольку метаэвристические рамки определены в общих чертах, метаэвристические алгоритмы могут быть адаптированы для удовлетворения потребностей большинства задач оптимизации в реальном времени с точки зрения ожидаемого качества решения и допустимого вычислительного времени, которое может сильно различаться в разных задачах и в разных ситуациях.

И третий вид алгоритмов относится к **Reinforcement Learning**(обучение с подкреплением)

Обучение с подкреплением - это обучение модели машинного обучения для принятия последовательности решений. Модель обучается тому, что делать и как сопоставлять ситуации с действиями, чтобы максимизировать вознаграждения. Модели не говорят, какие действия совершать, но вместо этого она должна выяснить, какие действия приносят наибольшую награду, попробовав их. В наиболее интересных и сложных случаях действия могут повлиять не только на немедленную награду, но и на следующую ситуацию, а через нее и на все последующие награды. Эти две характеристики — поиск методом проб и ошибок и отсроченное вознаграждение - являются двумя наиболее важными отличительными особенностями обучения с подкреплением. Используя возможности поиска и многочисленных испытаний, обучение с подкреплением в настоящее время является наиболее эффективным способом.

Далее будут описаны алгоритмы, которые использовались в данном обзоре. В течение описания неоднократно будут использоваться слова «подтур», «подмаршрут» и «подграф». Все три понятия являются синонимами.

4.3 Алгоритм «Имитации отжига»

Имитация отжига интерпретируется как медленное уменьшение вероятности принятия худших решений по мере исследования пространства решений. Принятие худших решений позволяет проводить более широкий поиск глобального оптимального решения.

В общем случае алгоритмы имитационного отжига работают следующим образом:

Температура постепенно снижается от начального положительного значения до нуля. На каждом временном шаге алгоритм случайным образом выбирает решение, близкое к текущему, измеряет его качество и переходит к нему в соответствии с зависящими от температуры вероятностями выбора лучших или худших

Опишем один шаг алгоритма «имитации отжига» для оптимизации конкретного подмаршрута:

1. Выбирается некоторое начальное решение S, т.е. это маршрут полученный путем случайным перемешиванием городов.
2. Задается параматер T, который называется *начальной температурой* и некоторый коэффициент $r \in (0, 1)$, который требуется для уменьшения начальной температуры. Также задается *минимальная температура* T_{min} это температура, до которой опускается температура T.
3. Сравниваем начальную темпераутру и минимальную: $T \geq T_{min}$.

   Если данное условие не выполняется, то алгоритм закачивает свою работу и происходит переход к 1 шагу для оптимизации следующего подмаршрута, иначе переходим к шагу 4.
4. Происходит следующая модификация маршрута. В маршруте случайным образом выбирается 2 города, между которыми города инвертируются за исключением этих двух городов. Тем самым получаем новый маршрут S'.
5. Считаем разность расстояний старого маршрута и модифицированного: $\Delta = dist(S) - dist(S')$.
6. Если $\Delta \leq 0$, тогда старый маршрут заменяется на новый: $S := S'$, и далее происходит переход к шагу 8.
7. Если $\Delta > 0$, тогда высчитывается вероятность по формуле: $e^{-\frac{\Delta}{T}}$, и случайным образом выбирается число $p \in (0; 1)$. Если $e^{-\frac{\Delta}{T}} < p$, то $S := S'$, в противном случае ничего не происходит. Далее осуществляется переход к шагу 8.

8. Происходит обновление начальной температуры: $T := T \cdot r$ и далее осуществляется переход к шагу 3.

Преимущества и недостатки метода

Из преимуществ можно отметить следующее:

1. Относительно высокая скорость поиска решения
2. Метод достаточно прост в реализаци

Из недостатков отметим следующее:

1. Алгоритм находит локальное оптимальное решение, которое не обязательно будет являться глобальным.
2. В алгоритме присутствует несколько мест, где используется «random», что влияет на итоговое решение. Например,
    a. случайным образом перемешивается начальный маршрут;
    b. случайным образом выбираются 2 города и города, которые между ними инвертируются;
    c. случайным образом выбирается вероятность принятия/непринятия неоптимального решения.

Данный алгоритм имеет много весомых недостатков и не является точным. Поэтому после изучения и реализации алгоритма «Имитация отжига» возникла потребность рассмотреть точный алгоритм, с которым можно было бы сравнивать результаты. Для экспериментов был использован коммерческий проект Gurobi, о котором известно, что он базируется на целочисленном программировании и в нем используются методы, аналогичные методам «ветвям и границам», поэтому следующим шагом работы было изучение данного алгоритма.

4.4 Алгоритм «Branch & cut»

«Branch & cut» является вариантом известного алгоритма ветвей и границ Лэнда и Дойга [1960 г.], наиболее широко используемого алгоритма для решения многих видов невыпуклых задач оптимизации.

В данной работе мы рассматриваем решение общей оптимизационной задачи.

Данный метод относится к методам решения задач дискретного программирования.

Метод ветвей и границ относится к группе комбинаторных методов и позволяет существенно уменьшить объем перебора вариантов.

В основу метода положены следующие построения:

1. ***Вычисление нижней оценки*** $\xi(G)$ для значения целевой функции на конечном множестве G. Для любого множества G нужно уметь вычислять величину
$$\xi(G) \leq \varphi(x),\ x \in G$$
2. ***Ветвление (разбиение на подмножества)***. Реализация метода связана с постоянным ветвлением множества допустимых решений на дерево подмножеств. Процесс ветвления состоит из некоторых итераций:

ИТЕРАЦИЯ 0.

Полагаем $G^{(0)} = G$. Далее некоторым способом разбиваем исходное множество на систему подмножеств:

$$G^{(0)} = G_1^{(1)} \cup G_2^{(1)} \cup \ \dots \ \cup G_p^{(1)},\ \bigcap_{i=1}^{p} G_i^{(1)} = \varnothing.$$

ИТЕРАЦИЯ k. ($k \geq 1$).

Пусть $G^{(k)}_1$, $G^{(k)}_2$, …,$G^{(k)}_r$ - это множества, еще не подвергшиеся ветвлению. По некоторому правилу выбираем множество $G^{(k)}_{v(k)}$, и это множество разбивается на подмножества:

$$G_{v(k)}^{(k)} = G_{v(k),1}^{(k)} \cup G_{v(k),2}^{(k)} \cup \ \dots \ \cup G_{v(k),s}^{(k)},\ \bigcap_{i=1}^{s} G_{v(k),i}^{(k)} = \varnothing.$$

Дерево подмножеств (процесс ветвления)

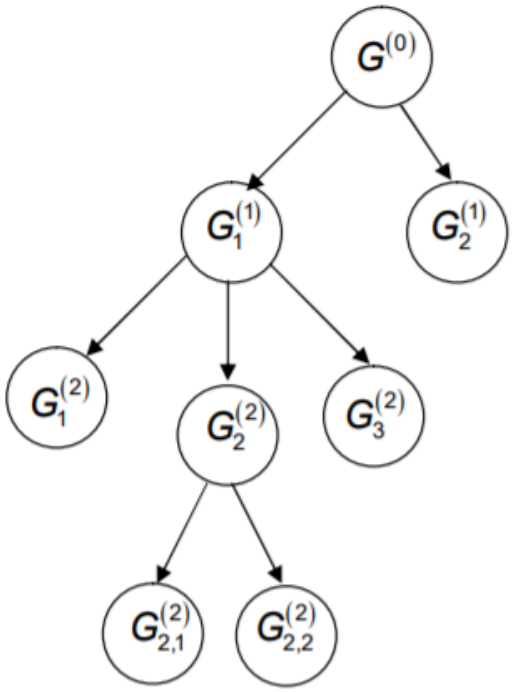


Рис.1

3. ***Границы.*** В процессе ветвления появляются множества, которые не могут быть подвергнуты дальнейшему ветвлению (граничные). Эти множества называются ***концевыми вершинами дерева подмножеств*** (см Рис. 1).
4. ***Нахождение решения***. Для граничных множеств нужно уметь находить допустимые решения, способ зависит от специфики рассматриваемой задачи.
5. ***Пересчет оценок***.
   Если $G_1 \subset G_2$, то $min_{x \in G_1} \varphi(x) \geq min_{x \in G_2} \varphi(x)$. Поэтому оценки не убывают:
   $$\xi(G^i) \geq \xi(G^0)$$
6. ***Признак оптимальности.*** Пусть $G = \bigcup_{i=1}^{s} G_i$, и некоторое решение $x_0 \in G_v$. Если при этом $\varphi(x_0) = \xi(G_v) \leq \xi(G_i)$, G_i- множества, не подвергавшиеся ветвлению, то текущее решение x_0- оптимально.

7. ***Оценка точности приближенного решения***. Пусть $G = \bigcup_{i=1}^{s} G_i$, $\xi = min_i \xi(G_i)$. Пусть x - некоторое решение, тогда $\xi \leq min_x \varphi(x)$ и $\Delta = \varphi(x) - \xi$ - точность.

Изучив, данный алгоритм была подготовлена его реализация с использованием коммерческого проекта Gurobi. Следующим шагом возникла необходимость изучить и реализовать один из самых лучших и точных метаэвристических алгоритмов, а именно эвристика Лина-Кёрнигана.

4.4 Алгоритм «Lin-Kernighan heuristic»

Алгоритм Лина-Кернигана относится к классу так называемых алгоритмов локальной оптимизации. Алгоритм задается в терминах *opt* (обмена или ходы), которые могут преобразовать один тур в другой. При наличии осуществимого тура алгоритм неоднократно выполняет обмены, которые сокращают продолжительность текущего тура, пока не будет достигнут тур, для которого никакой обмен не приведет к улучшению. Этот процесс может повторяться много раз с начальных туров, сгенерированных каким-либо рандомизированным способом. Алгоритм описан ниже более подробно.

В алгоритме предполагается, что некоторое начальное разбиение графа на подграфы уже существует, затем имеющееся приближение улучшается в течение некоторого количества итераций. Применяемый способ улучшения состоит в обмене вершинами в каждом подграфе в отдельности.

В описании общей схемы алгоритма будет использовано выражение «оптимизация подграфа», поэтому стоит заранее прояснить, что подразумевается под «*оптимальностью*» подграфа:

Тур считается λ–*оптимальным (или просто оптимальным)*, если невозможно получить более короткий тур, заменив любое его λ рёбер любым другим набором λ рёбер.

Из этого определения следует, что любой λ - оптимальный тур также является λ'-оптимальным для $1 \leq \lambda' \leq \lambda$. Также легко видеть, что тур, содержащий n городов, оптимален, если и только если этот тур n-оптимален.

В общем, чем больше значение λ, тем больше вероятность того, что последний тур оптимален. При достаточно больших λ кажется, по крайней мере интуитивно, что λ - оптимальный тур должен быть оптимальным.

К сожалению, количество операций по проверке всех λ - обменов быстро увеличивается по мере увеличения количества городов. В наивной реализации тестирование λ-opt имеет временную сложность $O(n^{\lambda})$. Кроме того, нет нетривиальной оценки сверху количества λ - обменов.

Однако **недостатком** является то, что λ нужно указывать заранее. Это трудно узнать, какой λ нужно использовать для достижения наилучшего компромисса между временем работы и качеством решения.

Лин и Керниган устранили этот недостаток, введя мощную *переменную λ-opt* алгоритм. Алгоритм изменяет значение λ во время своего выполнения, решая на каждой итерации, каким должно быть значение λ. На каждой итерации алгоритм проверяет, для возрастающих значений λ, приведет ли перестановка λ рёбер к более короткому туру. Учитывая, что рассматривается обмен r рёбрами, проводится серия тестов, чтобы определить, следует ли рассматривать обмен r+1 рёбрами. Это продолжается пока не будут выполнены некоторые условия остановки.

На каждом шаге алгоритм учитывает растущий набор потенциальных обменов (начиная с r = 2). Эти обмены выбраны таким образом, чтобы возможный тур мог быть сформирован на любом этапе процесса. Если в ходе исследования удастся найти новый более короткий тур, то фактический тур заменяется новым туром.

Опишем общую схему одной итерации **алгоритма Кернигана-Лина** для оптимизации конкретного подмаршрута:

1. **Формирование множества пар вершин для перестановки**.

Из вершин, которые ещё не были переставлены на данной итерации, формируются все возможные пары (в парах должно присутствовать по одной вершине из каждой части имеющегося подграфа).

2. **Построение новых вариантов разбиения графа**.

   Каждая пара, подготовленная на шаге 1, поочерёдно используется для обмена вершин между частями имеющегося подграфа для получения множества новых вариантов деления.

3. **Выбор лучшего варианта разбиения графа**.

   Для сформированного на шаге 2 множества новых делений подграфа выбирается лучший вариант. Этот вариант далее фиксируется как новое состояние подграфа, а соответствующая выбранному варианту пара вершин отмечается как использованная на текущей итерации алгоритма.

4. **Проверка использования всех вершин**.

   При наличии в подграфе вершин, ещё не использованных при перестановках, выполнение итерации алгоритма снова продолжается с шага 1. Если же перебор вершин графа завершён, далее следует шаг 5.

5. **Выбор наилучшего варианта подграфа.**

   Среди всех подграфов, полученных на шаге 3 проведённых итераций, выбирается (и фиксируется) наилучший вариант подграфа. Если дальнейшее улучшение подграфа не происходит, то алгоритм переходит к оптимизации следующего подграфа, переходя к шагу 1.

Дополнительно стоит пояснить, что на шаге 2 итерации алгоритма перестановка вершин каждой очередной пары осуществляется для одного и того же подграфа, выбранного до начала выполнения итерации или определённого на шаге 3. Общее

количество выполняемых итераций фиксируется заранее и является параметром алгоритма (за исключением случая остановки при отсутствии улучшения разбиения на очередной итерации).

Для реализации алгоритма LKH я использовал $\lambda = 2, 3$, так как являются наиболее часто используемыми.

4.5 Эксперименты

Эксперименты проводились на наборах данных города Алматы. Всего составлено три набора данных.

Каждая задача в первом наборе включает в себя 20 клиентов и депо, каждая задача во втором 50 клиентов и депо, и в третьем - 100 клиентов и депо.

Файл с данным соедржит в себе:

- первый и второй столбец это широта и долгота клиенов и депо;
- третий столбец состоит и весов грузов необходимых для доставки. Данный параметр у депо = 0.
- Если решается задача CVRPTW то добавляется еще 3 столбца (время в каждом столбце выражено в минутах):
    - первый столбец содержит левую границу временного окна, раньше которого груз не может быть доставлен клиенту. Данный параметр у депо = 0;
    - второй столбец содержит правую границу временного окна, позже которого груз не может быть доставлен клиенту. Данный параметр у депо = 0;
    - третий столбец содержит время обслуживания клиента. Данный параметр у депо = 0.

На решение каждой задачи в каждом наборе данных отводилось по 200 секунд

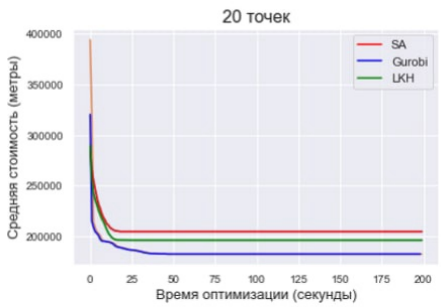
Перед тем, как сравнивать метаэвристические алгоритмы с коммерческим проектом Gurobi необходимо было узнать какой стоит использовать алгоритм для сравнений 2-opt или 3-opt. Поэтому были проведены эксперименты алгоритмов 2-opt и 3-opt для решения задач CVRP и CVRPTW.

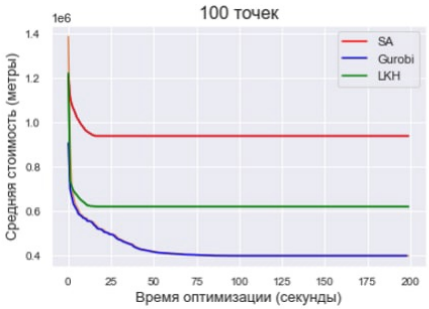
Графики

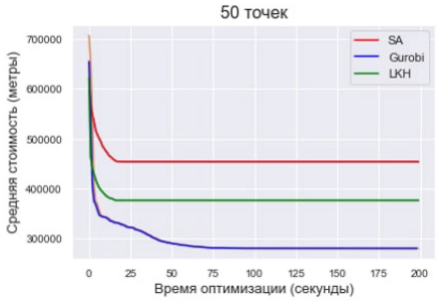
Стоит отметить, что оба алгоритма в конечном счете сходятся к одинаковому решению, поэтому важную роль здесь играет время оптимизации. По данным графикам можно сделать вывод, что алгоритм 3-opt находит оптимальное решение быстрее, чем 2-opt. Поэтому для дальнейших сравнений в качестве алгоритма LKH будет выступать алгоритм 3-opt.

Следующий этап экспериментов заключался в сравнении “SA”, “LKH” и коммерческого проекта “Gurobi” для задачи CVRP. Важно отметить, что вместимость транспортного средстдва для

- 20 городов равна 30;
- 50 городов равна 40;
- 100 городов равна 50







По полученным результатам можно сделать выводы:

- Коммерческий проект “Gurobi” в основе которого находится метод Ветвей и границ превосходит метаэвристические алгоритмы “LKH” и “SA”. Данный вывод явно можно сделать для случаев 50 и 100.
- Алгоритм “SA” не подходит для решения реальных задач. На графике для 100 городов явно видно, что “SA” нужно в несколько раз больше времени, чтобы оптимизировать маршрут хотя до такого же состояния, как это сделал алгоритм “LKH”.

- Метаэвристические алгоритмы хоть и демонстрируют приближенные решения, но им требуется гораздо меньше времени для оптимизации, чем коммерческому проекту. По графикам видно, что в среднем на оптимизацию требуется небольше 25 секунд, далее маршрут оптимизируется,но требуется больше времени.
-