Санкт-Петербургский государственный университет

Скрипцов Дмитрий Алексеевич

Магистерская диссертация

Задача организации пассажирских перевозок частной компанией в условиях пандемии

Уровень образования: магистратура

Направление 01.04.02 «Прикладная математика и информатика»

Основная образовательная программа ВМ.5505.2019 «Математическое и информационное обеспечение экономической деятельности»

Научный руководитель:

доцент, Кафедра технологии программирования,

Раевская Анастасия Павловна

Санкт-Петербург

2021

Содержание

Введение

В декабре 2019 года в Китае была зарегистрирована первая вспышка коронавируса, а 11 марта того же года Всемирная организация здравоохранения объявила ее пандемией. Человечество борется с вирусом уже на протяжении двух лет, и нельзя не отметить, как сильно он повлиял на нашу жизнь.

Чтобы помешать распространению вируса и обезопасить свой бизнес и своих сотрудников, многие компании стали массово переходить на удалённую работу, однако далеко не у всех есть такая возможность. Производственные предприятия и заводы не могут работать в удаленном режиме, но если в данной ситуации не предпринять необходимые меры, здоровье сотрудников будет подвержено опасности, а значит и работа самого предприятия попадает под удар.

Одним из решений данной проблемы является организация пассажирских перевозок для сотрудников компании. Это позволит в некоторой мере изолировать предприятие, ограничивая пересечение сотрудников со случайными людьми в общественном транспорте и на улицах города в час пик.

Постановка задачи

Необходимо как можно быстрее доставить всех сотрудников на территорию предприятия с помощью имеющегося транспортного средства. В силу того, что на предприятии может работать большое количество людей, и некоторые из них могут жить рядом друг с другом, рациональным будет решение собирать таких соседей в одной точке для оптимизации процесса. Однако, как было сказано ранее, пандемия накладывает ограничения на безопасное передвижение людей по городу. Поэтому описанные точки сбора должны располагаться не дальше некоторого расстояния R от сотрудника, «привязанного» к этой точке.

Решение поставленной задачи должно быть реализовано в виде программы, способной кластеризовать адреса сотрудников и проложить через все кластеры оптимальный маршрут, начало и конец которого – территория предприятия.

Поставленную задачу разделим на две подзадачи: кластеризация адресов и поиск оптимального маршрута.

В задачу кластеризации входит получение координат по адресам, анализ алгоритмов кластеризации и выбор наиболее подходящего из них.

Задача маршрутизации в данном случае, по сути, представляет собой задачу коммивояжера. Необходимо провести сравнительный анализ методов решения задачи и реализовать один из них.

Обзор литературы

Кластерный анализ и транспортная логистика начали развиваться лишь в прошлом веке, но интерес к этим областям растет с каждым годом. На данный момент существует немало книг и статей, посвященных этим областям, и среди них продолжают появляться новые методы и исследования.

Наиболее важные идеи кластерного анализа, а так же подробное описание нескольких методов кластеризации и целевых функций можно найти в книге Гитиса Л. Х. [1], посвященной теории распознавания образов и кластерному анализу, как одному из методов ее реализации. Дополнит теоретический материал и даст практические рекомендации Айвазян С. А. и его авторский коллектив [2].

Больше неклассических методов, их реализация и связь с геоданными можно найти в статье Дулина, Розенберга и Уманского [3]. В труде представлен обзор подходов, «наиболее успешно используемых в последнее десятилетие для эффективного исследования геоданных». Описана классификация методов кластеризации, подробно рассмотрены более современные методы и области их применения.

Одной из важных особенностей методов кластеризации является возможность запрограммировать их, что наглядно показано в статье Joseph Magiya [4], в которой автор реализует метод k-средних на языке программирования Python.

Среди множества математических методов и моделей в логистике в книгах Тихомировой, Сидоренко [5] и Богданова и Селезнева [6] найдется информация и о транспортной логистике и задачах обслуживания. Даны базовые определения и приведены решения некоторых задач.

Более подробный разбор некоторых методов решения задачи коммивояжера, а так же их сравнительный анализ представлен в трудах Поборчего [7], Гарабы И.В. [8], Борознова В.О. [9] [12] и Ивашечкина [11]. Хотя задача коммивояжера проста для понимания, а рассматриваемые

алгоритмы давно изучены, она принадлежит классу NP-полных задач, а рассмотренные работы помогут оценить эффективность других методов и выявить наиболее подходящий для нашей задачи.

Некоторые алгоритмы поддаются распараллеливанию. Это означает, что с помощью современных технологий можно значительно ускорить работу таких алгоритмов без изменения сути самого алгоритма. О таком способе оптимизировать процесс поиска решения пишут Васильчиков В. В. [10] и Семенкина О. Е. [14].

Согласно многим исследованиям, в частности работе Мартынова А. В. [15], использование сразу нескольких алгоритмов и их соединение в один может положительно сказаться на точности и скорости работы.

§1. Задача кластеризации

Как было сказано ранее, некоторые сотрудники компании могут жить недалеко друг от друга. Если собрать соседей в одной точке, то можно сократить число остановок транспортного средства, что не только ускорит транспортировку, но и упростит вычисления. Таким образом, задача кластеризации заключается в разбиении множества элементов, обладающих некоторым набором признаков, на группу подмножеств.

Задача кластерного анализа в общей постановке выглядит следующим образом: множество объектов разбить на сравнительно небольшое число однородных в определенном смысле подмножеств. То есть каждый объект должен принадлежать одному и только одному подмножеству разбиения и элементы одного подмножества должны быть схожими, в то время как элементы разных подмножеств значительно отличаются друг от друга.

Для решения задач кластерного анализа необходимо определить меру сходства объектов, которая называется функцией расстояния или сходства. В нашем случае мерой сходства является расстояние между двумя точками. Каждая точка задается географическими координатами – широтой и долготой. Для того чтобы получить расстояние между такими точками, можно использовать формулу гаверсинуса:

$$\mathrm{hav}\left(\frac{d}{r}\right) = \mathrm{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\mathrm{hav}(\lambda_2 - \lambda_1), \qquad (1)$$

где d – расстояние между точками на сфере, r – радиус сферы, φ_1 и φ_2 - широта точек в радианах, λ_1 и λ_2 – долгота в радианах.

Выразим d из формулы:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right), \qquad (2)$$

получим формулу расстояния между двумя точками на сфере, которую будем использовать в качестве функции расстояния. Стоит отметить, что более точные данные можно получить с помощью карт с функцией построения

маршрута между двумя точками, например Яндекс.Карты или Google Maps, однако данный метод является более сложным с технической точки зрения, а обработка большого массива данных будет проще, если использовать формулу (2).

В качестве критерия правильности разбиения методами кластерного анализа используется функционал качества разбиения. Наилучшим разбиением является то, на котором достигается экстремум выбранного функционала качества. Сравнив результаты работы алгоритма с помощью целевой функции, мы сможем определить наиболее оптимальное разбиение [1] [2].

1.1 Получение и обработка геокоординат

Для получения геокоординат на основе адресов проживания сотрудников выбран геокодер API HERE Technologies. На языке программирования Python был создан скрипт, отправляющий запрос в формате .csv файла с id и адресами, и распаковывающий полученный архив с текстовым файлом, но уже со списком координат и некоторых характеристик для каждого объекта. Работа скрипта продемонстрирована на рисунке 1, рисунке 2 и рисунке 3.

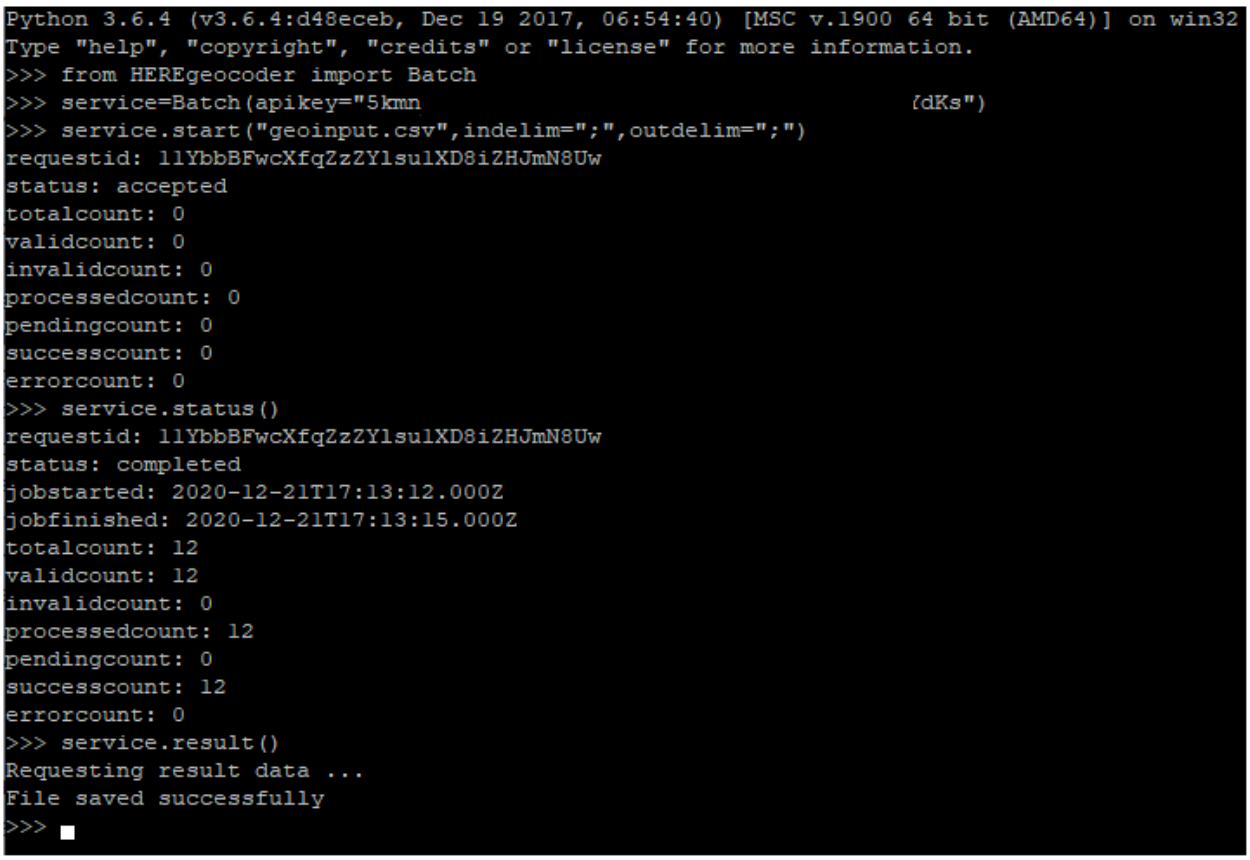

Рисунок 1. - Отправка запроса с помощью скрипта HEREgeocoder и файла geoinput.csv и сохранение ответа. Скрипт представлен в приложении 1.

	A	B	C	D
1	recId	searchText	country	
2	1	Санкт-Петербург, ул. Коллонтай, 6	RUS	
3	2	Москва, Алкон 1, Ленинградский пр-т., 72	RUS	
4	3	425 W Randolph St Chicago IL 60606	USA	
5	4	Румыния, DJ106 20-30, Sibiu 557260	ROU	
6	5	200 S Mathilda Ave Sunnyvale CA 94086	USA	
7	6	Moscow, Арбатская пл., 4, 119019	RUS	
8	7	Moscow, Arbatskaya 4	RUS	
9	8	Saint-Petersburg, Collontay 6	RUS	
10	9	Saint-Petersburg, Universitetskiy 35	RUS	
11	10	Saint-Petersburg, Teatralnaya 1	RUS	
12	11	Saint-Petersburg, Leninskiy 84	RUS	
13	12	Moscow, Alcon 1, Leningradskiy 72	RUS	
14				

Рисунок 2. - Данные в файле geoinput.csv

	A	B	C	D	E	F
1	recId	SeqNumber	seqLength	displayLatitude	displayLongitude	locationLabel
2	1	1	1	55.51063	41.97634	6 СНТ, Муром, Центральный федеральный округ, Россия
3	2	1	1	52.53982	103.87924	72-й квартал 1, Ангарск, Россия, 665830
4	3	1	1	41.88432	-87.63877	425 W Randolph St, Chicago, IL 60606, United States
5	4	1	1	45.85317	23.77604	DJ106G 6, 557152 Miercurea Sibiului, România
6	5	1	1	37.37634	-122.03405	200 S Mathilda Ave, Sunnyvale, CA 94086, United States
7	6	1	1	55.75267	37.58369	119019, Москва, Центральный федеральный округ, Россия
8	7	1	1	55.75154	37.60191	Арбатская площадь 4, Москва, Россия, 119019
9	8	1	1	59.9151901	30.4500481	улица Коллонтай 6, Санкт-Петербург, Россия, 193318
10	9	1	1	59.88204	29.82895	Университетский проспект 35, Санкт-Петербург, Россия, 198504
11	10	1	1	59.92544	30.29594	Театральная площадь 1, Санкт-Петербург, Россия, 190000
12	11	1	1	59.8570627	30.1990699	Ленинский проспект 84, Санкт-Петербург, Россия, 198332
13	12	1	1	55.8041336	37.5193534	Ленинградский проспект 72, Москва, Россия, 125315

Рисунок 3. - Основная информация в ответе сервиса

Стоит отметить, что адреса необходимо писать с помощью транслитерации. Основываясь на проверке небольшой выборки данных, можно сказать, что полученные в результате работы геокодера координаты в действительности соответствуют своим адресам.

1.2 Обзор некоторых алгоритмов кластеризации

В настоящее время существует большое количество различных алгоритмов кластеризации, каждый из которых имеет свои особенности, сильные и слабые стороны. Рассмотреть все алгоритмы, которые могли бы подойти для решения поставленной задачи, не представляется возможным, поэтому ограничимся небольшой выборкой самых известных. Их можно разделить на иерархические и основанные на описании классов «ядрами».

Древовидная кластеризация (Иерархическая кластеризация).

Суть этого алгоритма состоит в постепенном объединении объектов во все большие кластеры, используя некоторую меру сходства или расстояние между объектами. Результатом такой кластеризации является иерархическое дерево. Алгоритм начинается с того, что каждый объект является кластером с одним элементом. На каждом шаге мы ослабляем критерий о том, какие объекты являются уникальными, а какие нет, и объединяем кластеры, которые попадают под этот критерий. В результате, мы связываем вместе всё большее и большее число объектов и кластеров, состоящих из все сильнее различающихся элементов. Окончательно, на последнем шаге все объекты объединяются вместе.

Безусловно, объединение всех элементов в один кластер – не то, чего мы хотим добиться. Тем не менее, анализ полученной в результате работы алгоритма древовидной диаграммы может выявить несколько подходящих под наши условия наборов кластеров. Например, если в какой-то момент кластеры долгое время не будут объединяться, то можно предположить, что мы нашли оптимальное решение. С другой стороны, в случае равномерно распределенных по карте точек, выявление подобного решения может стать затруднительным.

Метод k средних.

Этот метод можно использовать, если мы заранее знаем, сколько кластеров хотим получить. Алгоритм начинается с k случайно выбранных элементов (кластеров), а затем изменяет принадлежность к ним элементов

таким образом, чтобы минимизировать изменчивость внутри кластеров, и максимизировать изменчивость между кластерами.

Главный вопрос, на который придется ответить при использовании алгоритма k средних – какое k выбрать. Можно сделать вывод на основе визуального анализа расположения точек на карте. Можно перебирать k от 1 до тех пор, пока итоговая картина не будет нас устраивать.

Алгоритм Форель (FOREL).

Основан на идее объединения в один кластер объектов в областях их наибольшего сгущения. На каждом шаге мы случайным образом выбираем объект из выборки, раздуваем вокруг него окружность радиуса R, внутри этой сферы выбираем центр тяжести и делаем его центром новой окружности. На каждом шаге двигаем окружность в сторону локального сгущения объектов выборки, то есть стараемся захватить как можно больше объектов выборки окружности фиксированного радиуса. После того как центр стабилизируется, все объекты внутри окружности с этим центром мы помечаем как кластеризованные и выкидываем их из выборки. Этот процесс мы повторяем до тех пор, пока вся выборка не будет кластеризована.

Отличительной особенностью этого алгоритма является его привязка к радиусу кластера, что отлично подходит для нашей задачи. Минусами алгоритма являются плохая применимость при плохой разделимости выборки на кластеры и сильная зависимость от выбора начального объекта

Алгоритм кластеризации, основанный на минимальном покрывающем дереве.

В выбранном графе строится минимальное покрывающее дерево (например, методом Борувки). Из полученного дерева удаляются k наибольших ребер, таким образом, оно делится на части, которые объявляются кластерами. Мы можем последовательно удалять ребра из дерева до тех пор, пока каждый полученный кластер не будет удовлетворять всем нашим условиям (такие как расстояние до центра кластера, количество людей в кластере, и пр.).

Алгоритм хорошо подходит для решения поставленной задачи, так как предназначен для работы с графом, однако из-за построения минимального покрывающего дерева проигрывает в скорости остальным.

Вывод.

На данном этапе был сделан выбор в пользу алгоритма FOREL. Его единственным управляющим параметром является радиус шаров, которые покрывают выборку, что отлично подходит под поставленную задачу.

1.3. Алгоритм FOREL и его программная реализация

1. Случайно выбираем объект из текущей выборки
2. Помечаем объекты выборки, находящиеся в радиусе R от текущего
3. Вычисляем центр тяжести, помечаем его как новый текущий объект
4. Повторяем шаги 2-3, пока меняется центр тяжести
5. Кластеризованные объекты удаляем из выборки
6. Повторяем шаги 1-5, пока выборка не пуста

Алгоритм минимизирует функционал следующего вида:

$$F = \sum_{j=1}^{k} \sum_{x \epsilon K_j} \rho(x, W_j),$$

где k - количество кластеров, $x \epsilon K_j$ - элемент x кластера K_j, $\rho(x, W_j)$ - расстояние от элемента x до центра кластера W_j. Прогнав алгоритм несколько раз с разными начальными точками, мы получим множество решений. Сравним их с помощью функционала качества – и получим лучшее разбиение из рассматриваемого множества.

Программа, реализующая алгоритм, была написана на Python и представлена в приложении 2.

Прежде чем начать работу, необходимо извлечь координаты из файла geooutput.csv, который содержит данные в известном нам виде. Для простоты будем считать, что данные выглядят следующим образом:

	A	B	C	D	E
1	id	lat	long	adress	
2	1	59.942164	30.274546	Средний проспект Васильевского острова 41	
3	2	59.941371	30.278067	8-я линия Васильевского острова 39	
4	3	59.938766	30.276477	11-я линия Васильевского острова 18	
5	4	59.938162	30.277187	11-я линия Васильевского острова 14/39	
6	5	59.938279	30.277860	10-я линия Васильевского острова 5	
7	6	59.939063	30.284220	Большой проспект Васильевского острова 18	
8					

Рисунок 4 - Файл geooutput для алгоритма FOREL

Далее эти данные переносятся во вложенный список, и алгоритм начинает свою работу. В силу того, что создать большую выборку из

настоящих адресов проживания довольно трудно, была вдобавок реализована функция, создающая точки (пару координат) в пределах [59.86, 59.97] для широты и [30.27, 30.505] для долготы. При таком выборе границ большинство точек попадет на территорию Санкт-Петербурга, очерченную ЗСД и КАД.

Для одного набора точек проводится несколько разбиений, после чего сравниваются значения функционала качества каждого разбиения, и выбирается лучшее из полученных решений.

Итогом работы программы является список all_clusters, который содержит в себе, в том числе, список всех созданных разбиений. Переменная min_clust является номером разбиения, которое было выбрано лучшим.

Образец вывода программы для данных из рисунка 4 представлен на рисунке 5

```
Разбиение №1
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 1 включает точки с индексами: 3 4 5
Кластер 2 включает точки с индексами: 1 2
Кластер 3 включает точки с индексами: 6
Разбиение №2
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 1 включает точки с индексами: 4 3 5
Кластер 2 включает точки с индексами: 2 1
Кластер 3 включает точки с индексами: 6
Разбиение №3
Значение функционала качества: 0.3383948057338315
Количество кластеров: 3
Кластер 2 включает точки с индексами: 1 2
Кластер 1 включает точки с индексами: 5 3 4
Кластер 3 включает точки с индексами: 6
Разбиение №4
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 3 включает точки с индексами: 6
Кластер 1 включает точки с индексами: 4 3 5
Кластер 2 включает точки с индексами: 2 1
Разбиение №5
Значение функционала качества: 0.3383948057338315
Количество кластеров: 3
Кластер 3 включает точки с индексами: 6
Кластер 1 включает точки с индексами: 5 3 4
Кластер 2 включает точки с индексами: 2 1
Разбиение №6
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 1 включает точки с индексами: 4 3 5
Кластер 3 включает точки с индексами: 6
Кластер 2 включает точки с индексами: 1 2
Разбиение №7
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 1 включает точки с индексами: 4 3 5
Кластер 2 включает точки с индексами: 1 2
Кластер 3 включает точки с индексами: 6
Разбиение №8
Значение функционала качества: 0.3383948057338315
Количество кластеров: 3
Кластер 1 включает точки с индексами: 5 3 4
Кластер 2 включает точки с индексами: 1 2
Кластер 3 включает точки с индексами: 6
Разбиение №9
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 1 включает точки с индексами: 4 3 5
Кластер 2 включает точки с индексами: 1 2
Кластер 3 включает точки с индексами: 6
Разбиение №10
Значение функционала качества: 0.33839480573260555
Количество кластеров: 3
Кластер 1 включает точки с индексами: 4 3 5
Кластер 2 включает точки с индексами: 2 1
Кластер 3 включает точки с индексами: 6
Наилучший результат показал кластер №1
со значением функционала = 0.33839480573260555
```

Рисунок 5 - Образец вывода программы, реализующей алгоритм FOREL

Так же были проведены эксперименты со случайным набором точек для разного числа точек, радиуса круга и количества разбиений. При 1000 случайных точек и 100 итераций алгоритм завершает свою работу примерно за 3 минуты. При тестировании использовался процессор AMD Ryzen 5 2600 3,40 GHz с 16 ГБ оперативной памяти.

§2. Задача коммивояжера

Задача коммивояжера – классическая задача комбинаторики, заключающаяся в поиске самого выгодного маршрута, проходящего через все указанные точки (вершины) хотя бы по одному разу и возвращающегося в исходную точку.

Хотя постановка задачи проста и понятна, она относится к классу NP-полных задач, а оптимизационная задача коммивояжера NP-трудна. Точные методы ее решения имеют экспоненциальную сложность, что не позволяет получить решение задачи за разумное время при большой размерности.

Дан полный взвешенный граф, в нем необходимо найти Гамильтонов цикл с наименьшим суммарным весом по ребрам. Цикл называется Гамильтоновым, если он содержит все ребра и каждое из них – лишь один раз. Математически задача может быть представлена следующей целевой функцией:

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \to \min \qquad (3)$$

с ограничениями вида:

$$\begin{cases} \sum_{j=1}^{n} x_{ij} = 1\ , & i = 1 \dots n \\ \sum_{i=1}^{n} x_{ij} = 1\ , & j = 1 \dots n \\ u_i - u_j + nx \leq n - 1, & i, j = 1 \dots n, i \neq j \end{cases} \qquad (4)$$

где c_{ij} - длина пути из вершины i в вершину j, x_{ij} - бинарная переменная, равная 1, если путь был выбран и 0 в обратном случае, u_i- номер шага, на котором посетили точку i. Последнее ограничение обеспечивает замкнутость маршрута и отсутствие петель.

В связи с тем, что решение задачи происходит в условиях города, рассматриваемый граф является полным. Так же предположим, что работаем с неориентированным графом, то есть задача коммивояжера симметричная. Вершинами будем считать найденные в ходе решения задачи кластеризации точки остановки транспорта, а вес ребер является длиной пути между

вершинами. В данном случае длина пути эквивалентна времени проезда между смежными вершинами.

2.1. Методы решение задачи коммивояжера

Методы решения задачи коммивояжера можно разделить на точные и эвристические.

Точные, в соответствии со своим названием, дают точное решение задачи, однако поиск решения может занять продолжительное время, в силу того что он осуществляется перебором всех возможных вариантов.

Методы, сокращающие полный перебор – эвристические. Они не строят наиболее эффективный маршрут, а дают приближенное решение, хотя которое, как правило, является достаточно хорошим. Зато это решение будет получено намного быстрее, если речь идет о задачах большой размерности.

Среди множества методов выделим наиболее известные, а так же некоторые их модификации:

1. Метод полного перебора.
2. Метод ветвей и границ. Алгоритм Литтла, его распараллеленная версия.
3. Метод ближайшего соседа.
4. Метод имитации обжига.
5. Прима-Эйлера.
6. BV-метод.
7. Генетический алгоритм, ГА с частичной параллелизацией.
8. Муравьиный алгоритм ASC и ASC-Q.
9. Гибридный алгоритм.

2.2. Сравнительный анализ методов решения задачи коммивояжера

Метод полного перебора (МПП)

Осуществляет поиск решения перебором всех возможных вариантов. Результатом работы является точное решение, однако, уже при малом количестве вершин графа N время работы метода может достигать абсурдных значений. Метод может быть использован для оценки точности других методов при малых N.

Метод ветвей и границ. Алгоритм Литтла, и его оптимизированная и распараллеленная версии.

Метод ветвей и границ - точный, поэтому, как и МПП, применим только для малых N. Отличается от МПП отсевом множеств допустимых решений, которые не содержат оптимального решения, поэтому сложность данного метода меньше. [12]

Алгоритм Литтла является частным случаем метода ветвей и границ, предназначенным для решения задачи коммивояжера. Согласно труду Поборчего [7] время работы алгоритма для асимметричной задачи коммивояжера при N=150 составляет 436,28 секунд.

Для неориентированного графа время работы алгоритма значительно возрастает, чего можно избежать, модифицировав алгоритм одним из представленных Васильчиковым способом (дополнительная редукция, «оценка нуля», изменение матрицы при добавлении и удалении ребра) [10]. Небольшое количество проведенных экспериментов показало, что оптимизированный алгоритм может иметь значительное преимущество в быстродействии над обычной версией.

Одной из интересных особенностей точных алгоритмов является возможность организации параллельных вычислений. В исследовании Васильчикова показано, что как обычный, так и оптимизированный алгоритм Литтла выигрывают от применения распараллеливания. В случае с

ориентированным графом, N=70 и 8 параллельными потоками обычный алгоритм Литтла работает быстрее однопоточной версии в 3,91 – 9,75 раз, находя решение задачи за 63 – 106 секунд.

Метод ближайшего соседа.

Основан на предположении о том, что если мы будем посещать ближайшую точку на каждом шаге, то полученный путь будет сравнительно хорошим решением.

Относится к «жадным» алгоритмам, то есть дает локально оптимальное решение. Работает чрезвычайно быстро и для задачи коммивояжера может дать неплохое приближение, однако дает слишком большую погрешность при больших N [7].

Для того, чтобы увеличить точность метода, можно применять алгоритм несколько раз, циклически меняя при этом начальную точку для алгоритма. Такой метод дает в среднем на 16% более точные результаты [7]. Время работы при этом так же возрастает, но все остается сравнительно маленьким. Тем не менее, проигрыш результатов относительно точного решения для N=150 составляет 26,56% и будет расти вместе с размерностью задачи.

Метод имитации обжига.

Имитирует процесс кристаллизации вещества. Похож на градиентный спуск, но благодаря случайности выбора промежуточной точки имеет меньший шанс попасть в локальный минимум (максимум).

Согласно труду Ивашечкина [11], для N=38, N=194 и N=980 время работы метода имитации обжига составляет 5, 31 и 125 минут соответственно, при этом проигрыш в точности по сравнению с методом ветвей и границ составил около 10%.

Однако другое исследование отмечает сильное ухудшение точности результатов работы метода имитации обжига при N > 10 [8]. Такое разногласие может являться следствием малого количества экспериментов.

Метод Прима-Эйлера.

Является сочетанием двух методов: сначала с помощью метода Прима строится каркас минимального дерева, затем применяется метод Эйлера – находим эйлеров цикл и преобразуем его в гамильтонов цикл. Отметим, что алгоритм можно применять только в том случае, когда для вершин графа выполняется неравенство треугольника.

Сравнительный анализ Гарабы [8] показал, что при количестве вершин большем 10 точность сильно падает относительно результатов работы метода ближайшего соседа, генетического алгоритма и алгоритма Литтла.

BV-метод.

Состоит из трех этапов: получение эталонного решения «жадным» методом (ближайшего соседа), построение BV-матрицы, и оптимизация этого решения с помощью BV-модификаторов.

Исследования Борознова [9] [12] показали, что BV-метод в скорости и точности превосходит муравьиный алгоритм ACS-Q и классический генетический алгоритм, однако в задачах с матрицей расстояний, содержащей близкие или равные друг другу значения, генетический алгоритм дает лучший результат.

Генетический алгоритм (ГА) и его частичная параллелизация.

Генетический алгоритм является один из самых популярных, что обусловлено его оптимальностью в смысле соотношения результат/время, а так же гибкостью настройки.

Алгоритм имитирует процесс эволюции в природе и состоит из четырех основных этапов: генерация поколения, отбор, скрещивание и мутация. Количество особей в поколении, количество поколений, метод отбора и скрещивания, и вероятность мутации являются настраиваемыми параметрами.

Решение, полученное с помощью ГА, обладает хорошей точностью в сравнении с «жадными» алгоритмами, муравьиным алгоритмом и BV-методом [8] [9] [12] [13]. Хотя время работы ГА как правило больше своих визави, оно

все еще остается в пределах разумного. Так, например, задачу с N=160 ГА решил за ~31,5 секунд, проиграв в точности 6,71% BV-методу [12].

Ускорить поиск решения поможет параллелизация ГА. Согласно исследованиям Пугина и Семенкиной [16] [14] увеличение количества потоков в n раз приводит к ускорению работы алгоритма чуть меньше чем в n раз. При этом качество решения остается прежним или меняется незначительно [14].

Муравьиный алгоритм (МА).

Как и гибридный алгоритм, является поисковым алгоритмом, в некоторой степени имитирующим жизненный процесс колонии муравьев. На каждом шаге муравьи выбирают город для перехода, основываясь на феромонах, оставленных колонией, т.е. на истории посещения городов другими муравьями. Настраиваемыми параметрами являются: количество муравьев, коэффициент испарения феромона, важность расстояния между городами и текущего решения муравья. При удачном подборе параметров, определяющих «жадность» и «стадность» алгоритма, можно добиться хорошего приближения к оптимальному решению.

Так, алгоритм ACS-Q (модификация, в которой изменена формула для феромона) дает лучшие результаты в сравнении с генетическим алгоритмом, уступив ему в быстродействии [9]. Конкурентоспособность муравьиного алгоритма доказывает и исследование Семенова [13], согласно которому точность МА выше точности метода имитации отжига и немного проигрывает ГА, при заметно лучших показателях скорости.

МА поддается распараллеливанию, причем, как и в случае ГА, значительное улучшение получает только быстродействие алгоритма [14].

Гибридный алгоритм.

Алгоритм, предложенный Мартыновым и Курейчиком [15], включает в процесс работы муравьиного алгоритма генетический алгоритм. МА производит поиск кратчайшего пути в графе с помощью колонии муравьев, при этом наиболее приспособленные особи передают генетическую информацию последующим поколениям с помощью инструментов ГА.

Тесты показали, что для N=101 время работы алгоритма составило 2 минуты, а точность в сравнении с муравьиным алгоритмом ACO возросла, причем, чем больше N – тем заметнее улучшение точности.

Вывод.

Для того чтобы определить наиболее подходящий для решения задачи метод, необходимо в первую очередь обратить внимание на точность и скорость работы методов.

В силу того, что для одной компании нужно найти лишь одно решение, которое может изредка меняться в случае, например, неспособности некоторых сотрудников выйти на работу, мы имеем возможность выделить на поиск оптимального пути вплоть до нескольких часов. Подобные ограничения, разумеется, необходимо обсудить с заказчиком (владельцем компании), а пока предположим, что в распоряжении алгоритма будет, по меньшей мере, 1 час.

Точность, в свою очередь, является более важным критерием, так как от скорости транспортировки сотрудников напрямую зависит рабочее время или (в случае ранних сборов и поздних возвращений домой) время сотрудников, отведенной на восстановление сил.

Благодаря кластеризации количество точек в графе задачи коммивояжера должно заметно снизиться, но это не исключает возможности столкнуться с задачей большой размерности.

Учитывая все вышеперечисленное, было принято решение реализовать BV-метод, так как он показал хорошие результаты по сравнению с другими методами, а так же сравнительно редко появляется в научных статьях.

2.3. BV-метод

BV-метод был получен на основе сравнения точных и приближенных решений [9]. Он основан на анализе и оптимизации эталонного маршрута, полученного «жадным» алгоритмом. Как было сказано ранее, BV-метод можно разделить на три этапа.

Первый этап – применение «жадного» алгоритма для получения набора возможных решений. Для N точек строится N маршрутов, каждый из новой начальной точки. Среди них выбирается наилучший – эталонный – маршрут.

На втором этапе алгоритма строится BV-матрица. Для этого рассматриваются все N маршрутов, и элемент нулевой матрицы N*N увеличивается на 1, если в маршруте используется соответствующее ребро.

Третий этап алгоритма посвящен применению BV-модификаторов. К эталонному решению последовательно применяются каждый из четырех (если это возможно) BV-модификаторов. В результате будут получены новые маршруты, и если среди них найдется такой путь, длина которого меньше длины эталонного маршрута, то он становится новым эталонным решением.

Существует 4 BV-модификатора [9]:

1. модификатор «Сглаживания» - меняет местами позиции двух городов;
2. модификатор «Переноса» - сдвигает выбранный интервал на 1 влево, а первый город ставит на позицию второго;
3. «Инверсный переход» - перезаписывает города в открытом интервале в обратном порядке;
4. «Инверсный перенос» - перезаписывает города в закрытом интервале в обратном порядке.

2.4. Реализация BV-метода

Для начала необходимо реализовать метод ближайшего соседа и получить N маршрутов с его помощью. Запоминаем пройденные города и меняем начальную точку для каждого подхода, все полученные данные сохраняем и среди них выделяем эталонное решение.

Строим BV-матрицу согласно описанному выше принципу: если в маршруте используется ребро (i, j), то к элементу (i, j) нулевой матрицы прибавляется единица.

Далее, перебирая элементы полученной матрицы, для каждого ненулевого применяем BV-модификаторы. В качестве входных данных модификаторы получают номера городов, соответствующие индексам ненулевого элемента BV-матрицы. Если в результате модификации был получен маршрут, длина которого меньше всех предыдущих, то он принимается за эталонный, а элемент BV-матрицы приравнивается к 0. Алгоритм продолжается до тех пор, пока мы находим лучшие маршруты.

Ряд проведенных экспериментов со случайной матрицей расстояний показал, что алгоритм дает приближенное решение задачи коммивояжера, и показывает хорошую скорость даже при больших размерностях.

Программная реализация на языке Python представлена в приложении 3.

Заключение

В результате проведенной работы была решена поставленная задача транспортировки сотрудников, а именно: создан скрипт, получающий географические координаты для требуемых адресов; дан обзор некоторых алгоритмов кластеризации, реализован один из них и приспособлен для работы с географическими координатами; проведен сравнительный анализ некоторых методов решения задачи коммивояжера, реализован BV-метод, показавший хорошие результаты как при оценке точности, так и по производительности.

В дальнейшем возможно рассмотрение более сложных задач кластеризации и коммивояжера. Так, например, рационально будет оценивать расстояние между точками на карте, основываясь на действительном времени передвижения между ними, так же городские условия могут добавить такие нюансы, как пробки, дорожные работы, непересекаемые препятствия вроде рек и больших сооружений.

В случае с задачей коммивояжера, следующим шагом может стать решение задачи коммивояжера с несколькими транспортными средствами и ограничением по загруженности.

Список литературы

1. Гитис Л. Х. Статистическая классификация и кластерный анализ. – М.: Издательство Московского государственного горного университета, 2003. – 157 с.

2. Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: Классификация и снижение размерности: Справ. изд. – М.: Финансы и статистика, 1989. – 607 с.: ил.

3. Дулин С.К., Розенберг И.Н., Уманский В.И. Методы кластеризации в исследовании массивов геоданных // Системы и средства информатики. – 2009. - Дополнительный выпуск – С. 86–113.

4. Joseph Magiya, Clustering GPS Coordinates and Forming Regions with Python // gitconnected.com. 2019. Режим доступа: https://levelup.gitconnected.com/clustering-gps-co-ordinates-forming-regions-4f50caa7e4a1.

5. Тихомирова А. Н., Сидоренко Е. В. Математические модели и методы в логистике: Учебное пособие. – М.: НИЯУ МИФИ, 2010 – 320 с.

6. Богданов А. И., Селезнев А. А. Математические модели и методы в логистике: монография. – СПб.: ФГБОУВО «СПбГУПТД», 2016. – 105 с.

7. Поборчий И. В. Исследование эвристических методов решения задачи коммивояжера // СПбГУ 2016. Режим доступа: http://hdl.handle.net/11701/4478.

8. Гараба И. В. Сравнительный анализ методов решения задачи коммивояжера для выбора маршрута прокладки кабеля сети кольцевой архитектуры // Молодежный научно-технический вестник. М: ФГБОУ ВПО «МГТУ им. Н. Э. Баумана», 2013.

9. Борознов В. О. Исследование решения задачи коммивояжера // Вестник Астраханского государственного технического университета, 2009

10. Васильчиков В. В. Об оптимизации и распараллеливании алгоритма Литтла для решения задачи коммивояжера // Моделирование и анализ

информационных систем, том 23, номер 4, 2016. Режим доступа: https://doi.org/10.18255/1818-1015-2016-4-401-411.

11. Ивашечкин А. П. Динамическая адаптация метода имитации отжига для решения задачи коммивояжера // СПбГУ 2018.

12. Борознов В. О. Исследование эвристического метода решения задачи коммивояжера // Электронный научный жернал «ИССЛЕДОВАНО В РОССИИ» 2008. Режим доступа: https://zhurnal.ape.relarn.ru/articles/2008/028.pdf

13. Семенов С. С., Педан, А. В., Воловиков В. С., Климов И. С. Анализ трудоемкости различных алгоритмических подходов для решения задачи коммивояжера // Системы управления, связи и безопасности, 2017. Режим доступа: http://sccs.intelgr.com/archive/2017-01/08-Semenov.pdf

14. Семенкина О. Е. Параллельные генетический и муравьиный алгоритмы для решения задачи коммивояжера // Математические методы моделирования, управления и анализа данных, 2012

15. Мартынов А. В., Курейчик В. М. Гибридный алгоритм решения задачи коммивояжера // Известия ЮФУ. Технические науки, номер 4, раздел 1, 2015.

16. Пугин К. В., Ефимов С. С. Генетические алгоритмы с частичной параллелизацией в системах с обхей памятью на примере задачи коммивояжера // Математические структуры и моделирование, вып. 26, с 110-117, 2012.

Приложение 1

```
import requests
import zipfile
import io
from bs4 import BeautifulSoup


class Batch:

    SERVICE_URL = "https://batch.geocoder.ls.hereapi.com/6.2/jobs"
    jobId = None

    def __init__(self, apikey="Ключ для REST API "):
        self.apikey = apikey

    def start(self, filename, indelim=";", outdelim=";"):
        file = open(filename, 'rb')

        params = {
            "action": "run",
            "apiKey": self.apikey,
            "politicalview": "RUS",
            "gen": 9,
            "maxresults": "1",
            "header": "true",
            "indelim": indelim,
            "outdelim": outdelim,
            "outcols":
"displayLatitude,displayLongitude,locationLabel,houseNumber,street,district,city
,postalCode,county,state,country",
            "outputcombined": "true",
        }

        response = requests.post(self.SERVICE_URL, params = params, data = file)
        self.__stats(response)
        file.close()

    def status(self, jobId = None):
        if jobId is not None:
            self.jobId = jobId

        statusUrl = self.SERVICE_URL + "/" + self.jobId

        params = {
            "action": "status",
            "apiKey": self.apikey,
        }

        response = requests.get(statusUrl, params=params)
        self.__stats(response)

    def result(self, jobId = None):

        if jobId is not None:
            self.jobId = jobId

        print("Requesting result data ...")

        resultUrl = self.SERVICE_URL + "/" + self.jobId + "/result"

        params = {
            "apiKey": self.apikey
```

```
        }

        response = requests.get(resultUrl, params=params, stream=True)

        if (response.ok):
            zipResult = zipfile.ZipFile(io.BytesIO(response.content))
            zipinfos = zipResult.infolist()

            for zipinfo in zipinfos:
                zipinfo.filename = "geooutput.txt"
                zipResult.extract(zipinfo)
            print("File saved successfully")

        else:
            print("Error")
            print(response.text)

    def __stats(self, response):
        if response.ok:
            parsedXMLResponse = BeautifulSoup(response.text, "lxml")

            self.jobId = parsedXMLResponse.find('requestid').get_text()

            for stat in parsedXMLResponse.find('response').findChildren():
                if len(stat.findChildren()) == 0:
                    print("{name}: {data}".format(name=stat.name,
data=stat.get_text()))

        else:
            print(response.text)
```

Приложение 2

```
import csv
from geopy.distance import great_circle
from random import choice, uniform


def random_points_list(N):
    rand_list = []
    for i in range(0, N):
        rand_list.append([i+1, round(uniform(59.86, 59.97), 6),
round(uniform(30.27, 30.505), 6), 0])
    return rand_list


def create_centroid(points):
    sumLat = 0
    sumLong = 0
    for point in points:
        sumLat = sumLat + point[1]
        sumLong = sumLong + point[2]
    return [sumLat/len(points), sumLong/len(points)]


def clustering_quality(cluster_data):
    F = 0
    j = 0
    for cluster in cluster_data[1]:
        for point in cluster:
            F = F + great_circle((point[1], point[2]), (cluster_data[2][j][0],
cluster_data[2][j][1])).kilometers
        j = j + 1
    return F


r_file = open("geooutput.csv")
file_reader = csv.reader(r_file, delimiter=";")
r_data_const = list(file_reader)
r_data_const.pop(0)
r_dict_const = {}
for row in r_data_const:
    row[0] = int(row[0])
    row[1] = float(row[1])
    row[2] = float(row[2])
    row[3] = 0
    r_dict_const[row[0]] = [row[1], row[2], row[3]]
r_file.close()


'''
N = 100
r_data_const = random_points_list(N)
r_dict_const = {}
for data in r_data_const:
    r_dict_const[data[0]] = [data[1], data[2], data[3]]
'''


w_data = []
count = 0
for row in r_data_const:
    for i in range(0, len(r_data_const)-row[0]):
        w_data.append([])
```

```
        w_data[count].append(row[0])
        w_data[count].append(r_data_const[row[0] + i][0])
        w_data[count].append(great_circle((row[1], row[2]), (r_data_const[row[0]
+ i][1], r_data_const[row[0] + i][2])).kilometers)
        count += 1

# ------------------------------- FOREL

R = 0.3

clusterList = []
centroidList = []
all_clusters = []
r_dict = []
r_data = []

for count in range(0, 50):
    i = 0
    r_dict = r_dict_const.copy()
    r_data = r_data_const.copy()
    clusterList.clear()
    centroidList.clear()

    while True:
        if len(r_dict) == 0:
            break

        i = i + 1

        newCenterPointKey = choice(list(r_dict.keys()))

        r_data[newCenterPointKey - 1][3] = i

        del r_dict[newCenterPointKey]

        clusterList.append([])
        clusterList[i - 1].append(r_data[newCenterPointKey - 1])

        if len(r_dict) == 0:
            centroidList.append([r_data[newCenterPointKey - 1][1],
r_data[newCenterPointKey - 1][2]])
            break

        for elem in w_data:
            if elem[0] == newCenterPointKey:
                if elem[2] <= R:
                    r_data[elem[1] - 1][3] = i
                    clusterList[i - 1].append(r_data[elem[1] - 1])
                    del r_dict[elem[1]]
                    if len(r_dict) == 0:
                        break
            elif elem[1] == newCenterPointKey:
                if elem[2] <= R:
                    r_data[elem[0] - 1][3] = i
                    clusterList[i - 1].append(r_data[elem[0] - 1])
                    del r_dict[elem[0]]
                    if len(r_dict) == 0:
                        break

        centroidList.append(create_centroid(clusterList[i - 1]))

        while True:
            j = 0
```

```
            for key in r_dict:
                if great_circle(centroidList[i - 1], (r_data[key - 1][1],
r_data[key - 1][2])).kilometers <= R:
                    j = j + 1
                    r_data[key - 1][3] = i
                    clusterList[i - 1].append(r_data[key - 1])
                    del r_dict[key]

            if j == 0:
                break
            else:
                centroidList[i - 1] = create_centroid(clusterList[i - 1])

    all_clusters.append([r_data.copy(), clusterList.copy(),
centroidList.copy()])
    all_clusters[-1].append(clustering_quality(all_clusters[-1]))


i = 0
min_val = 1000
min_clust = 0
min_len = 0
for data in all_clusters:
    i = i + 1
    print("Разбиение №" + str(i))
    print("Значение функционала качества: " + str(data[3]))

    if min_val > data[3]:
        min_val = data[3]
        min_clust = i
        min_len = len(data[1])

    print("Количество кластеров: " + str(len(data[1])))
    for cluster in data[1]:
        temp = ""
        for point in cluster:
            temp = temp + str(point[0]) + " "
        print("Кластер " + str(cluster[0][3]) + " включает точки с индексами: "
+ temp)


print("Наилучший результат показал кластер №" + str(min_clust) + " \nco
значением функционала = " + str(min_val) +
      " \nи колличеством кластеров = " + str(min_len))
```

Приложение 3

```
import numpy.random as rand
from copy import deepcopy
from numpy import sum


def generate_matrix(dimension):
    randmatrix = []
    for i in range(dimension):
        randmatrix.append([])
        for j in range(dimension):
            if i == j:
                randmatrix[i].append(float('inf'))
            else:
                randmatrix[i].append(rand.randint(1, 100))
    print_matrix(randmatrix)
    return randmatrix


def generate_matrix_symmetric(dimension):
    randmatrix = []
    for i in range(dimension):
        randmatrix.append([])
        for j in range(i + 1):
            if i == j:
                randmatrix[i].append(float('inf'))
            else:
                randmatrix[i].append(rand.randint(1, 100))
    for i in range(dimension):
        for j in range(i + 1, dimension):
            randmatrix[i].append(randmatrix[j][i])
    print_matrix(randmatrix)
    return randmatrix


def print_matrix(matrix):
    for i in range(len(matrix)):
        print(matrix[i])


matrix = [
    [float('inf'), 5, 7, 6, 8, 3],
    [1, float('inf'), 8, 4, 6, 2],
    [3, 9, float('inf'), 6, 5, 3],
    [7, 8, 4, float('inf'), 4, 2],
    [2, 7, 5, 6, float('inf'), 6],
    [5, 2, 6, 4, 5, float('inf')]
]
n = len(matrix[0])

'''
n = 1000
matrix = generate_matrix_symmetric(n)
'''

path_list = []
path_length = []

for i in range(n):
    path_list.append([i])
    path_length.append(0)
```

```
    current_point = i

    while True:
        min_range = float('inf')

        for j in range(n):
            if current_point != j and j not in path_list[i] and
matrix[current_point][j] < min_range:
                min_range = matrix[current_point][j]
                next_point = j

        path_length[i] += matrix[current_point][next_point]
        path_list[i].append(next_point)

        current_point = next_point

        if len(path_list[i]) == n:
            path_length[i] += matrix[current_point][i]
            break


best_path = path_list[path_length.index(min(path_length))]
best_length = min(path_length)


def create_bv_matrix(n, path_list):
    bv_matrix = []
    for i in range(n):
        bv_matrix.append([])
        for j in range(n):
            bv_matrix[i].append(0)

    for path in path_list:
        for i in range(len(path) - 1):
            bv_matrix[path[i]][path[i + 1]] += 1
        bv_matrix[path[-1]][path[0]] += 1

    return bv_matrix


bv_matrix = create_bv_matrix(n, path_list)

def bv_method(bv_mat, b_path, b_length):
    path = b_path
    length = b_length
    while True:
        if sum(bv_mat) == 0:
            break
        count = 0
        for i in range(len(bv_mat)):
            for j in range(len(bv_mat)):
                if bv_mat[i][j] > 0:
                    res1 = bv_smoothing(path, i, j)

                    if abs(i - j) >= 2:
                        res2 = bv_transposition(path, i, j)
                        res4 = bv_inversion_transport(path, i, j)
                    else:
                        res2 = [[], float('inf')]
                        res4 = [[], float('inf')]

                    if abs(i - j) >= 3:
                        res3 = bv_inversion_transition(path, i, j)
```

```
                    else:
                        res3 = [[], float('inf')]

                    res_path = [res1[0], res2[0], res3[0], res4[0]]
                    res_length = [res1[1], res2[1], res3[1], res4[1]]
                    best_modify_length = min(res_length)

                    if length > best_modify_length:
                        b_lenght = best_modify_length
                        path = res_path[res_length.index(best_modify_length)]
                        bv_mat[i][j] = 0
                        count += 1
            if count == 0:
                break

    return [path, length]


def bv_smoothing(b_path, index1, index2):
    path = b_path.copy()
    i = path.index(index1)
    j = path.index(index2)
    path[i] = index2
    path[j] = index1

    length = 0
    for k in range(len(path) - 1):
        length += matrix[path[k]][path[k+1]]
    length += matrix[path[-1]][path[0]]

    return [path, length]


def bv_transposition(b_path, index1, index2):
    path = b_path.copy()
    i = path.index(index1)
    j = path.index(index2)
    if i < j:
        for k in range(j - i):
            path[k+i] = path[k+i+1]
        path[j] = index1
    else:
        for k in range(i - j):
            path[k+j] = path[k+j+1]
        path[i] = index2

    length = 0
    for k in range(len(path) - 1):
        length += matrix[path[k]][path[k + 1]]
    length += matrix[path[-1]][path[0]]

    return [path, length]


def bv_inversion_transition(b_path, index1, index2):
    path = b_path.copy()
    i = path.index(index1)
    j = path.index(index2)
    if i < j:
        temp = []
        for k in range(i - j - 1):
            temp.insert(0, path[k+i])
        c = 1
```

```
            for elem in temp:
                path[i+c] = elem
                c += 1
        else:
            temp = []
            for k in range(j - i - 1):
                temp.insert(0, path[k+j])
            c = 1
            for elem in temp:
                path[j + c] = elem
                c += 1

        length = 0
        for k in range(len(path) - 1):
            length += matrix[path[k]][path[k + 1]]
        length += matrix[path[-1]][path[0]]

        return [path, length]


    def bv_inversion_transport(b_path, index1, index2):
        path = b_path.copy()
        i = path.index(index1)
        j = path.index(index2)
        if i < j:
            temp = []
            for k in range(j - i + 1):
                temp.insert(0, path[k+i])
            c = 0
            for elem in temp:
                path[i+c] = elem
                c += 1
        else:
            temp = []
            for k in range(j - i + 1):
                temp.insert(0, path[k+j])
            c = 0
            for elem in temp:
                path[j+c] = elem
                c += 1

        length = 0
        for k in range(len(path) - 1):
            length += matrix[path[k]][path[k + 1]]
        length += matrix[path[-1]][path[0]]

        return [path, length]


    print("Лучший маршрут после BV-метода: " + str(bv_method(bv_matrix, best_path,
    best_length)[0]) + "\nс длинной: " +
          str(bv_method(bv_matrix, best_path, best_length)[1]))
```