1.Аннотация

Целью данного обзора является рассмотрение существующих классических подходов, решающих проблему маршрутизации транспортных средств с некоторыми ограничениями. В будущем планируется разработать собственный алгоритм обучения с подкреплением в задаче построения оптимизированных маршрутов на реальных географических данных с ограничениями, а полученные результаты сравнить с результатами из данного обзора.

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

Основной задачей данного исследования является разработка собственного алгоритма обучения с подкреплением в задаче построения оптимизированных маршрутов на реальных географических данных с ограничениями на:

- вместимость транспортных средств;
- время доставки и обслуживании клиентов.

На данный момент решалась второстепенная задача, результаты которой будут в будущем использованы при решении основной задачи.

Данная задача заключалась в следующем:

- Изучение точных и эвристических подходов для решения задачи.
- Реализация наиболее популярных алгоритмов.
- Сравнительный анализ на реальных географических данных.

4.Обзор существующих решений

4.1 Цели обзора

В рамках данного обзора для достижения необходимых результатов предполагается решение следующих подзадач:

1. Обзор существующих видов алгоритмов задачи VRP.
2. Обзор примеров классических эвристических алгоритмов «*Simulated Annealing*» и «*LKH»,* и точного алгоритма «*Branch-and-Cut*».
3. Подготовка реализаций алгоритмов по оптимальности решения и времени работы при решении задач *CVRP*, *CVRPTW*.
4. Проведение экспериментов на открытых наборах данных.
5. Формулировка направлений дальнейших исследований.

4.2 Виды VRP алгоритмов

Обзор VRP алгоритмов начнем с эвристических 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 алгоритмы смешанного целочисленного программирования **(Mixed-Integer Programming algorithms).** Данная задача заключается в том,что в ней некоторые переменные решения ограничены целыми значениями (т.е. целыми числами, такими как -1, 0, 1, 2, и т.д.) в оптимальном решении. MIP алгоритмы предлагают вычислить все возможные решения, пока не будет достигнут лучший из них.

Однако целочисленные переменные делают задачу оптимизации невыпуклой, а значит и более трудной для решения. Объем памяти и время решения могут увеличиваться экспоненциально по мере добавления целочисленных переменных. 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.

Последний 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 параметр 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. Сравниваются начальная и минимальная температуры: $T \geq T_{min}$. Если данное условие не выполняется, то алгоритм заканчивает свою работу и происходит переход к 1 шагу для оптимизации следующего подмаршрута, иначе осуществляется переход 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 получается новый маршрут S'.
5. Вычисляется 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 вычисляется 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 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 реализации.

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 была использована коммерческая реализация алгоритма решения задачи целочисленного программирования 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 метода.

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.

Метод ветвей и границ относится к группе комбинаторных методов и позволяет существенно уменьшить объем перебора вариантов. Рассмотрим постановку Данцига-Фалкерсона-Джонсона:

1. $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \rightarrow min_{x}, x_{ij} \in \{0,1\},$
2. $\sum_{i=1}^{n} x_{ij} = 1, \forall j,$

3. $\sum_{j=1}^{n} x_{ij} = 1, \forall i,$

4. $\sum_{i \in Q} \sum_{j \in Q, j \neq i} x_{ij} \leq |Q| - 1, |Q| \geq 2,$

где x_{ij} принимает значения 0 или 1 в зависимости от того, входит ли ребро из вершины i в вершину j, c_{ij} - расстояние между вершинами i и j. Условия 2,3 гарантируют, что из каждой вершины выходит и входит ровно одно ребро. Условие 4 гарантирует, что полученное решение является единым циклом, проходящем через все вершины, а не объединением непересекающихся циклов.

Основная идея решения данной задачи с помощью метода “Ветвей и границ” заключается в том, чтобы ослабить ограничения на переменные в постановке задачи целочисленного программирования до линейных, затем решить задачу в новой постановке и если решение получилось целочисленное, то получить ответ. Если же решение не является целочисленным,то можно выбрать одну из переменных и решить относительно нее две постановки - изначальные ослабленные условия, а также равенство нулю в одной постановке и единице в другой. Иной способ приблизить к целочисленному решению это отделить полученное вещественное решение гиперплоскостью от возможных целочисленных, т.е. создав так называемую “границу”. Итеративное решение с использованием указанных приемов составляет метод “ветвей и границ”, который позволяет получать точные решения.

Изучив, данный метод 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.

Алгоритм Лина-Кернигана относится к классу так называемых алгоритмов локальной оптимизации [17, 18]. Алгоритм указан в терминах обмена (или ходы), которые могут преобразовать один тур в другой. Учитывая возможный тур, алгоритм неоднократно выполняет обмены, которые сокращают продолжительность текущего тура, пока не будет достигнут тур, для которого обмен не приведет к улучшению. Этот процесс может повторяться много раз из начальных туров, сгенерированных каким-либо рандомизированным способом. Алгоритм более подробно описан ниже.

Пусть T будет текущим туром. На каждой итерации алгоритм пытается найти два набора звеньев, $X = \{x_1, \dots, x_r\}$ и $Y = \{y_1, \dots, y_r\}$, таким образом, что, если звенья X будут удалены из T и заменены звеньями Y, то результатом будет лучший тур. Этот обмен ссылками называется движением *r-opt*. Рисунок 3.1 иллюстрирует движение 3-opt.

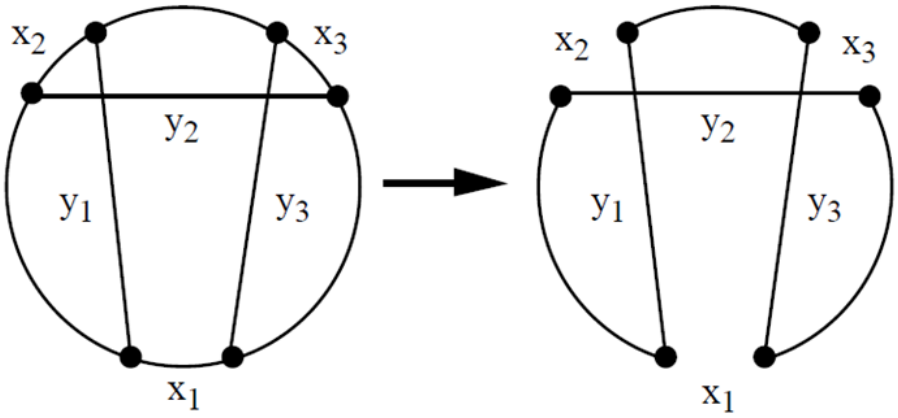


Figure 3.1 A 3-opt move

Два набора X и Y строятся поэлементно. Изначально X и Y пусты. На шаге i пара звеньев, x_i и y_i, добавляются в X и Y соответственно.

Для достижения достаточно эффективного алгоритма в X и Y могут входить только звенья, соответствующие следующим критериям:

(1) Критерий последовательного обмена

x_i и y_i должны иметь общую конечную точку, как и y_i и x_{i+1}. Если t_1 обозначает один из двух концов x_1, то в общем случае имеем: $x_i = (t_{2i-1}, t_{2i})$, $y_i = (t_{2i}, t_{2i+1})$ и $x_{i+1} = (t_{2i+1}, t_{2i+2})$ для $i \geq 1$. См. Рисунок 3.2.

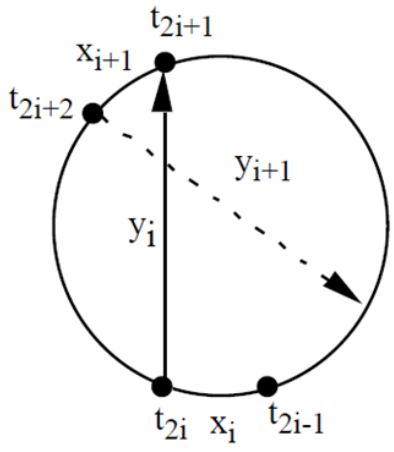


Figure 3.2. Restricting the choice of x_i, y_i, x_{i+1}, and y_{i+1}.

Как видно, последовательность $(x_1, y_1, x_2, y_2, \ \dots\ , x_r, y_r)$ представляет собой цепочку смежных звеньев.

Необходимым (но не достаточным) условием того, что обмен звеньями X с звеньями Y приведет к туру, в котором цепочка замкнута, т. е. $y_r = (t_{2r}, t_1)$. Такой обмен называется последовательным (***sequential***).

Как правило, улучшение тура может быть достигнуто путем последовательного обмена соответствующей нумерацией затронутых звеньев. Однако это не всегда так. На рис. 3.3 показан пример, в котором последовательный обмен невозможен.

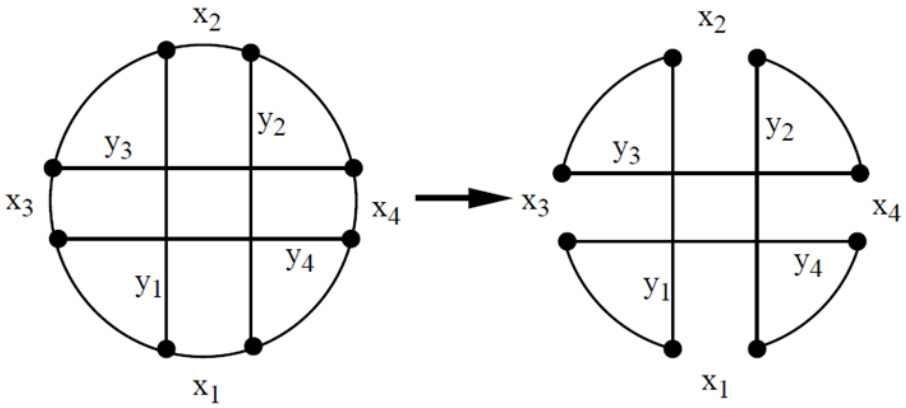


Figure 3.3 Nonsequential exchange (r = 4).

(2) Критерий осуществимости

Требуется, чтобы $x_i = (t_{2i-1}, t_{2i})$ всегда выбирался так, чтобы, если t_2i соединяется с t_1, то итоговая конфигурация - это тур. Этот критерий осуществимости используется для i ≥ 3 и гарантирует, что можно вплотную приблизиться к туру. Этот критерий был включен в алгоритм как для сокращения времени выполнения, так и для упрощения реализации.

(3) Критерий положительного усиления

Требуется, чтобы y_i всегда выбирался так, чтобы коэффициент усиления(gain), G_i, из предложенного набора обменов являлся

положительным. Предположим, что g_i = c(x_i) - c(y_i) является улучшением при замене x_i на y_i. Тогда G_i равна сумме g_1 + g_2 + … + g_i.

Этот критерий остановки играет большую роль в эффективности алгоритма. Требование, чтобы каждая частичная сумма, G_i, была положительной, сразу кажется, что это слишком ограничительно. Однако то, что это не так, следует из следующего простого факта:

Если последовательность чисел имеет положительную сумму, существует циклическая перестановка этих чисел, так что каждая частичная сумма положительна.

(4) Критерий дизъюнктивности

Наконец, требуется, чтобы множества X и Y не пересекались. Это упрощает реализацию, сокращает время выполнения и обеспечивает эффективный критерий остановки.

Ниже приводится краткое описание основного алгоритма для оптимизации выбранного подтура(упрощенная версия исходного алгоритма).

1. Сгенерируем случайный начальный тур T.

2. Пусть i = 1. Выберем t1

3. Выберем x1 = (t_1, t_2) $\in$ T

4. Выберем y1 = (t_2, t_3) $\notin$ *T* так, чтобы G1> 0. Если это невозможно, переходим на шаг 12

5. Пусть i = i + 1

6. Выберем x_i = (t_(2i-1), t_2i) $\in$ T так, чтобы

(a) Если t_2i присоединен к t_1, тогда исходом в результате будет тур T` и

(b) x_i $\neq$ y_s для всех s < i

Если тур T` лучше тура T, то T = T`, и мы переходим на шаг 2

7. Выберем $y\_i = (t\_2i, t\_(2i+1)) \notin T$ так, чтобы

(a) $G\_i > 0$

(b) $y_i \neq x_$s для всех s ≤ i, и

(c) x_(i+1) существует

Если такой y_i существует, то переходим на шаг 5

8. Если есть неиспытанный вариант для y_2, пусть i = 2, и переходим к шагу 7

9. Если есть неиспытанный вариант для x_2, пусть i = 2, и переходим к шагу 6

10. Если есть неиспытанный вариант для y_1, пусть i = 1, и переходим к шагу 4

11. Если есть неиспытанный вариант для x_1, пусть i = 1, и переходим к шагу 3

12. Если есть неиспытанный вариант для t_1, то переходим к шагу 2

13. Стоп (или переходим на шаг 1)

Комментарии к алгоритму:

Шаг 1. В качестве отправной точки для исследований выбирается случайный тур.

Шаг 3. Выберите звено x_1 = (t_1, t_2) в туре. Когда выбран t_1, есть два варианта для x_1. Здесь глагол “выбрать” означает “выбрать неиспытанный вариант”. Однако каждый раз, когда было найдено улучшение тура (на шаге 6), все альтернативы считаются неиспробованными.

Шаг 6. Есть два варианта x_i. Однако для данного y_(i-1) (i ≥ 2) только один из них позволяет “закрыть” тур (путем добавления y_i). Другой выбор приводит к двум несвязанным подтурам. Однако только в одном случае допускается такой неосуществимый выбор, а именно при i = 2. На рисунке показана эта ситуация.

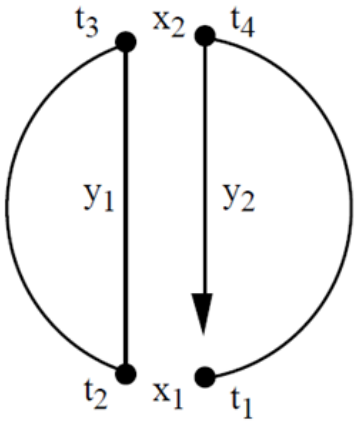


Figure 3.5 No close up at x_2.

Если y_2 выбран так, чтобы t_5 лежал между t_2 и t_3, то тур может быть закрыт на следующем шаге. Но тогда t_6 может находиться по обе стороны от t_5 (см. Рисунок 3.6); оригинальный алгоритм исследовал обе альтернативы.

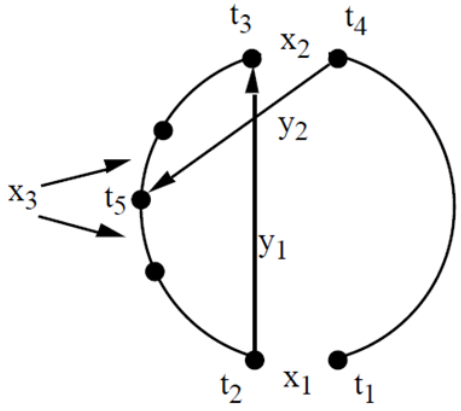


Figure 3.6 Two choices for x_3.

С другой стороны, если y2 выбран так, чтобы t5 лежал между t4 и t2, для t6 есть только один выбор (он должен лежать между t4 и t5), а t7 должен лежать между t2 и t3. Но тогда t8 может находиться по обе стороны от t7. Оригинальный алгоритм исследовал альтернативу, для которой c(t7, t8) является максимальным.

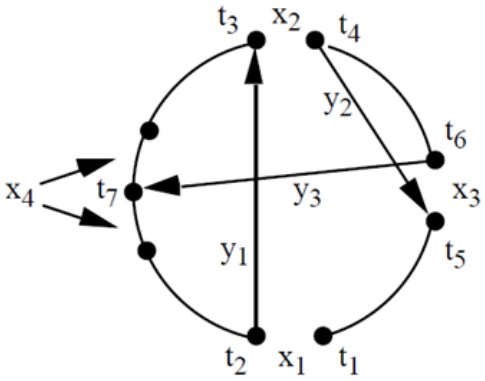


Figure 3.7 Unique choice for x_3. Limited choice of y_3. Two choices for x_4.

Условие (b) на шаге 6 и шаге 7 гарантирует, что множества X и Y не пересекаются: y_i не должен быть ранее разорванной связью, а x_i не должен быть ранее добавленной связью.

Шаги 8-12. Эти шаги вызывают откат назад. Обратите внимание, что возврат разрешен только в том случае, если улучшения не обнаружено, и только на уровнях 1 и 2.

Шаг 13. Алгоритм завершается туром решения, когда все значения t_1 были проверены без улучшения. При необходимости на шаге 1 может быть рассмотрен новый случайный начальный тур.

Описанный выше алгоритм отличается от оригинального своей реакцией на улучшения тура. В приведенном выше алгоритме тур T заменяется более коротким туром T', как только будет найдено улучшение (на шаге 6). В отличие от этого оригинальный алгоритм продолжает свои шаги, добавляя потенциальные обмены, чтобы найти еще более короткий тур. Когда больше обмены невозможны или когда Gi ≤ G*, где G* - лучшее улучшение T, зарегистрированное на данный момент, поиск прекращается, и текущий тур T заменяется наиболее выгодным туром. В своей статье [1] Лин и Керниган не изложили причины внедрения этого метода. Это усложняет реализацию и не приводит ни к лучшим решениям, ни к сокращению времени выполнения.

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

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 содержит 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

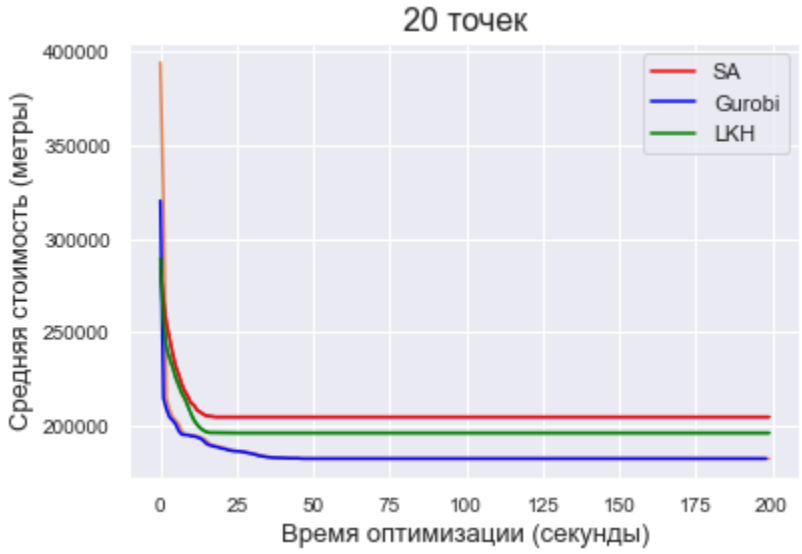
Перед тем, как сравнивать метаэвристические алгоритмы с коммерческим проектом Gurobi необходимо было узнать какой стоит использовать алгоритм для сравнений 2-opt или 3-opt. Поэтому были проведены эксперименты алгоритмов 2-opt и 3-opt для решения задач CVRP и CVRPTW.

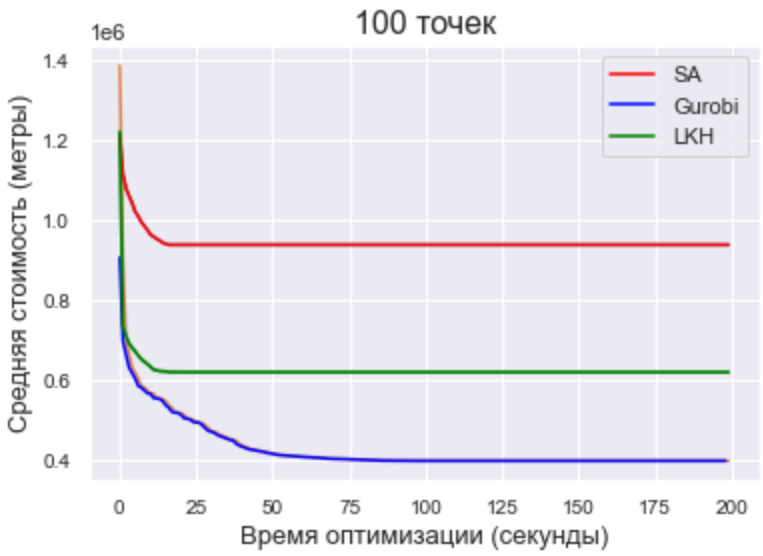
Графики

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 средства для

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





По полученным результатам можно сделать выводы:

- Для маленького количества клиентов (20 клиентов) алгоритм “SA” демонстрирует неплохое решение, приближенное к “Gurobi”. Однако с увеличением количества городов, например как видно для случаев 50, 100 городов, результат ухудшается, поэтому для больших задач данный алгоритм не подходит.
- Коммерческий проект “Gurobi”, в основе которого находится метод “Ветвей и границ”, превосходит по результатам метаэвристические алгоритмы “LKH” и “SA”. Данный вывод явно можно сделать для случаев 50 и 100 городов. Однако для получения итогового решения метаэвристическим алгоритмам требуется гораздо меньше времени для поиска решения. По графикам видно, что в среднем на оптимизацию требуется не больше 25 секунд, в то время “Gurobi” требуется около 50-75 секунд. На задачах большего размера данная разница во времени будет только увеличиваться. Таким образом, подтверждается тот факт, что хоть метаэвристические алгоритмы дают приближенное решение, однако им требуется в несколько раз меньше времени на решение задачи, чем MIP алгоритмам.