

Федеральное государственное автономное образовательное учреждение  
высшего образования  
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**  
**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**  
**Факультет экономических наук**

**КУРСОВАЯ РАБОТА**  
**Анализ различных постановок и методов решения**  
**задачи размещения объектов**  
по направлению подготовки Экономика  
образовательная программа «Экономика и статистика»

Выполнил:  
Студент группы БСТ182  
Золотарев Антон Олегович

Руководитель:  
доктор физико-математических наук,  
профессор,  
Лепский Александр Евгеньевич

Москва 2020

# ОГЛАВЛЕНИЕ

1 Введение .....	3
2 Описание проблемы .....	3
2.1 Классификация задач по группам свойств .....	3
2.2 Обзор имеющихся постановок задачи.....	9
2.2.1 Поиск точки Ферма .....	9
2.2.2 Задача о геометрической медиане .....	11
2.2.3 Задача Вебера .....	11
2.2.4 Задача о р-медиане .....	13
2.2.5 Задача с препятствиями.....	15
2.2.6 Приближённые методы решения .....	15
3 Решение задач Вебера на реальных данных.....	17
3.1 Размещение дополнительного склада для сети супермаркетов “BILLA” в г. Москва при наличии ограничения на область оптимального размещения объекта.....	18
3.2 Размещение точек автоматической реализации дезинфицирующих средств в районе Щукино г. Москва с учётом наличия весов у жилых домов.....	23
4 Выводы .....	30
5 Список использованной литературы .....	30
6 Приложение .....	32
6.1 Приложение 1. Решение примера задачи о р-медиане в приложении Microsoft Office Excel через надстройку «Поиск решения». .....	32
6.2 Приложение 2. Пример выполнения программы через приближённые методы решения на языке Python в Jupyter Notebook. ....	32
6.3 Приложение 3. Пример выгруженных данных для поиска оптимальных координат размещения дополнительного склада сети супермаркетов “BILLA” .....	33
6.4 Приложение 4. Точки, выступившие в качестве препятствий для задачи оптимального размещения дополнительного склада сети супермаркетов “BILLA” .....	34
6.5 Приложение 5. Решение задачи поиска оптимальной точки размещения дополнительного склада для сети супермаркетов “BILLA”, выполненное на языке Python в Jupyter Notebook.....	35
6.6 Приложение 6. Пример выгруженных данных для поиска оптимальных координат размещения вендинговых автоматов по продаже дезинфицирующих средств в районе Щукино г. Москва .....	38
6.7 Приложение 7. Решение задачи поиска координат оптимальных точек размещения вендинговых аппаратов по продаже дезинфицирующих средств в районе Щукино г. Москва, выполненное на языке Python в Jupyter Notebook .....	39
6.8 Приложение 8. Ссылка на папку в Яндекс.Диске с вспомогательными инструментами для необходимых вычислений по настоящей работе .....	48

## **1 ВВЕДЕНИЕ**

Важность постановки и решения задачи оптимального размещения объектов, с точки зрения экономики, обуславливается необходимостью эффективно разместить точки, минимизируя необходимые затраты на дорогу и максимизируя выгоду от оптимального расположения. Вообще говоря, эта задача представляет огромный класс более конкретных проблем, имеющих приложение как в логистике, экономике и урбанистике, так и в задачах классификации, кластеризации и машинного обучения. Именно поэтому данная тема особенно интересна и, начиная с XVII века, многие ведущие математики своего времени предлагали разные методы решения представленной задачи, внедряя и используя разные ограничения и особенности для заданного пространства возможных и имеющихся точек.

Непосредственно задача оптимального размещения объектов не сформулирована в общем виде ввиду всеобъемлющего характера данной темы. Поэтому в нашем исследовании следует сначала выделить группы свойств, которыми могут обладать предметы данной задачи, а затем рассмотреть несколько конкретных случаев.

## **2 ОПИСАНИЕ ПРОБЛЕМЫ**

### **2.1 Классификация задач по группам свойств**

Исходя из специфики конкретных проблем, решаемых данной задачей, следует выделить следующую классификацию подтипов задач, имеющих прикладное применение в жизни:

1. Тип объектов, между которыми производится оптимизация.

а. Однородные объекты. В данном случае рассматривается необходимость размещения похожих по своему функционалу и задачам объектов, которые должны покрывать заданную область с

определенной интенсивностью (например, сети ресторанов быстрого питания в определённом районе города)

- b. Неоднородные объекты. Здесь во внимание берётся расположение объектов, обладающих разными функциями. Такая задача может описывать внутренний процесс создания сложных конечных товаров фирмой (например, в автомобилестроении процесс изготовления запчастей и их сборки часто происходит на разных предприятиях)

## 2. Методика вычисления.

Существование данного свойства задачи определяется множеством метрик, делающих возможным вычисление расстояния на плоскости  $\mathbb{R}^2$ . Метрическим пространством принято называть пару  $(X, d)$ , в которой  $X$  – множество,  $d$  – числовая функция, такая, что выполняются следующие свойства:

$$\begin{cases} 1. d(x, y) = 0 \Leftrightarrow x = y, \\ 2. d(x, y) = d(y, x), & \text{где } d(\cdot) \geq 0 \forall x, y, z \in X. \\ 3. d(x, z) \leq d(x, y) + d(y, z), \end{cases}$$

На практике, чаще всего используются  $l_p$ -метрики, которая задаётся следующим образом:

$$(\sum_{n=1}^N |x_n - y_n|^p)^{(1/p)}.$$

Теперь перейдём к рассмотрению основных методов вычисления расстояния в задаче оптимального размещения объектов [8, с. 276-278]:

- a. Евклидово расстояние или  $l_2$ -метрика, в которой расстояние возвращает длину отрезка и определяется так:

$$d_E(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2};$$

- b. Манхэттенское расстояние или  $l_1$ -метрика, особенность которой заключается в отсутствии возможности составления путей по диагонали, в  $\mathbb{R}^2$  можно передвигаться только по ОХ или только по ОУ за одну итерацию. Определяется следующим образом:

$$\|x-y\|_1 = |x_1 - y_1| + |x_2 - y_2|;$$

- c. расстояние Чебышёва или  $l_\infty$ -метрика, в которой за расстояние между предметами берётся максимальное значение по одной из осей пространства возможного расположения объектов.

$$\|x-y\|_\infty = \max\{|x_1-y_1|, |x_2-y_2|\},$$

- d. Французская железнодорожная метрика. Данная метрика не является  $l_p$ -метрикой и используется лишь в случае существования центрального объекта, являющегося промежуточным пунктом между всеми дорогами пространства. Оно основано на умозрительном примере железной дороги во Франции, где все пути проходили через Париж (' $p$ ' в уравнении ниже) и не существовало кольцевых путей, связывающих окраины государства. Таким образом, расстояние между двумя городами 'x' и 'y' в данном случае будет описано так:

$$\rho(x, y) = \begin{cases} \|x - y\|, & x - p = \lambda(y - p) \\ \|x - p\| + \|y - p\|, & x - p \neq \lambda(y - p) \end{cases}, \quad x, y \in X, \lambda \in \mathbb{R}.$$

### 3. Характер измерения расстояния

- a. Дискретный случай. Множество возможных точек размещения в настоящей задаче является дискретным лишь тогда, когда все точки пространства можно однозначно отделить друг от друга, то есть количество возможных способов расположения объекта ограничено. Большинство рассматриваемых задач по данной теме разбирают именно дискретное расстояние между объектами ввиду возможности приближённого численного решения путём применения ЭВМ при решении задачи и пренебрежимо малой погрешности при асимптотическом стремлении непрерывного расстояния к дискретным значениям. Также, в этом случае область может рассматриваться в виде графа, инструментарий для анализа которого позволяет оперативно найти эффективное размещение объектов.
- b. Непрерывный случай. В случае, если множество возможных размещений объекта не является счётным, существует бесконечное количество возможных решений задач, поскольку варианты (X, Y) и

$(X+\varepsilon, Y+\varepsilon)$  могут считаться одинаково оптимальными решениями, имеющими равную итоговую полезность.

#### 4. Характер пространства размещения объектов

a. Отсутствие весов для всех точек пространства. В этом случае всё пространство  $\mathbb{R}^2$  обладает одинаковой условной важностью для решения задачи ввиду равномерного распределения плотности двухмерного пространства. Иными словами, если решается задача размещения предприятия обслуживания жителей города, делается допущение об одинаковом количестве проживающих в каждой точке города (что, конечно же, неверно). Формально это допущение можно изобразить так:

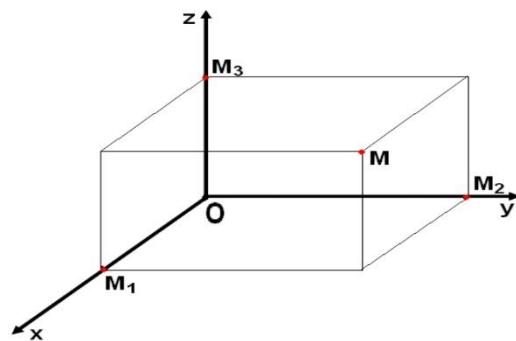


Рисунок 1, Графическое представление объектов при отсутствии весов

Здесь ось ОХ и ОУ представляют собой координаты расположения области или города, ось ОZ отражает плотность распределения граждан по городу, точка О – наиболее юго-западная часть города,  $M_1$  и  $M_2$  представляют юго-восточную и северо-западную части города, а  $M_3$  представляет собой отношение постоянного населения города к его площади в квадратных метрах. Таким образом, область, покрываемая объектом, будет представляться так:

$$F(a_i) = M_3 S(a_i),$$

где  $i$  – индекс оптимизируемого объекта,  $a_i$  – характеристики  $i$ -го объекта,  $S(a_i)$  – площадь, покрываемая  $i$ -ым объектом.

b. Разбиение пространства на зоны, имеющие разную плотность.

Предыдущее допущение содержит в себе очень значительную погрешность, в экономике нередко приводящую к многократному снижению эффективности решения задачи оптимального размещения, поэтому при решении практических кейсов часто применяется взвешивание каждого участка пространства  $\mathbb{R}^2$ .

В случае дискретного пространства область будет представлять собой совокупность вытянутых вверх параллелепипедов, объём которых будет представлять насыщенность взятого сектора, способствующая оптимизации предлагаемого решения. Тогда покрываемая  $i$ -ым объектом область пространства будет записана так:

$$F(a_i) = \sum_{j=1}^k w_j I(j \in S(a_i)),$$

где  $w_j$  – вес  $j$ -го сектора из всех  $k$  секторов, покрываемых  $i$ -ым объектом, а  $I$  – индикаторная функция, принимающая значение ‘1’ в случае принадлежности сектора  $j$  площади, покрываемой объектом и принимающая ‘0’ в противном случае. Очевидно, что в таком случае расстояние Чебышёва является наиболее пригодным для качественного измерения площади, покрываемой  $i$ -ым объектом.

В случае непрерывного пространства пример области выглядит следующим образом:

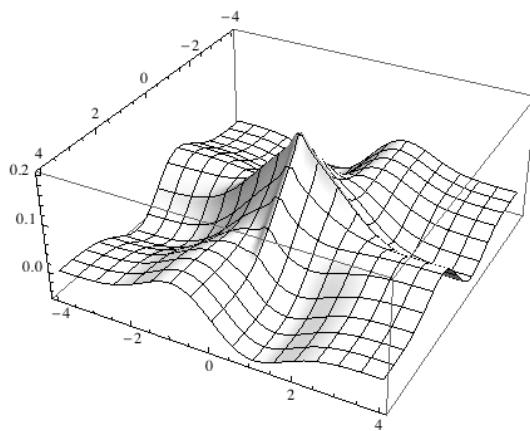


Рисунок 2, Графическое представление объектов при наличии весов

В этом случае область пространства, покрываемая  $i$ -ым объектом, будет выражена так:

$$F(a_i) = \iint_{S(a_i)} f(x, y) dx dy,$$

где  $f(x,y)$  отражает плотность распределения, на которую будет воздействовать  $i$ -й объект. При этом сама плотность может задаваться кусочно с некоторой долей погрешности относительно действительности ввиду возможной сложности и разнородности исходного пространства  $\mathbb{R}^2$ .

5. Характер дороги. Данный параметр предполагается лишь в случае, когда пространство  $\mathbb{R}^2$  может быть представлено в виде неориентированного графа, дорогам которого могут быть приданы веса сложности их прохождения. В противном же случае априори предполагается одинаковая сложность всех возможных дорог. Необходимо отметить, что длина пути учитывается в обоих вариантах данного свойства. Возможны два случая:

a. Одинаковый удельный вес дорог между всеми объектами. Отсутствие весов дорог не ухудшает результаты решения задачи в том и только в том случае, если расстояние между объектами с минимальной долей погрешности можно представить в виде однородных дорог без существования каких-либо препятствий, энергозатратных подъёмов и спусков.

b. Наличие весов у рёбер графа. В общем случае наличие весов у каждой из дорог помогает выдать более точный результат для решения задачи, поскольку имеет место неоднородность путей, порождающая необходимость дифференцирования дороги, содержащую в себе плеяду естественных природных преград, которые необходимо учесть и выразить в длине прямой дороги по асфальту.

6. Характер однородности пространства. Как и предыдущий параметр, однородность пространства может не быть учтена при решении задачи оптимального размещения объектов, однако в ряде известных задач это весьма значительно влияет на эффективность решения. Возможны два случая:

a. Наличие только размещаемых объектов и дорог между ними. В данном случае предполагается, что в пространстве  $\mathbb{R}^2$  площадь объектов пренебрежимо мала относительно расстояний между самими

объектами, что позволяет прокладывать дороги сквозь имеющиеся объекты, а также отсутствуют какие бы то ни было препятствия (естественные – болота, леса, реки и другие; созданные человеком – стадионы, вокзалы и другие) для прокладывания дорог.

b. Существование препятствий между объектами, вынуждающих изменять траекторию дороги. Если же на пространстве  $\mathbb{R}^2$  существует хотя бы одно из описанных выше препятствий или расстояние между объектами не является большим настолько, чтобы площадью объектов при построении оптимальных маршрутов можно было пренебречь, решение задачи обуславливается учётом данных условий и изменяется в зависимости от характеристики местности.

Очевидно, что если учитывать каждый возможный набор описанных свойств при постановке задачи, то итоговая совокупность конкретных случаев составит огромное количество различных вариаций, поэтому имеет смысл разобрать лишь наиболее важные и используемые в оптимизации процессов типы рассматриваемой проблемы, а также их решения.

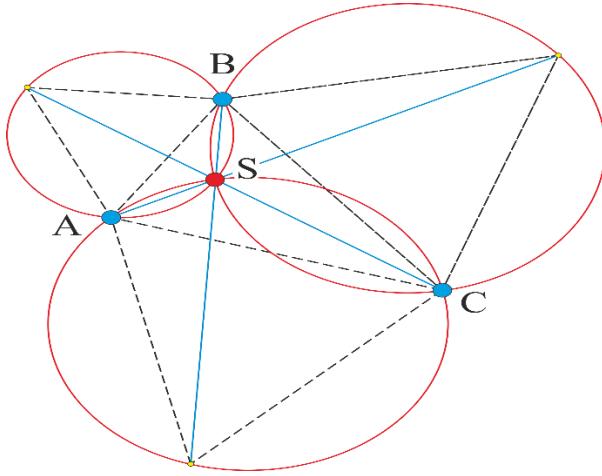
## **2.2 Обзор имеющихся постановок задачи**

### **2.2.1 Поиск точки Ферма**

Впервые математическая постановка данной задачи была представлена Пьером де Ферма в 1679 году, где было необходимо найти для трёх точек такую четвёртую, чтобы суммарное расстояние от исходных точек до неё было минимальным. Ниже приведена формулировка алгоритма нахождения данной точки без доказательства:

Пусть имеется произвольный треугольник с вершинами А, В, С. Тогда, если построить за пределами треугольника вершины А', В', С' такие, что  $AB=AC'=BC'$ ,  $BC=BA'=CA'$ ,  $AC=AB'=CB'$  (то есть три внешних равносторонних треугольника), то точка пересечения прямых  $AA'$ ,  $BB'$  и  $CC'$  и описанных вокруг трёх равносторонних треугольников  $ABC'$ ,  $A'BC$ ,  $AB'C$  окружностей будет являться

точкой Ферма, суммарное расстояние от которой до трёх исходных вершин треугольника будет минимальным. В случае, если треугольник имеет угол больше  $120^\circ$ , то вершина тупого угла будет являться точкой Ферма.



*Рисунок 3, Графическое представление поиска точки ферма. Источник: wikipedia.org*

На практике алгоритм поиска точки Ферма может быть несколько упрощён:

1. Проверить, есть ли в треугольнике угол больше  $120^\circ$ .
2. Если его нет, то построить один внешний равносторонний треугольник от любой из сторон.
3. Описать вокруг него окружность.
4. Провести прямую между вершиной, не принадлежащей внешнему равностороннему треугольнику, и вершиной, порождённой при построении внешнего равностороннего треугольника ( $AA'$ ,  $BB'$  или  $CC'$  из исходной постановки задачи).
5. Точка пересечения описанной окружности и проведённой прямой будет являться точкой Ферма.

Ввиду очевидной простоты данной задачи и ограниченности рассматриваемых точек, по отношению к которым необходимо минимизировать суммарное расстояние, поиск точки Ферма не стал конечной стадией развития задачи оптимального размещения объектов, но дал возможность для развития большого количества более комплексных задач, рассмотренных далее.

## 2.2.2 Задача о геометрической медиане

В соответствии с изложением данной задачи П.А. Пановым [3, 4, 5], задача о геометрической медиане представляет из себя случай поиска минимальной суммы на всем рассматриваемом пространстве для бесконечно возможного количества точек. Формально геометрическая медиана на плоскости для конечного множества  $S$  определяется так:

$$m(S) = \arg \min_{y \in \mathbb{R}^2} \sum_{x \in S} \|x - y\|.$$

Здесь  $y$  – координаты геометрической медианы, а  $x$  – все значения исходных точек на рассматриваемой области.

Графически задача геометрической медианы отображается следующим образом:

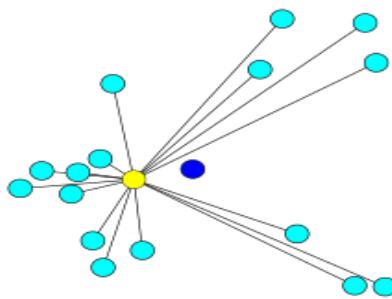


Рисунок 4, Графическое представление решения задачи о геометрической медиане

В непрерывном случае для ограниченной области с кусочно-дифференцируемыми границами определение геометрической медианы таково:

$$m(S) = \arg \min_{y \in \mathbb{R}^2} \int_{x \in \Omega} \|x - y\| d\Omega.$$

Данная функция, ввиду своей дифференцируемости и выпуклости, будет являться единственным решением градиентного уравнения  $\nabla \int_{x \in \Omega} \|x - y\| d\Omega = 0$ .

## 2.2.3 Задача Вебера

Обобщая поиск точки Ферма и геометрической медианы, задача Вебера, названная в честь немецкого экономиста начала XX века, способствует

эффективному расположению фабрик и предприятий при существовании исходных точек хранения ресурсов. На самом деле, под данным наименованием скрывается целый класс экономических задач, содержащий в себе как случай трёх, так и большего количества исходных точек хранения, а также разновидности задач притяжения и отталкивания, в которых исходные точки могут иметь положительные или отрицательные условные веса полезности. Более того, существует два типа задачи Вебера, покрывающие первое свойство приведённой выше классификации постановок задач (размещение однородных объектов (Multisource Weber Problem) или нескольких взаимосвязанных, но разных по функционалу предприятий (Multifacility Weber Problem) на пространстве  $\mathbb{R}^2$ ). Формальная постановка конкретной задачи выглядит так [6, с. 19-26]: «Пусть имеется пространство  $L = \{x: x \in \mathbb{R}^2\}$ ,  $p$  - число предприятий, которые необходимо разместить на  $L$ ,  $m$  – число клиентов, зафиксированных точками на  $L$ ,  $w_j$  – стоимость обслуживания клиента  $j$ ,  $X_i=(x_i, y_i)$ ,  $X_i \in L$  – координаты размещения предприятия  $i$ ,  $i=\overrightarrow{1, p}$ ,  $A_j=(a_j, b_j)$   $A_j \in L$  – координаты размещения клиента  $j$ ,  $j=\overrightarrow{1, m}$ .

Пусть  $d(X_i, z_{ij})$  - расстояние между предприятием  $i$  и клиентом  $j$ , а  $z_{ij} \in \{0, 1\}$  – логическая переменная, свидетельствующая об обслуживании клиента  $i$  предприятием  $j$ ».

Тогда целевая функция будет сформулирована следующим образом:

$$F(X_i, z_{ij}) = \sum_{i=1}^p \sum_{j=1}^m z_{ij} w_j d(X_i, A_j) \rightarrow \min_{X, Z}$$

$$\text{s.t. } \begin{cases} \sum_{i=1}^p z_{ij} = 1, j = \overrightarrow{1, m} \\ z_{ij} \in \{0, 1\}, j = \overrightarrow{1, m}, i = \overrightarrow{1, p} \\ X_i \in L, i = \overrightarrow{1, p} \end{cases}$$

Разберём пример решения задачи такого типа: пусть в определённом районе размером  $1000*1000$  м<sup>2</sup> необходимо разместить ресторан быстрого обслуживания, при этом в этом районе живёт 1000 человек, которые можно разделить на два класса: 400 человек, от которых ожидается регулярное ежедневное посещение будущего ресторана, и 600 человек, которые могут зайти в данный ресторан в среднем один раз в 10 дней. При этом жители города проживают в 10 домах,

каждый из которых вмещает по 100 человек, строго принадлежащих первому или второму классу жителей. На графике, представленном ниже, дома A, B, C, D являются собственностью горожан, в будущем регулярно посещающих ресторан, тогда как E, F, G, H, I, J принадлежат горожанам, в будущем редко заходящим в необходимое для размещения заведение.

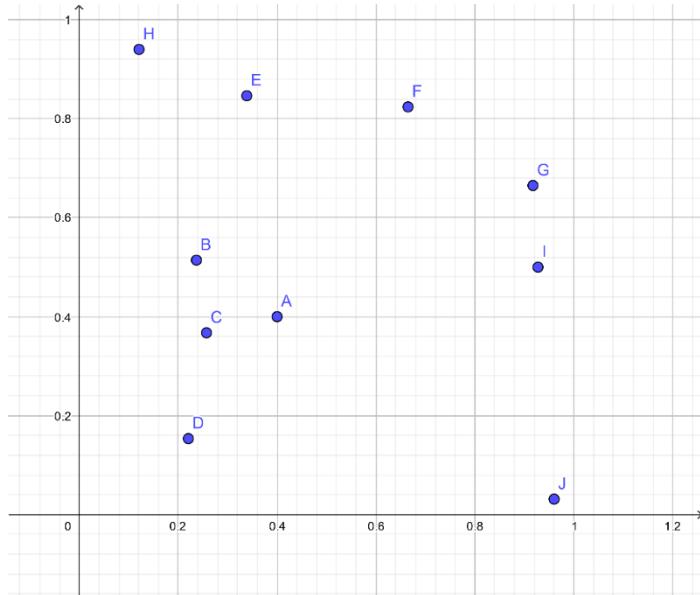


Рисунок 5, Пример постановки задачи Вебера. Источник: [math10.com](http://math10.com)

В данном случае обозначенные переменные примут следующие значения:  $L = \{x: x \in 1000 \times 1000\}$ ,  $p = 1$ ,  $m = 1000$ ,  $w_{j1} = 1$ ,  $w_{j2} = 0,1$  (веса клиентов в данном случае отражают количество посещений в день); где  $A_j, j = 1, \dots, 400$  – клиенты 1-го типа,  $B_j, j = 1, \dots, 600$  – клиенты 2-го типа;  $z_{ij} = 1$  для каждого клиента, поскольку

в данном случае будет единственное предприятие.

Задача будет поставлена так:

$$F(X) = \sum_{j=1}^{400} d(X, A_j) + \sum_{j=1}^{600} 0,1d(X, B_j) \rightarrow \min_X \\ \text{s.t. } X \in L$$

Таким образом, поставленная задача может быть решена через приближённые методы решения, которые будут рассмотрены в главе [2.2.6](#) настоящей работы, поскольку здесь мы имеем дело с размещением оптимальной точки, минимизирующей сумму расстояний для более чем трёх объектов.

#### 2.2.4 Задача о $p$ -медиане

Дискретная задача Вебера, в которой вместо плоскости предприятия должны располагаться на конечном множестве точек. В дополнение к имеющимся

ограничениям задачи Вебера, вводится множество точек  $P$ ,  $|P|=n$ ,  $P \subset L$ , а также булева переменная  $t_k \in \{0,1\}$ ,  $k \in P$ , равная единице в случае, если в точке  $k$  было размещено предприятие.

Задача о  $p$ -медиане представляет из себя весьма практический-ориентированный случай, примером её применения является размещение предприятий бытового обслуживания, складов, пунктов заправок на дорогах. Решения задачи о  $p$ -медиане большой размерности может быть найдено через GRASP эвристики [12], метод ветвей, отсечений и оценок [13]. В данной задаче можно рассмотреть область в качестве графа.

Приведём простой пример реализации данной задачи: пусть дана область размерностью  $10 \times 10$  с шестью объектами, размещёнными в точках, значения которых принимают только целые значения. Чтобы найти наиболее оптимальную точку размещения, сумма расстояний от исходных точек до которой будет наименьшей, следует использовать манхэттеновское расстояние ввиду дискретности пространства и считать расстояние как количество пройденных узлов

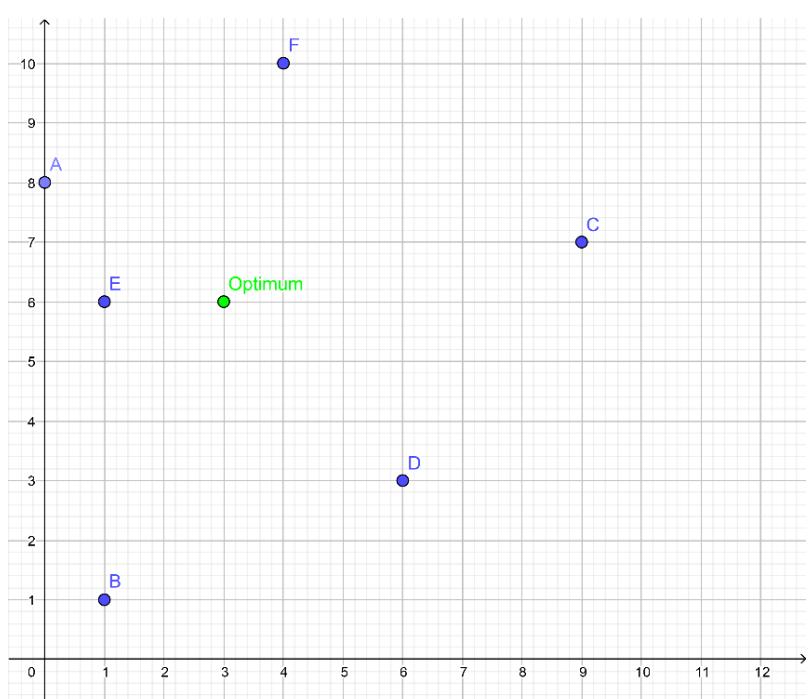


Рисунок 6. Пример решения задачи о  $p$ -медиане. Источник: [math10.com](http://math10.com)

с целыми значениями координат X и Y до искомой точки, включая её саму. Оптимальная точка в примере ниже располагается в узле (3; 6), и расстояние между ней и точками A, B, C, D, E, F будет соответственно 5, 7, 7, 6, 2 и 5. Решение данной задачи было найдено через симплекс-метод для задач линейного программирования и представлено в [Приложении 1](#).

## 2.2.5 Задача с препятствиями

Одним из подклассов задач Вебера, наиболее применяемых в жизни, является её частный случай с препятствиями, когда разные преграды мешают наискорейшим образом добраться до необходимой точки. В таком случае дополнительно вводится следующее условие:  $X_j \notin R_z$ , где  $R_z$  – множество недоступных точек.

Графически пример задачи с препятствиями может быть представлен следующим образом, где необходимо разместить объект, сумма расстояний до которого минимальна с учётом расположения устья реки на множестве доступных точек размещения.

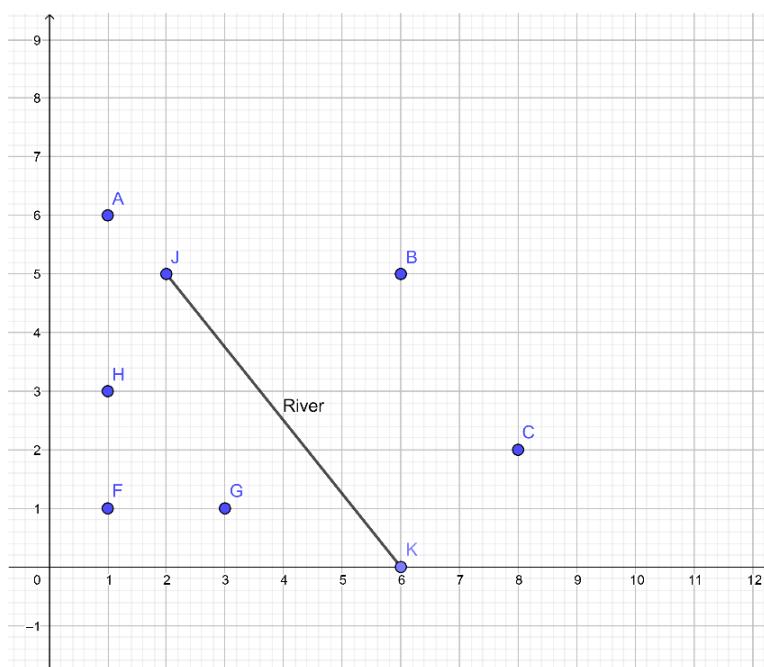


Рисунок 7. Пример решения задачи с препятствиями. Источник: [math10.com](http://math10.com)

В таком случае поиск оптимальной точки размещения сводится к разбиению как минимум двух путей от исходных точек до неё через сумму евклидовых расстояний, которые очевидным образом будут пересекаться друг с другом в точке  $J$  в приведённом примере. Допустим, оптимальной точкой здесь будет  $Z = (2,2)$ , тогда путь до искомого объекта из точек  $A$ ,

$F, G, H$  будет традиционно задан обычным отрезком, в то время как путь из точек  $B$  и  $C$  будет задаваться как сумма отрезков  $BJ+JZ$  и  $CJ+JZ$  соответственно.

## 2.2.6 Приближённые методы решения

В случае, когда описанные выше задачи должны найти оптимальное размещение объекта, путь к которым необходимо проложить от более чем трёх точек, решить задачу геометрически становится невозможно, в результате чего приходится применять приближённые методы решения, итеративно решающие

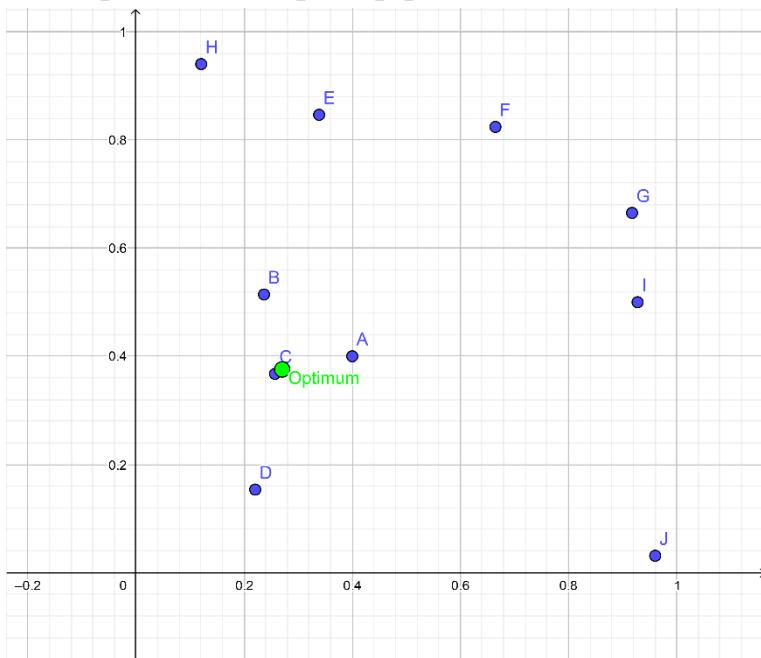
поставленную задачу. Эта задача может быть решена рекурсивно, когда для каждого отдельно взятого обслуживающего предприятия стартовая точка размещения берётся случайно и методом градиентного спуска запускается поиск позиции, минимизирующей затраты на время в пути. Формально координаты точки оптимального размещения находятся следующим образом:

$$P_{opt} = \left( \frac{P_1}{|PP_1|} + \cdots + \frac{P_m}{|PP_m|} \right) / \left( \frac{1}{|PP_1|} + \cdots + \frac{1}{|PP_m|} \right)$$

Данная система уравнений решается методом простой итерации, когда берётся случайная начальная точка и затем осуществляется поиск локального минимума заданной функции, по итогам определённого количества запусков обеспечивающей оптимальное размещение данной точки. Сам метод записывается следующим образом:

$$\begin{cases} P(x_{i+1}) = P_{opt}(x_i), \\ P(y_{i+1}) = P_{opt}(y_i). \end{cases}$$

Представим пример решения задачи, постановка которой приведена в пункте



[2.2.3](#) настоящей работы: для района с населением 1000 жителей, проживающих в 10 разных домах и имеющих две разных ожидаемых интенсивности посещения ресторана необходимо найти оптимальное размещение этого ресторана, дабы максимизировать эффективность

времяпрепровождения этих жителей.

Чтобы решить данную задачу, запишем имеющиеся координаты в явном виде:

1.  $A = (0.4, 0.4);$
2.  $B = (0.24; 0.51);$
3.  $C = (0.26, 0.37);$

Рисунок 8. Пример решения задачи Вебера через приближённые методы решения. Источник: [math10.com](http://math10.com)

4.  $D = (0.22, 0.15);$
5.  $E = (0.34, 0.85);$

6.  $F = (0.66, 0.82)$ ;
7.  $G = (0.92, 0.67)$ ;
8.  $H = (0.12, 0.94)$ ;
9.  $I = (0.93, 0.5)$ ;
10.  $J = (0.96, 0.03)$ .

Теперь становится возможной запись начальных условий для написанной на языке Python в Jupyter Notebook программы (см. [Приложение 2](#)), по результатам работы которой координаты оптимального размещения ресторана принимают следующие значения на каждой из итераций (итоговое оптимальное значение –  $(0.270, 0.376)$ , находится в непосредственной близости с домом C):

```
In [9]: P_opt(P_start, main_mas, 30, weights)
```

```
[0. 0.]
[0.2885071  0.33376068]
[0.28720667  0.37360545]
[0.28182748  0.38010371]
[0.27927367  0.38037442]
[0.2777181   0.37976507]
[0.27656249  0.3791346 ]
[0.27563936  0.37860011]
[0.27488182  0.37815775]
[0.27425106  0.37778988]
[0.27372024  0.3774812 ]
[0.27326959  0.37721991]
[0.27288412  0.37699704]
[0.27255227  0.37680565]
[0.27226497  0.37664034]
[0.27201503  0.37649681]
[0.27179664  0.37637164]
[0.27160511  0.37626204]
[0.27143657  0.37616574]
[0.27128783  0.37608086]
[0.27115622  0.37600585]
[0.2710395   0.37593939]
[0.27093577  0.37588038]
[0.27084341  0.37582789]
[0.27076104  0.37578111]
[0.27068748  0.37573937]
[0.2706217   0.37570205]
[0.2705628   0.37566867]
[0.27051002  0.37563876]
[0.27046266  0.37561194]
```

*Рисунок 9, Координаты оптимальной точки для примера задачи Вебера на каждой из итераций. Источник: Jupyter Notebook*

### 3 РЕШЕНИЕ ЗАДАЧ ВЕБЕРА НА РЕАЛЬНЫХ ДАННЫХ

Для решения задачи оптимального размещения объектов на практике необходимо решить два ключевых вопроса: каково оптимальное количество необходимых для размещения объектов и где именно они должны располагаться.

Приведём примеры решения двух практических задач, основанных на реальных данных. В первой из них оптимальное количество складов определено заранее, однако задача осложняется наличием атипичного препятствия для размещения. Во второй задаче представлена попытка решения задачи об оптимальном количестве необходимых объектов, а также у каждой точки из

множества, по отношению к которому необходимо разместить объекты, имеются свои веса.

### **3.1 Размещение дополнительного склада для сети супермаркетов “BILLA” в г. Москва при наличии ограничения на область оптимального размещения объекта.**

В практической части своей работы я решил рассмотреть следующую задачу: пусть необходимо разместить дополнительный склад для промежуточного хранения продуктов питания между заводами партнёров-производителей и супермаркетами “BILLA” в Москве. Таким образом, необходимо решить задачу Вебера, в которой предприятием становится склад, а клиентами – непосредственно супермаркеты сети “BILLA”.

Данная задача была выбрана по нескольким причинам:

- ввиду логистических сложностей большие компании часто ограничиваются минимально возможным количеством складов, что накладывает естественное ограничение на количество дополнительно размещаемых складов в размере одного;
- структура каждого отдельно взятого супермаркета BILLA такова, что все они примерно одинаковы по своей площади и интенсивности посещаемости покупателями, поэтому в данном случае становится возможным сделать предпосылку о равном весе каждого из супермаркетов;
- особый интерес представляет тот факт, что склад является огромным помещением, которое с экономической точки зрения не выгодно ни покупать, ни арендовать в пределах МКАД, поскольку любая совокупность переменных затрат на доставку товаров не будет дороже цены земли таких масштабов в столице Российской Федерации даже в долгосрочном периоде ввиду неликвидности необходимого вложения. Поэтому в данной задаче в целях сохранения здравого смысла будет сделано ограничение об отсутствии возможности разместить склад в

пределах МКАД и на расстоянии менее чем 10 километров от московской кольцевой автодороги за её пределами.

Перейдём к постановке задачи: нам необходимо определить оптимальные координаты размещения единственного дополнительного склада BILLA, находящегося на расстоянии не менее 10 километров от МКАД за её пределами, при этом чтобы сумма расстояний до имеющихся в Москве супермаркетов BILLA была минимальна. Формально, необходимо решить непрерывную задачу Вебера, которая выглядит так:

$$F(X) = \sum_{j=1}^m d(X, A_j) \rightarrow \min_X$$

s.t.  $X \notin \{\text{внутри МКАД}\} \cup \{\text{менее 10 километров за пределами МКАД}\}$

После нанесения всех необходимых точек супермаркетов в конструкторе «Яндекс.Карт», стала возможной выгрузка их координат в отдельный файл(пример выгруженных данных представлен в [Приложении 3](#)), данная информация является наиболее важной при решении задач оптимального размещения.

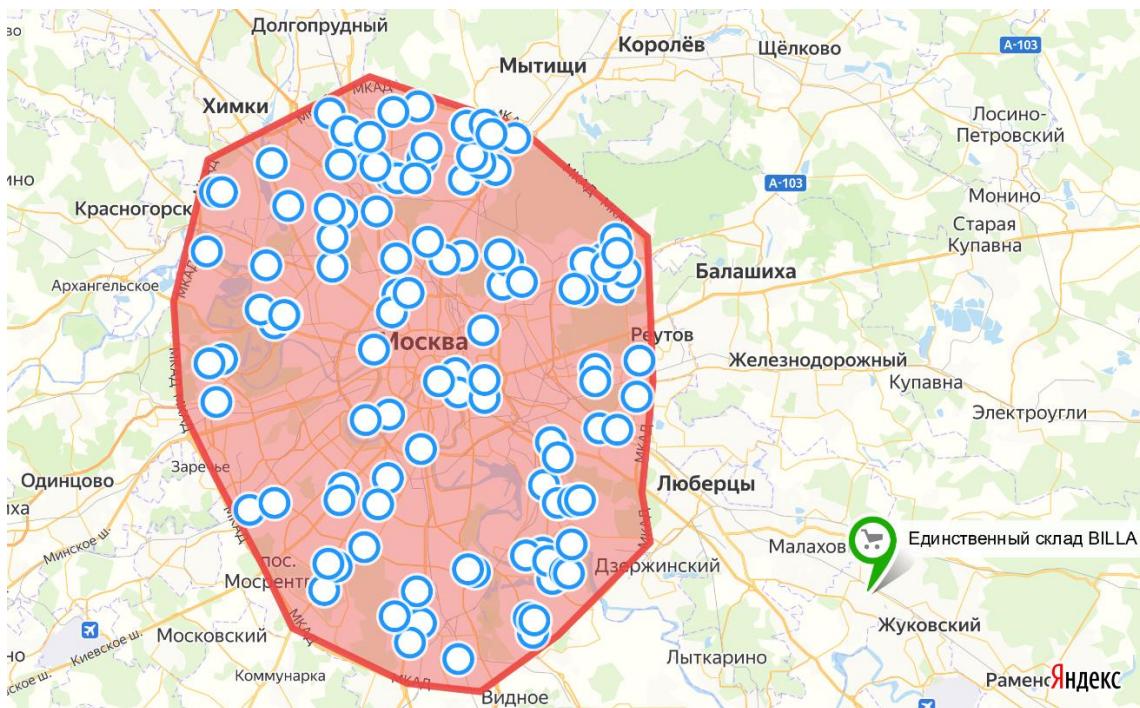


Рисунок 10, Представление расположения супермаркетов BILLA в пределах МКАД. Источник: конструктор Карт «Яндекса»

Также было решено добавить на карту ограничения в виде совокупности точек, каждая из которых находится на расстоянии 10 километров от МКАД, поскольку для ограничений в виде многоугольников или областей невозможно

выгрузить геоданные через представленную платформу. Именно поэтому дальнейшее решение поставленной задачи упёрлось в анализ точечных ограничений, координаты которых представлены в [Приложении 4](#).

Для решения самой задачи было принято использовать евклидову метрику, поскольку манхэттеновское расстояние уместно использовать лишь в случае такой структуры города, при которой все улицы стоят строго перпендикулярно друг другу, что совершенно неприменимо в случае Москвы, имеющей кольцевое строение.

Поскольку в представленной задаче необходимо найти минимальную сумму расстояний до более, чем трёх точек, было решено использовать приближённые методы решения, дополнив алгоритм программы, представленной в Приложении 2. В первую очередь, она была дополнена обработкой имеющихся точечных ограничений, а также проверкой на соответствие приближающегося к оптимальному значения этим ограничениям.

Необходимо отметить, что точечные ограничения было решено поделить на 8 классов в соответствии с расположением промежуточной оптимальной точки относительно наиболее южной, северной, западной и восточной точек размещения супермаркетов «BILLA». Например, если предполагаемый оптимум располагался бы левее самой западной и выше самой северной точек расположения супермаркетов внутри МКАД, то в расчёт бралась предпосылка о том, что этот оптимум должен быть также одновременно левее и выше координат наиболее близкого к ним ограничения. В случае, если данное условие выполнялось, то на выход на данной итерации отправлялось предполагаемое оптимальное значение без изменений, иначе оптимальное значение было бы скорректировано в сторону приближения к границе допустимых значений. Более подробный алгоритм обработки ограничений вкупе с полным кодом для решения задачи представлен в [Приложении 5](#).

На практике, по причине очень большой ограничительной области как таковой, решение задачи свелось к поиску геометрической медианы для совокупности супермаркетов в пределах МКАД, после чего был запущен поиск

ближайшей к медиане ограничительной точки, выдаваемой программой за оптимальное решение задачи. Представим значения координат по ходу первых 30 итераций:

```
In [16]: def P_opt(P_start, main_mas, num_steps=20, weights=[1]*len(main_mas), limitations=None, pre_limits=None):
    candidates = []
    P_opt = np.zeros([num_steps+1, 2])
    P_opt[0] = P_start
    for i in range(num_steps):
        P_opt[i+1] = [sum_of_nominator_x(P_opt[i], main_mas, weights)/sum_of_denominator(P_opt[i], main_mas, weights),
                      sum_of_nominator_y(P_opt[i], main_mas, weights)/sum_of_denominator(P_opt[i], main_mas, weights)]
        if limitations!=None and pre_limits!=None and i%30==0 and i>0:
            P_opt[i+1] = limitations.handling(P_opt[i+1], limitations, pre_limits, out_of_limitation)
            print('Следующий элемент является геометрической медианой для данного множества.')
            print('Элемент после следующего является наиболее близким допустимым значением к геометрической медиане')
            optimal_point=list(P_opt[i+1])
            print(P_opt[i])
    return optimal_point

In [17]: optimal_point = P_opt(P_start, main_mas, num_steps=300, limitations=limitations, pre_limits=pre_limits)

[0. 0.]
[55.76287556 37.63151642]
[55.76663833 37.6338289]
[55.76861942 37.63435128]
[55.76970111 37.63433074]
[55.77030762 37.63418673]
[55.77065488 37.63404756]
[55.77085689 37.63394265]
[55.7709758 37.63387094]
[55.77104639 37.63382427]
[55.77108853 37.63379473]
[55.77111379 37.63377634]
[55.77112897 37.63376502]
[55.77113812 37.63375809]
[55.77114363 37.63375387]
[55.77114696 37.6337513]
[55.77114897 37.63374975]
[55.77115018 37.63374881]

[55.77115091 37.63374823]
[55.77115135 37.63374789]
[55.77115162 37.63374768]
[55.77115178 37.63374755]
[55.77115188 37.63374748]
[55.77115194 37.63374743]
[55.77115198 37.6337474]
[55.77115152 37.63374739]
[55.77115201 37.63374738]
[55.77115202 37.63374737]
[55.77115202 37.63374737]
[55.77115203 37.63374737]
Следующий элемент является геометрической медианой для данного множества.
Элемент после следующего является наиболее близким допустимым значением к геометрической медиане
[55.77115203 37.63374736]
[55.98551489093684 37.70574034604027]
[55.98551489093684 37.70574034604027]

In [18]: optimal_point

Out[18]: [55.98551489093684, 37.70574034604027]

In [19]: sum_of_dis_opt = 0
for i in range(len(main_mas)):
    sum_of_dis_opt += vec_distance(main_mas[i], optimal_point)

In [23]: round(sum_of_dis_opt, 2)

Out[23]: 3046.22
```

*Рисунок 11, Решение задачи оптимального размещения дополнительного склада для сети супермаркетов "BILLA". Источник: Jupyter Notebook*

Таким образом, геометрическая медиана находится в непосредственной близости от станции метро «Сухаревская», тогда как наиболее близкая к ней ограничительная точка – это набережная в СНТ Пирогово городского округа Мытищи Московской области. Обработав имеющиеся результаты, я нашёл два больших свободных участка в окрестности данной точки, которые представлены на карте ниже. Первый оптимум находится примерно в километре от деревни Пирогово, тогда как второй несколько нарушает поставленные ограничения и

является частью Пироговского лесопарка. Сумма расстояний до получившейся точки составила около 3046 километров, необходимых для преодоления средствами транспортировки хранящихся на складе товаров.

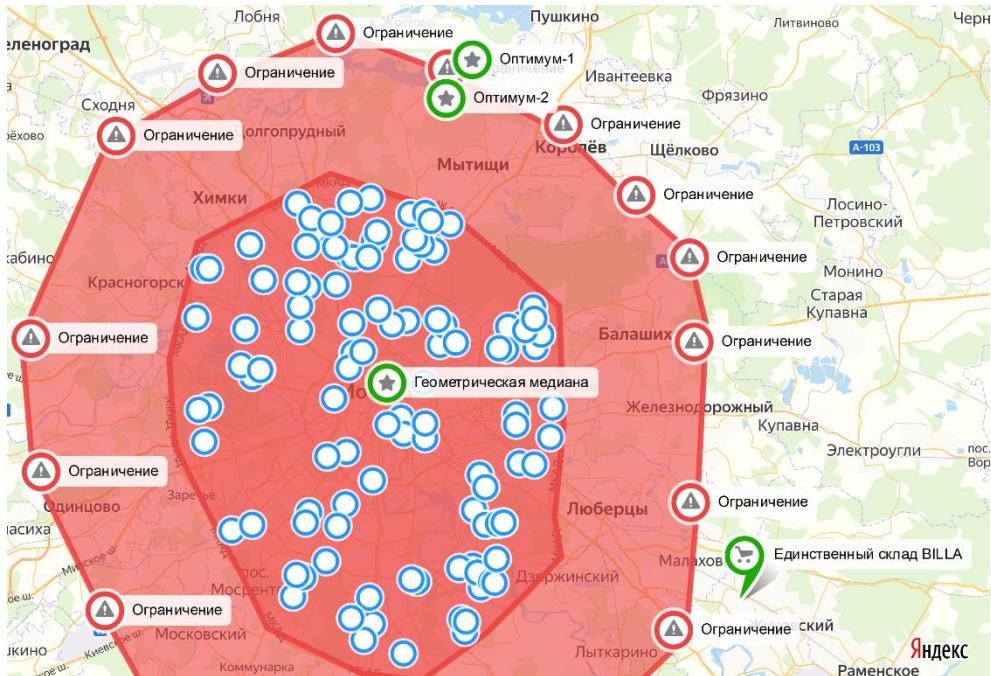


Рисунок 12. Графическое изображение решённой задачи о размещении склада для сети супермаркетов "BILLA".  
Источник: Конструктор карт «Яндекс»

На самом деле, в качестве перспективы исследования кажется весьма полезным попробовать разбить отрезки, прилегающие к выбранной оптимальной ограничительной точке, добавив к ним ещё несколько точек и попытавшись найти там ещё более оптимальное значение суммы расстояний. В свою очередь, верность решения задачи в данной постановке легко проверить с помощью следующего кода:

```

In [18]: optimal_point
Out[18]: [55.98551489093684, 37.70574034604027]

In [19]: sum_of_dis_opt = 0
for i in range(len(main_mas)):
    sum_of_dis_opt += vec_distance(main_mas[i], optimal_point)

In [23]: round(sum_of_dis_opt, 2)
Out[23]: 3046.22

In [21]: sum_of_dis = [0]*len(limitations)
for i in range(len(main_mas)):
    for j in range(len(limitations)):
        sum_of_dis[j] += vec_distance(main_mas[i], limitations[j])

In [25]: for i in sum_of_dis:
    print(round(i, 2))

3508.27
3362.42
3506.57
3445.42
3616.52
3556.94
3528.93
3426.33
3734.17
3963.99
3879.25
3461.15
3256.87
3046.22
3264.49
3469.5
3707.97

```

*Рисунок 13, Проверка верности решения поставленной задачи. Источник: Jupyter Notebook*

### **3.2      Размещение      точек      автоматической      реализации дезинфицирующих средств в районе Щукино г. Москва с учётом наличия весов у жилых домов.**

Поскольку весной 2020 года весь мир был охвачен распространением коронавирусной инфекции и в начале апреля в столице Российской Федерации был объявлен карантин, мне показалось актуальным рассмотреть дополнительную задачу, в которой было бы необходимо разместить несколько точек автоматической реализации дезинфицирующих средств для жителей Москвы.

В данной постановке особый интерес представляет необходимость изучения плотности населения по району, а также возможность почти полностью игнорировать препятствия по размещению объектов и дорог к ним, поскольку на уровне района почти везде есть возможность поставить небольшой автомат по выдаче медицинских масок, антисептиков для рук или других средств первой необходимости в условиях пандемии.

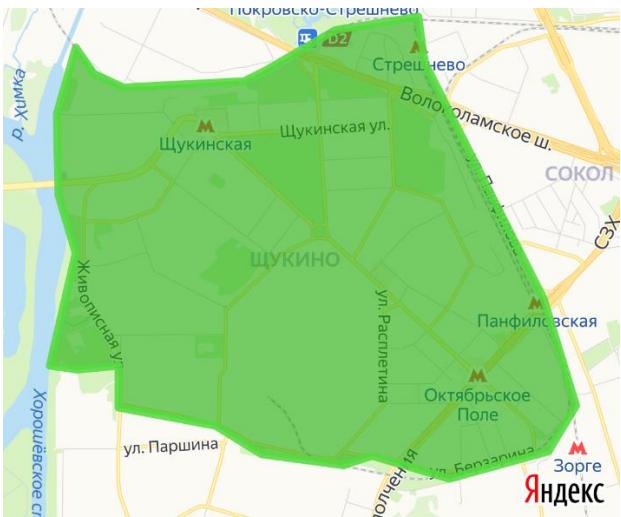


Рисунок 14, Район Щукино на карте Москвы.  
Источник: Конструктор карт "Яндекса"

Для рассмотрения я решил взять район Щукино города Москвы, поскольку в нём нет огромных препятствий для прокладывания дороги по типу стадионов или озёр в центре района. Этот район ограничен на севере железнодорожным путём МЦД-2, на востоке – железнодорожным путём МЦК, на юге – улицей Берзарина, на западе – Хорошёвским спрямлением реки Москвы.

Перейдём к постановке задачи: нам необходимо определить количество необходимых точек реализации средств первой медицинской помощи и их координаты размещения. Чтобы получить наиболее приближенные к реальности решения задачи, было решено добавить на карту все жилые дома района, за веса каждого из них взяв размер жилой площади [11]. Из всех общедоступных характеристик, которые можно найти о многоквартирных домах, именно жилая площадь каждого дома наиболее коррелирована с количеством проживающих там людей, которое и следует принять как основную клиентуру предполагаемых для размещения объектов без каких-либо ограничений на возраст или другие факторы. Таким образом, была получена информация о 305 жилых домах данного района с суммарной жилой площадью в размере около 2,5 миллионов квадратных метров. Пример выгруженных через конструктор Яндекс.Карт данных приведён в Приложении 6.

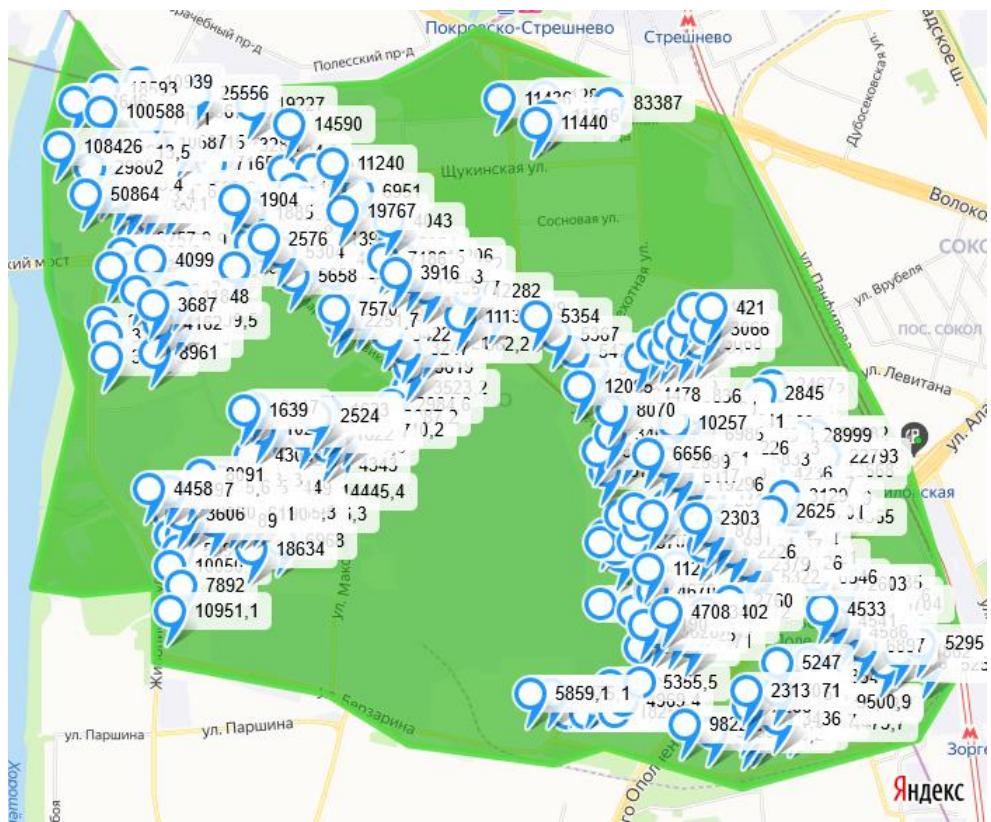


Рисунок 15. Значения жилых площадей для каждого из домов в районе Щукино. Источник: Конструктор карт "Яндекса"

Так, получив веса каждого из домов, стало возможным перейти к следующему этапу решения задачи: определение оптимального количества размещаемых объектов. Чтобы решить данную задачу, необходимо произвести ряд экономических вычислений, основанных на предпосылках об ожидаемом спросе на данный тип реализации медикаментов. Поделив суммарную жилую площадь имеющихся домов на данные о суммарном населении района, получим цифру в примерно 23 квадратных метра на одного жителя района, с помощью которой можно получить примерные данные о населении каждого из жилых домов. В свою очередь, случайно выбранный вендинговый аппарат [«Unicum Foodbox Long»](#) вмещает себя 1536 единиц товара, если разместить в нём 8 уникальных медицинских дезинфицирующих препаратов, то он будет содержать в себе 192 одинаковых экземпляра, рассчитанных на потребление всеми жителями района Щукино. Если предположить, что хотя бы одна десятая часть населения будет пользоваться данными автоматами по выдаче медикаментов, при этом для их полного использования им потребуется месяц, а каждый из вендинговых аппаратов будет пополняться еженедельно, то в результате несложных расчётов получится,

что один аппарат за месяц обслуживает порядка  $\frac{8640}{110272} \approx 0,078$  от всего предъявляющего спрос населения района. Таким образом, для удовлетворения спроса всего района необходимо поставить около 13 вендинговых аппаратов внутри Щукино, каждый из которых будет обслуживать дома с ожидаемым населением в сумме около 8640 человек, среди которых 10% будут активными покупателями предлагаемого товара.

В результате произведённых вычислений стало возможным чётко сформулировать задачу Вебера для данной задачи:

$$F(X_i, z_{ij}) = \sum_{i=1}^{13} \sum_{j=1}^{304} z_{ij} w_j d(X_i, A_j) \rightarrow \min_{X, Z}$$

$$\text{s.t. } \begin{cases} \sum_{i=1}^{13} z_{ij} = 1, j = \overline{1, 304} \\ z_{ij} \in \{0, 1\}, j = \overline{1, 304}, i = \overline{1, 13} \\ X_i \in L, i = \overline{1, 13} \end{cases}$$

Теперь необходимо разделить имеющийся район таким образом, чтобы сумма расстояний до 13 располагаемых объектов была минимальной. Используя метод Kmeans++ [2, стр. 19-26] из библиотеки sklearn, подключаемой к языку программирования Python, произведём кластеризацию объектов, разделив всё пространство на 13 подпространств в соответствии с имеющимися весами каждого из домов (обратим внимание, что за веса можно брать как значения жилой площади, так и ожидаемое население в каждом из домов, поскольку эти величины в данном исследовании абсолютно коррелированы). После разделения района для каждого подпространства была найдена точка, в которой сумма расстояний внутри образовавшегося микрорайона была бы минимальной. Таким образом, были получены координаты для 13 точек (полный алгоритм работы программы приведён в Приложении 7):

```
In [7]: X = main_mas
kmeans = KMeans(n_clusters=13, random_state=0)
kmeans.fit(X, sample_weight=weights).cluster_centers_

Out[7]: array([[55.79697046, 37.49314625],
   [55.80576348, 37.46364915],
   [55.80831116, 37.45005669],
   [55.7948686 , 37.48748762],
   [55.79672708, 37.46192157],
   [55.79047447, 37.495519 ],
   [55.80268601, 37.47084648],
   [55.80289409, 37.45380678],
   [55.79172765, 37.48322024],
   [55.79487295, 37.45619676],
   [55.80957767, 37.48023258],
   [55.80896842, 37.45692319],
   [55.79891441, 37.48326581]])
```

Рисунок 16, Результаты вычисления координат центров 13 кластеров для района Щукино. Источник: Jupyter Notebook

После нанесения полученных координат на карту, была получена следующая картина:

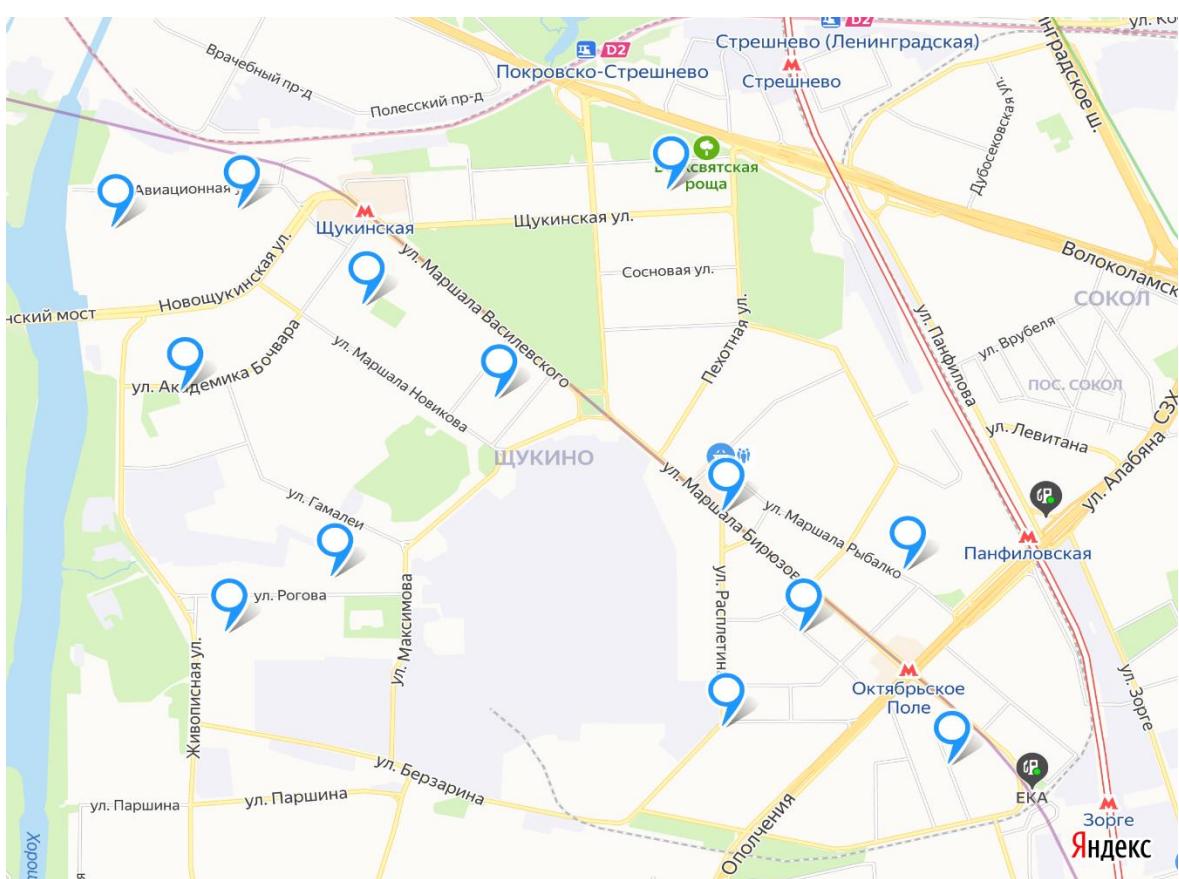


Рисунок 17, Координаты центров для 13 кластеров на карте Щукино. Источник: Конструктор карт "Яндекса"

Если визуализировать полученные данные и отнести каждый из домов к тому или иному кластеру, то первоначально полученная карта жилых домов может быть представлена следующим образом:

```
In [8]: plt.figure(figsize=(8, 8))
location_pred = kmeans.fit_predict(X, sample_weight=weights)
plt.scatter(main_mas[:, 0], main_mas[:, 1], c=location_pred, cmap=plt.cm.rainbow)
plt.xlabel("Широта")
plt.ylabel("Долгота")
plt.title("Прикрепление каждого из домов к расположенному поблизости вендинговому аппарату")

Out[8]: Text(0.5, 1.0, 'Прикрепление каждого из домов к расположенному поблизости вендинговому аппарату')
```

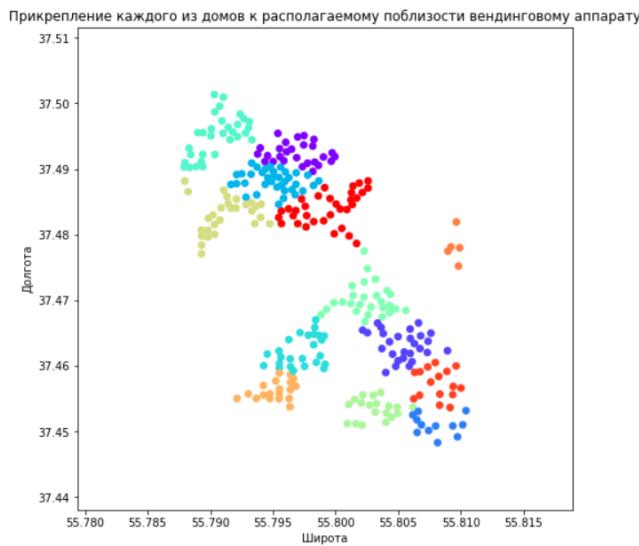


Рисунок 18, Визуальное отображение отношения каждого жилого дома района Щукино к одному из кластеров. Источник: Конструктор карт "Яндекса"

Для сравнения приведём пример поиска единственной центральной точки района, то есть случай, если бы кластеризации не производилось и всем жителям района предлагалось бы ходить только в одно место. Согласно расчётом в [Приложении 7](#), координаты геометрической медианы района дают в 5,5 раз большую сумму расстояний, выраженную в километрах, чем в случае размещения 13 точек реализации дезинфицирующих средств в районе. На карте ниже видно, что в таком случае абсолютно большинству жителей района было бы совершенно неудобно ходить в данную точку и вряд ли он пользовался бы таким же спросом, как две из 13 отдельно взятых точки размещения вендингового автомата. С точки зрения экономики, наличие большего количества брендированных автоматов по выдаче дезинфицирующих средств также является безусловным положительным эффектом с точки зрения укрепления компании на рынке аптек.

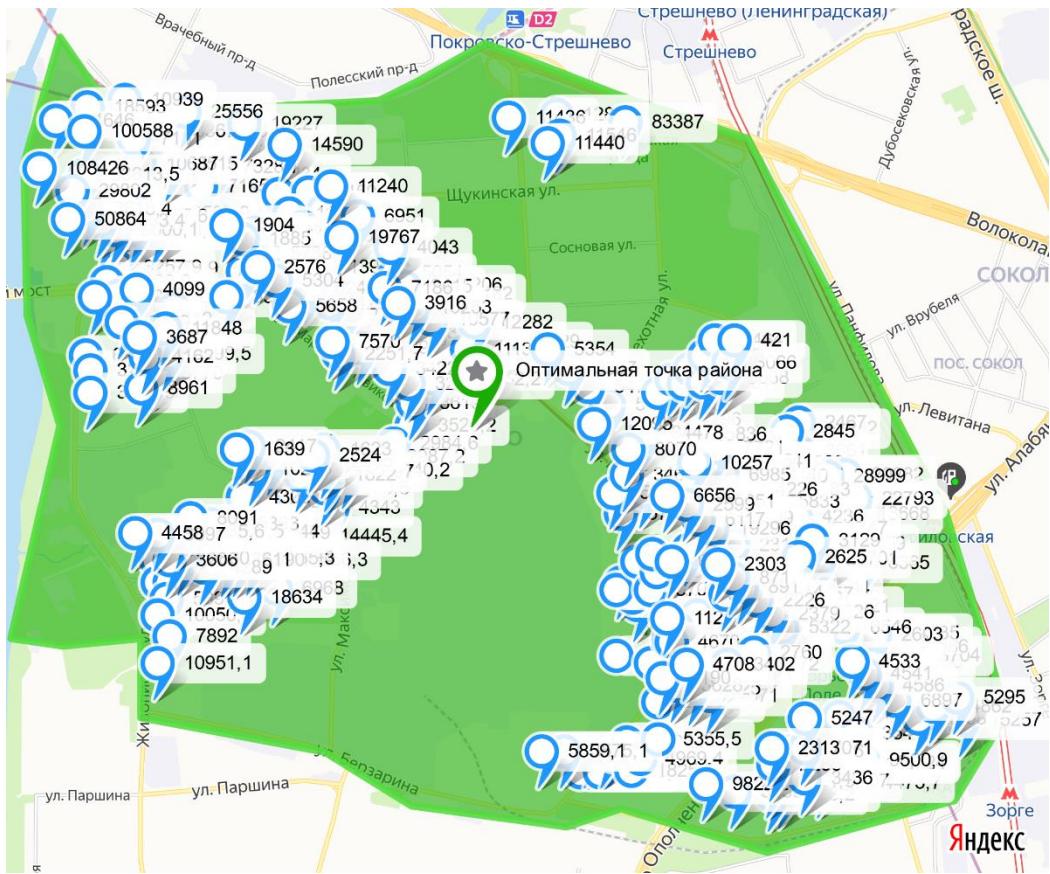


Рисунок 19, координаты оптимального размещения объекта в случае отсутствия разделения на кластеры. Источник: Конструктор карт "Яндекса"

Сумма расстояний до 1 вендинговых автоматов составляет 402.46  
 Сумма расстояний до 2 вендинговых автоматов составляет 201.5  
 Сумма расстояний до 3 вендинговых автоматов составляет 174.95  
 Сумма расстояний до 4 вендинговых автоматов составляет 157.01  
 Сумма расстояний до 5 вендинговых автоматов составляет 145.04  
 Сумма расстояний до 6 вендинговых автоматов составляет 121.3  
 Сумма расстояний до 7 вендинговых автоматов составляет 113.49  
 Сумма расстояний до 8 вендинговых автоматов составляет 101.74  
 Сумма расстояний до 9 вендинговых автоматов составляет 92.58  
 Сумма расстояний до 10 вендинговых автоматов составляет 87.45  
 Сумма расстояний до 11 вендинговых автоматов составляет 83.61  
 Сумма расстояний до 12 вендинговых автоматов составляет 76.98  
 Сумма расстояний до 13 вендинговых автоматов составляет 72.65

Рисунок 20, значения суммы расстояний в километрах в случае разного количества кластеров. Источник: Jupyter Notebook

Таким образом, были найдены координаты размещения объектов и сумма расстояний до них в 13 разных случаях. Мы видим почти везде невозрастающую последовательность сумм расстояний, принимающую своё минимальное значение, равное нулю, при  $k=304$ . Важно заметить, что наличие двух станций метро в районе Щукино делает крайне рекомендуемым размещение как минимум двух точек

реализации, поскольку разница между случаями размещения одного и двух вендинговых аппаратов крайне существенна.

## **4 ВЫВОДЫ**

Решение задачи оптимального размещения является сложным вычислительным процессом, подход к которому необходимо начинать с верной постановки задачи. Подбор правильных начальных условий позволяет чётко осознать, данные какого рода необходимы для успешного применения имеющихся математических методов решения поставленной задачи.

На двух приведённых в главе 3 настоящей работы примерах реальных задач можно убедиться в повсеместной применимости математических методов для задач оптимального размещения объектов и геомаркетинга. Безусловно, данная работа не покрывает весь спектр имеющихся подходов к решению данных задач, однако описывает наиболее популярные из них.

## **5 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Антамошкин А.Н., Казаковцев Л.А. Алгоритм случайного поиска для обобщённой задачи Вебера в дискретных координатах// Красноярск, 2013 [[http://ics.khstu.ru/media/2013/N35\\_11.pdf](http://ics.khstu.ru/media/2013/N35_11.pdf)]
2. Гудыма М.Н. Алгоритмы решения серии задач автоматической группировки// Красноярск, 2017 [<https://www.sibsau.ru/files/1522/>]
3. Панов П.А. О геометрической медиане выпуклых, а также треугольных и других многоугольных областей// Москва, 2018 [<http://mathizv.isu.ru/en/article/file?id=1282>]
4. Панов П.А. Равновесные расположения центров благ по городу// Москва, 2017 [<https://www.econorus.org/repec/journl/2017-33-28-42r.pdf>]
5. Панов П.А., Савватеев А.О. О геометрической медиане треугольной области и других медианоподобных точках// Москва, 2018 [<https://arxiv.org/pdf/1811.07306.pdf>]

6. Шангин Р.Э. Разработка и анализ алгоритмов решения задачи размещения графа// Челябинск, 2015 [https://sp.susu.ru/DissSovet/repository/Shangin\_disser-15.pdf]
7. Васильев И.Л., Ушаков А.В. Релаксации Лагранжа для нелинейной задачи о р-медиане// Иркутск, 2011 [https://cyberleninka.ru/article/n/relaksatsii-lagranzha-dlya-nelineynoy-zadachi-o-p-mediane]
8. Кондратьев В.Д. Методы решения задачи размещения объектов обслуживания // Управление большими системами: сборник трудов, Москва, 2008, стр. 46-56 [https://cyberleninka.ru/article/n/metody-resheniya-zadachi-razmeshcheniya-obektov-obsluzhivaniya]
9. Deza E.I., Deza M.M. Dictionary of distances, Ecole Normale Supérieure. - Paris, 2008.
10. Конструктор карт «Яндекса» [https://yandex.ru/map-constructor/]
11. Сайт «Недвижимость Москвы и Московской области» [https://welty.ru/address/rayon/schukino]
12. Resende M. G. C., Werneck R. F. A grasp with path-relinking for the p-median problem // Technical Report TD-5E53XL, AT&T Labs. 2002.
13. Avella P., Sassano A., Vasilev I. Computational study of large-scale p-median problems // Mathematical Programming. 2007. Vol. 109. P. 89–114.

## 6 ПРИЛОЖЕНИЕ

**6.1 Приложение 1. Решение примера задачи о р-медиане в приложении Microsoft Office Excel через надстройку «Поиск решения».**

The screenshot shows a Microsoft Excel spreadsheet and the 'Solver Parameters' dialog box. The spreadsheet contains a table with 8 rows and 6 columns. The columns are labeled: Точка (Point), Координаты X (Coordinates X), Координаты Y (Coordinates Y), Расстояние(X) (Distance X), Расстояние(Y) (Distance Y), and Расстояние (Distance). The last row is highlighted in green and labeled 'Оптимальная' (Optimal). The data is as follows:

	A	B	C	D	E	F
1	Точка	Координаты X	Координаты Y	Расстояние(X)	Расстояние(Y)	Расстояние
2	A1	0	8	3	2	5
3	A2	1	1	2	5	7
4	A3	9	7	6	1	7
5	A4	6	3	3	3	6
6	A5	1	6	2	0	2
7	A6	4	10	1	4	5
8	Оптимальная	3	6	17	15	32

The 'Solver Parameters' dialog box is open, showing the following settings:

- Optimize: \$F\$8 (Minimize)
- Subject to Constraints:
  - \$B\$8:\$C\$8 <= 10
  - \$B\$8:\$C\$8 = целое
- Options:
  - Checkmark for "Сделать переменные без ограничений неотрицательными" (Make variables non-negative)
  - Method: Поиск решения лин. задач симплекс-методом (Simplex method for linear problems)
  - Search Method: Для гладких нелинейных задач используйте поиск решения нелинейных задач методом ОПГ, для линейных задач - поиск решения линейных задач симплекс-методом, а для негладких задач - эволюционный поиск решения (For smooth nonlinear problems, use the OPG method for nonlinear problems; for linear problems, use the simplex method for linear problems; for nonsmooth problems, use evolutionary search).
- Buttons: Справка (Help), Найти решение (Find Solution), Закрыть (Close)

**6.2 Приложение 2. Пример выполнения программы через приближенные методы решения на языке Python в Jupyter Notebook.**

```

In [1]: import numpy as np
a=[0.4, 0.4]
b=[0.24, 0.51]
c=[0.26, 0.37]
d=[0.22, 0.15]
e=[0.34, 0.85]
f=[0.66, 0.82]
g=[0.92, 0.67]
h=[0.12, 0.94]
i=[0.93, 0.5]
j=[0.96, 0.03]

P_start = [0, 0]

In [2]: main_mas = np.array([a, b, c, d, e, f, g, h, i, j])

In [3]: weights = [1, 1, 1, 1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

In [4]: def vec_distance(a, b):
    return ((a[0]-b[0])**2+(a[1]-b[1])**2)**(1/2)

In [5]: def sum_of_nominator_x (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=main_mas[i, 0]*weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)

In [6]: def sum_of_nominator_y (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=main_mas[i, 1]*weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)

In [7]: def sum_of_denominator (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)

In [8]: def P_opt(P_start, main_mas, num_steps=20, weights=[1]*len(main_mas)):
    P_opt = np.zeros([num_steps+1, 2])
    P_opt[0]= P_start
    for i in range(num_steps):
        P_opt[i+1] = [sum_of_nominator_x(P_opt[i], main_mas, weights)/sum_of_denominator (P_opt[i], main_mas, weights),
                      sum_of_nominator_y(P_opt[i], main_mas, weights)/sum_of_denominator(P_opt[i], main_mas, weights)]
        print(P_opt[i])
    return

```

### 6.3 Приложение 3. Пример выгруженных данных для поиска оптимальных координат размещения дополнительного склада сети супермаркетов “BILLA”

Широта	Долгота	Описание
55,816964	37,403358	ул. Исаковского, 6, корп. 2, Москва, Россия
55,754	37,40592	Рублёвское ш., 48/1, Москва, Россия
55,850103	37,411321	бул. Яна Райниса, 41, Москва, Россия
55,732465	37,412528	Партизанская ул., 10, Москва, Россия
55,75652	37,417527	ул. Крылатские Холмы, 29, Москва, Россия
55,849965	37,417582	бул. Яна Райниса, 18, корп. 1, Москва, Россия
55,671645	37,445799	Озёрная ул., 42, Москва, Россия
55,784198	37,456815	Живописная ул., 12, корп. 1, Москва, Россия
55,808869	37,462318	ул. Маршала Василевского, 17, Москва, Россия
55,866036	37,468074	Валдайский пр., 8, Москва, Россия
55,866036	37,468074	Валдайский пр., 8, Москва, Россия
55,77679	37,470112	просп. Маршала Жукова, 42, стр. 1, Москва, Россия
55,675538	37,470638	ул. Мичуринский Проспект, Олимпийская Деревня, 4, корп. 3, Москва, Россия

**6.4 Приложение 4. Точки, выступившие в качестве препятствий для задачи оптимального размещения дополнительного склада сети супермаркетов “BILLA”**

Широта	Долгота
55.80031086236634	38.00334796797269
55.69083375001141	37.99888664931637
55.604011458389024	37.97846735867491
55.517959656708335	37.82650412472696
55.478933274504726	37.70419539364786
55.48780410246411	37.53613890561076
55.54283634736225	37.364305867280585
55.61761452102683	37.29152144345248
55.71221877648172	37.21444548520048
55.80253287019442	37.20208586605985
55.9398746959981	37.305082692231736
55.98243539752741	37.4279922381301
56.00975729686231	37.57115782650904
55.98551489093684	37.70574034604027
55.94758147412878	37.84547270688013
55.900160394033044	37.933020009126224
55.85731572782485	37.99756468686057

## 6.5 Приложение 5. Решение задачи поиска оптимальной точки размещения дополнительного склада для сети супермаркетов “BILLA”, выполненное на языке Python в Jupyter Notebook.

```
In [1]: import numpy as np
import pandas as pd
from math import cos

P_start = [0, 0]
out_of_limitation = False

In [2]: data = pd.read_excel('BILLA_part.xlsx')

In [3]: data.tail()

Out[3]:
    Широта   Долгота          Описание
105  55.716973  37.811440      ул. Хлобыстова, 20, Москва, Россия
106  55.796486  37.812725      13-я Парковая ул., 10/60, Москва, Россия
107  55.804945  37.818890      15-я Парковая ул., 40, корп. 1, Москва, Россия
108  55.735692  37.829799      Вешняковская ул., 13А, Москва, Россия
109  55.755753  37.832714      ул. Молостовых, 13, корп. 1, Москва, Россия

In [4]: min_width = min(data['Широта'])
max_width = max(data['Широта'])
min_length = min(data['Долгота'])
max_length = max(data['Долгота'])
pre_limits = [min_width, max_width, min_length, max_length]
pre_limits

Out[4]: [55.58753, 55.897689, 37.403358, 37.832714]

In [5]: limitations_BILLA = pd.read_excel('BILLA_part.xlsx', sheet_name = 'Ограничения')

In [7]: pre_mas = []
for i in range(len(data)):
    pre_mas.append([data['Широта'][i], data['Долгота'][i]])

In [8]: main_mas = np.array(pre_mas)

In [9]: limitations = []
for i in range(len(limitations_BILLA)):
    limitations.append([limitations_BILLA['Широта'][i], limitations_BILLA['Долгота'][i]])

In [10]: def vec_distance(a, b):
    return (((a[0]-b[0])*40008.55/360)**2+((a[1]-b[1])*40075.696/360*cos(a[0]/2+b[0]/2))**2)**(1/2)

In [11]: def find_nearest_limitation(P_opt, limitations):
    distance_min=100000
    for i in range(len(limitations)):
        if vec_distance(limitations[i], P_opt)<distance_min:
            distance_min = vec_distance(limitations[i], P_opt)
            ind_min = i
    return ind_min

In [12]: def limitations_handling(P_opt, limitations, pre_limits, out_of_limitation):
    nearest = limitations[find_nearest_limitation(P_opt, limitations)]
    if P_opt[0]>pre_limits[1]:
        if nearest[0]>P_opt[0] or nearest[1]<P_opt[1]:
            return [x + y for x, y in zip(P_opt, [0.01, -0.01])] #делаем оптимальную точку чуть выше и левее
    else:
        out_of_limitation = True
    return P_opt
```

```

    elif P_opt[1]>pre_limits[3]: #проверяем правую верхнюю часть относительно МКАД
        if nearest[0]>P_opt[0] or nearest[1]>P_opt[1]:
            return [x + y for x, y in zip(P_opt, [0.01, 0.01])] #делаем оптимальную точку чуть выше и правее
        else:
            out_of_limitation = True
            return P_opt

    else: #проверяем верхнюю часть относительно МКАД
        if nearest[0]>P_opt[0]:
            return [x + y for x, y in zip(P_opt, [0.01, 0])] #делаем оптимальную точку чуть выше
        else:
            out_of_limitation = True
            return P_opt

    elif P_opt[0]<=pre_limits[0]:
        if P_opt[1]>pre_limits[3]: #проверяем правую нижнюю часть относительно МКАД
            if nearest[0]<P_opt[0] or nearest[1]>P_opt[1]:
                return [x + y for x, y in zip(P_opt, [-0.01, 0.01])] #делаем оптимальную точку чуть ниже и правее
            else:
                out_of_limitation = True
                return P_opt

        elif P_opt[1]<pre_limits[2]: #проверяем левую нижнюю часть относительно МКАД
            if nearest[0]<P_opt[0] or nearest[1]<P_opt[1]:
                return [x + y for x, y in zip(P_opt, [-0.01, -0.01])] #делаем оптимальную точку чуть ниже и левее
            else:
                out_of_limitation = True
                return P_opt

        else: #проверяем нижнюю часть относительно МКАД
            if nearest[0]<P_opt[0]:
                return [x + y for x, y in zip(P_opt, [-0.01, -0.01])] #делаем оптимальную точку чуть ниже и левее
            else:
                out_of_limitation = True
                return P_opt

    else:
        if P_opt[1]>pre_limits[3]: #проверяем правую часть относительно МКАД
            if nearest[1]>P_opt[1]:
                return [x + y for x, y in zip(P_opt, [0, 0.01])] #делаем оптимальную точку чуть правее
            else:
                out_of_limitation = True
                return P_opt

        elif P_opt[1]<pre_limits[2]: #проверяем левую часть относительно МКАД
            if nearest[1]<P_opt[1]:
                return [x + y for x, y in zip(P_opt, [0, -0.01])] #делаем оптимальную точку чуть левее
            else:
                out_of_limitation = True
                return P_opt

        else: #если объект принимает недопустимое значение, то присваиваем ему значение ближайшей точки ограничения
            return nearest

```

```
In [13]: def sum_of_nominator_x (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=main_mas[i, 0]*weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)
```

```
In [14]: def sum_of_nominator_y (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=main_mas[i, 1]*weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)
```

```
In [15]: def sum_of_denominator (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)
```

```
In [16]: def P_opt(P_start, main_mas, num_steps=20, weights=[1]*len(main_mas), limitations=None, pre_limits=None):
    candidates=[]
    P_opt = np.zeros([num_steps+1, 2])
    P_opt[0]= P_start
    for i in range(num_steps):
        P_opt[i+1] = [sum_of_nominator_x(P_opt[i], main_mas, weights)/sum_of_denominator(P_opt[i], main_mas, weights),
                      sum_of_nominator_y(P_opt[i], main_mas, weights)/sum_of_denominator(P_opt[i], main_mas, weights)]
    if limitations!=None and pre_limits!=None and i%30==0 and i>0:
        P_opt[i+1] = limitations_handling(P_opt[i+1], limitations, pre_limits, out_of_limitation)
        print('Следующий элемент является геометрической медианой для данного множества.')
        print('Элемент после следующего является наиболее близким допустимым значением к геометрической медиане')
        Optimal_point=list(P_opt[i+1])
    print(P_opt[i])
return Optimal_point
```

```
In [17]: optimal_point = P_opt(P_start, main_mas, num_steps=300, limitations=limitations, pre_limits=pre_limits)
```

```
[0. 0.]  
[55.76287556 37.63151642]  
[55.76663833 37.6338289 ]  
[55.76861942 37.63435128]  
[55.76970111 37.63433074]  
[55.77030762 37.63418673]  
[55.77065488 37.63404756]  
[55.77085689 37.63394265]  
[55.7709758 37.63387094]  
[55.77104639 37.63382427]  
[55.77108853 37.63379473]  
[55.77111379 37.63377634]  
[55.77112897 37.63376502]  
[55.77113812 37.63375809]  
[55.77114363 37.63375387]  
[55.77114696 37.6337513 ]  
[55.77114897 37.63374975]  
[55.77115018 37.63374881]  
  
[55.77115091 37.63374823]  
[55.77115135 37.63374789]  
[55.77115162 37.63374768]  
[55.77115178 37.63374755]  
[55.77115188 37.63374748]  
[55.77115194 37.63374743]  
[55.77115198 37.6337474 ]  
[55.771152 37.63374739]  
[55.77115201 37.63374738]  
[55.77115202 37.63374737]  
[55.77115203 37.63374737]  
Следующий элемент является геометрической медианой для данного множества.  
Элемент после следующего является наиболее близким допустимым значением к геометрической медиане  
[55.77115203 37.63374736]  
[55.98551489 37.70574034604025]  
... 30010010 37.636010001
```

```
In [18]: optimal_point
```

```
Out[18]: [55.98551489093684, 37.70574034604027]
```

```
In [19]: sum_of_dis_opt = 0
for i in range(len(main_mas)):
    sum_of_dis_opt += vec_distance(main_mas[i], optimal_point)
```

```
In [23]: round(sum_of_dis_opt, 2)
```

```
Out[23]: 3046.22
```

```

In [18]: optimal_point
Out[18]: [55.98551489093684, 37.70574034604027]

In [19]: sum_of_dis_opt = 0
for i in range(len(main_mas)):
    sum_of_dis_opt += vec_distance(main_mas[i], optimal_point)

In [23]: round(sum_of_dis_opt, 2)
Out[23]: 3046.22

In [21]: sum_of_dis = [0]*len(limitations)
for i in range(len(main_mas)):
    for j in range(len(limitations)):
        sum_of_dis[j] += vec_distance(main_mas[i], limitations[j])

In [25]: for i in sum_of_dis:
    print(round(i, 2))

3508.27
3362.42
3506.57
3445.42
3616.52
3556.94
3528.93
3426.33
3734.17
3963.99
3879.25
3461.15
3256.87
3046.22
3264.49
3469.5
3707.97

```

## 6.6 Приложение 6. Пример выгруженных данных для поиска оптимальных координат размещения вендинговых автоматов по продаже дезинфицирующих средств в районе Щукино г. Москва

Широта	Долгота	Описание	Площадь
55.789795	37.479609	улица Расплетина, 1	4745
55.789315	37.47978	улица Расплетина, 2	12664
55.790342	37.480058	улица Расплетина, 3	13790
55.789806	37.480687	улица Расплетина, 4к1	5777
55.789289	37.480714	улица Расплетина, 4к2	1148
55.790347	37.481531	улица Расплетина, 6к1	3292
55.789856	37.482493	улица Расплетина, 6к2	1825
55.793561	37.481675	улица Расплетина, 7	4113
55.790772	37.482223	улица Расплетина, 8к1	3997
55.790216	37.483014	улица Расплетина, 8к2	4969
55.794021	37.482636	улица Расплетина, 9	7003
55.79475	37.481666	улица Расплетина, 11	5513
55.790874	37.484065	улица Расплетина, 12к2	5356

## 6.7 Приложение 7. Решение задачи поиска координат оптимальных точек размещения вендинговых аппаратов по продаже дезинфицирующих средств в районе Щукино г. Москва, выполненное на языке Python в Jupyter Notebook

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from math import cos

P_start = [0, 0]

In [2]: data = pd.read_excel('shchukino.xlsx', header=0)

In [3]: data.tail()

Out[3]:
    Широта   Долгота      Описание  Площадь Население
300  55.801389  37.487415  3-й Волоколамский проезд, 10к2    3068    132.24
301  55.802133  37.486427  3-й Волоколамский проезд, 10к1    1810     78.02
302  55.802588  37.487110  3-й Волоколамский проезд, 12к1    4464    192.41
303  55.801890  37.487865  3-й Волоколамский проезд, 12к2    3066    132.15
304  55.802594  37.488188  3-й Волоколамский проезд, 14к1      421     18.15

In [4]: pre_mas = []
for i in range(len(data)):
    pre_mas.append([data['Широта'][i], data['Долгота'][i]])

In [5]: main_mas = np.array(pre_mas)

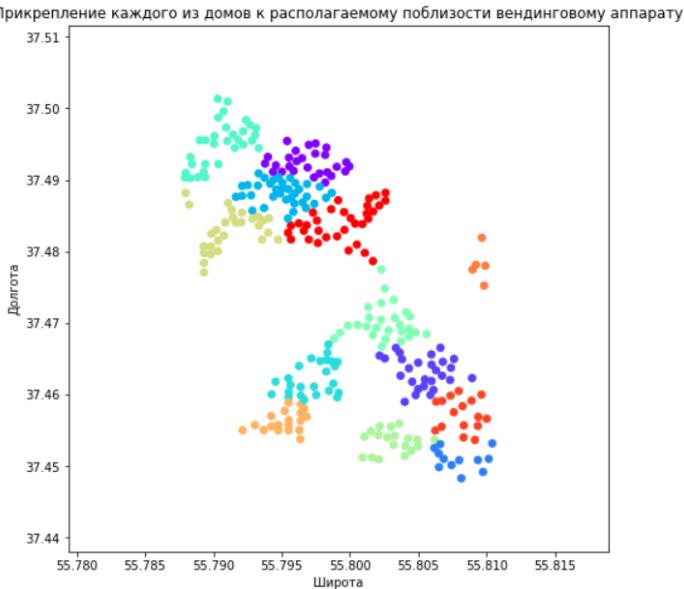
In [6]: weights = []
for i in range(len(data)):
    weights.append(data['Население'][i])

In [7]: X = main_mas
kmeans = KMeans(n_clusters=13, random_state=0)
kmeans.fit(X, sample_weight=weights).cluster_centers_

Out[7]: array([[55.79697046, 37.49314625],
       [55.80576348, 37.46364915],
       [55.80831116, 37.45005669],
       [55.7948686 , 37.48748762],
       [55.79672708, 37.46192157],
       [55.79047447, 37.495519 ],
       [55.80268601, 37.47084648],
       [55.80289409, 37.45380678],
       [55.79172765, 37.48322024],
       [55.79487295, 37.45619676],
       [55.80957767, 37.48023258],
       [55.80896842, 37.45692319],
       [55.79891441, 37.48326581]])
```

```
In [8]: plt.figure(figsize=(8, 8))
location_pred = kmeans.fit_predict(X, sample_weight=weights)
plt.scatter(main_mas[:, 0], main_mas[:, 1], c=location_pred, cmap=plt.cm.rainbow)
plt.xlabel("Широта")
plt.ylabel("Долгота")
plt.title("Прикрепление каждого из домов к расположаемому поблизости вендинговому аппарату")

Out[8]: Text(0.5, 1.0, 'Прикрепление каждого из домов к расположаемому поблизости вендинговому аппарату')
```



```
In [9]: def vec_distance(a, b):
    return (((a[0]-b[0])*40008.55/360)**2+((a[1]-b[1])*40075.696/360*cos(a[0]/2+b[0]/2))**2)**(1/2)
```

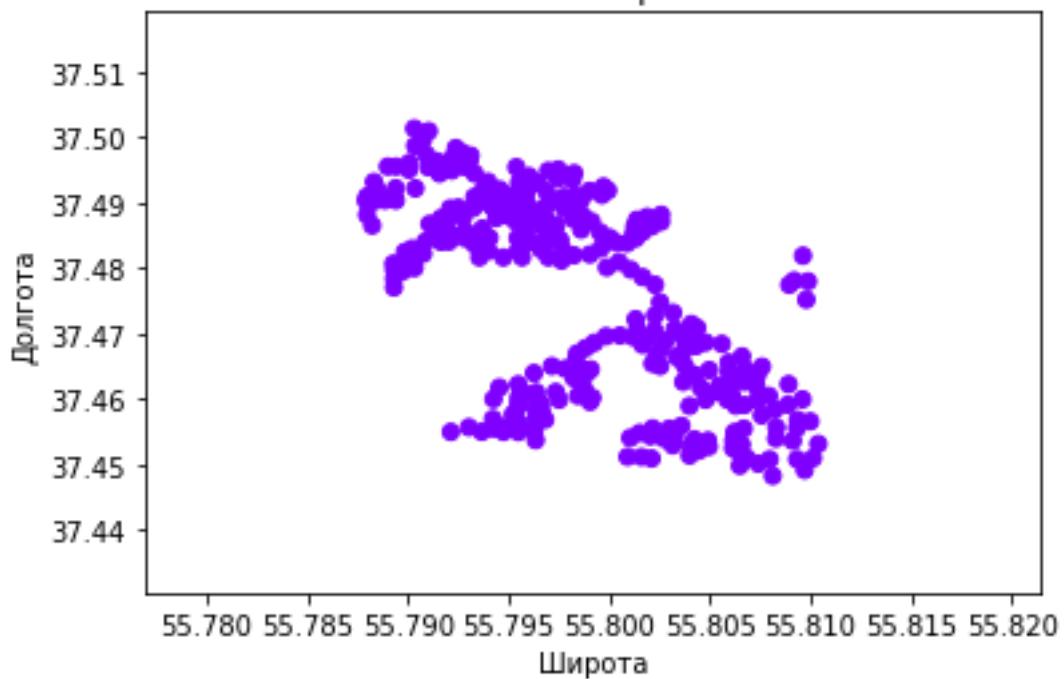
```
In [10]: def find_optimal_k_cluster(main_mas, weights, k):
    kmeans = KMeans(n_clusters=k, random_state=0)
    center_location = kmeans.fit(main_mas, sample_weight=weights).cluster_centers_
    location_pred = kmeans.fit_predict(main_mas, sample_weight=weights)
    return center_location, main_mas, location_pred
```

```
In [11]: def sum_of_distances_k_clusters(center_location, main_mas, location_pred):
    sum_of_distances = [0]*len(center_location)
    for i in range(len(center_location)):
        for j in range(len(main_mas)):
            if location_pred[j]==i:
                sum_of_distances[i] += vec_distance(center_location[i], main_mas[j])
    return sum_of_distances
```

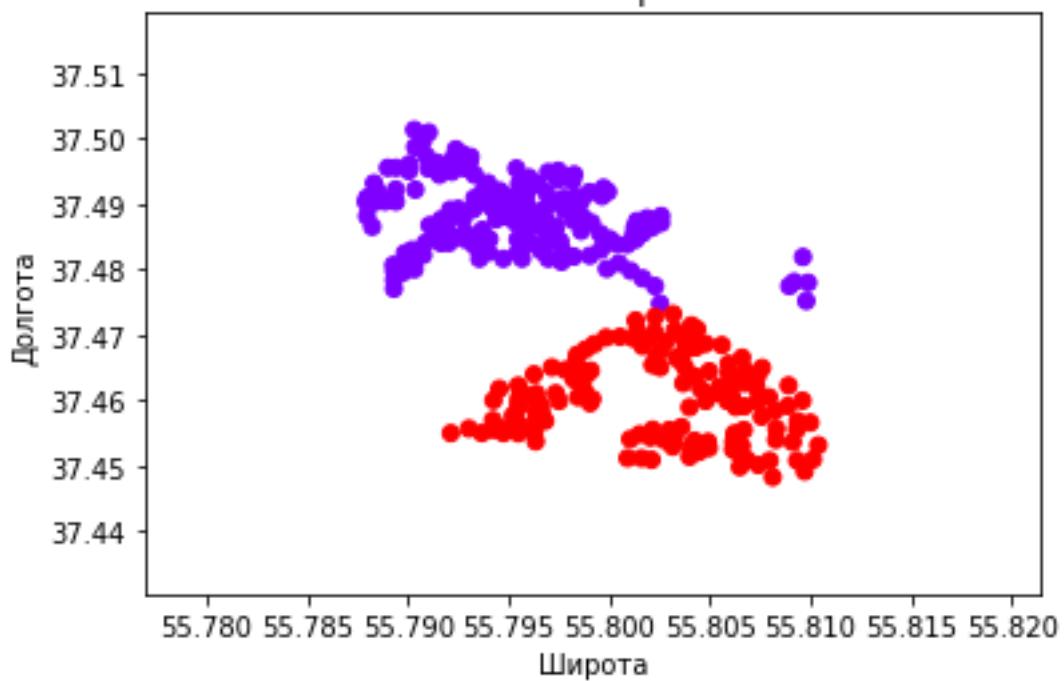
```
In [20]: for i in range(13):
    center_location, main_mas, location_pred = find_optimal_k_cluster(main_mas, weights, i+1)
    print('Сумма расстояний до', i+1, 'вендинговых автоматов составляет',
          round(sum(sum_of_distances_k_clusters(center_location, main_mas, location_pred)), 2))
```

Сумма расстояний до 1 вендинговых автоматов составляет 402.46  
 Сумма расстояний до 2 вендинговых автоматов составляет 201.5  
 Сумма расстояний до 3 вендинговых автоматов составляет 174.95  
 Сумма расстояний до 4 вендинговых автоматов составляет 157.01  
 Сумма расстояний до 5 вендинговых автоматов составляет 145.04  
 Сумма расстояний до 6 вендинговых автоматов составляет 121.3  
 Сумма расстояний до 7 вендинговых автоматов составляет 113.49  
 Сумма расстояний до 8 вендинговых автоматов составляет 101.74  
 Сумма расстояний до 9 вендинговых автоматов составляет 92.58  
 Сумма расстояний до 10 вендинговых автоматов составляет 87.45  
 Сумма расстояний до 11 вендинговых автоматов составляет 83.61  
 Сумма расстояний до 12 вендинговых автоматов составляет 76.98  
 Сумма расстояний до 13 вендинговых автоматов составляет 72.65

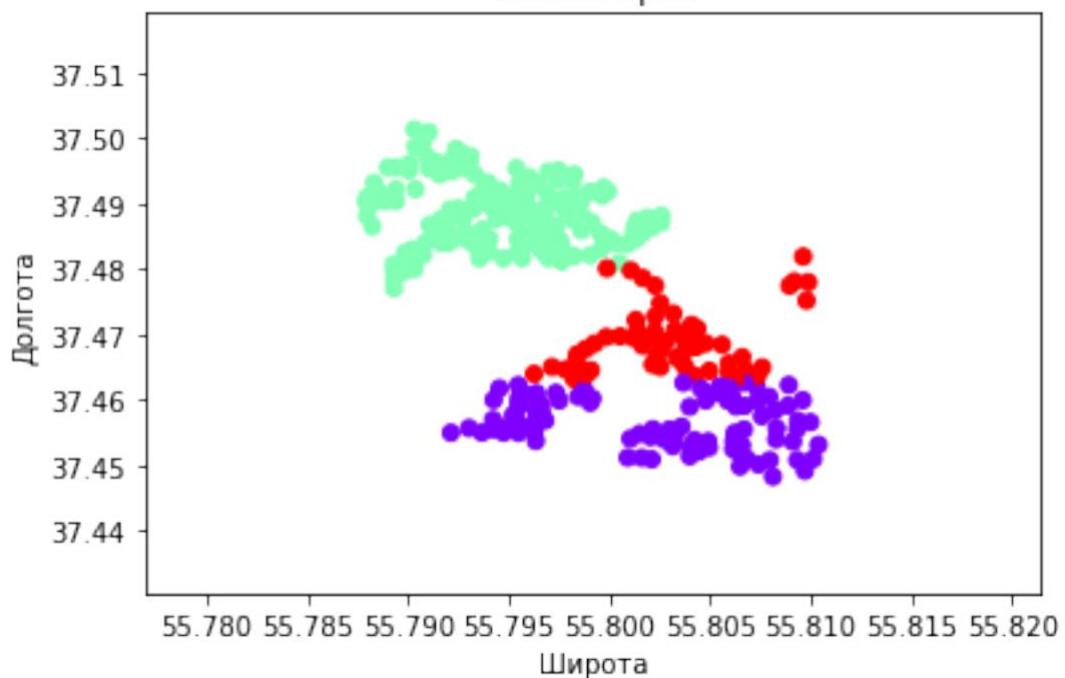
1 кластеров



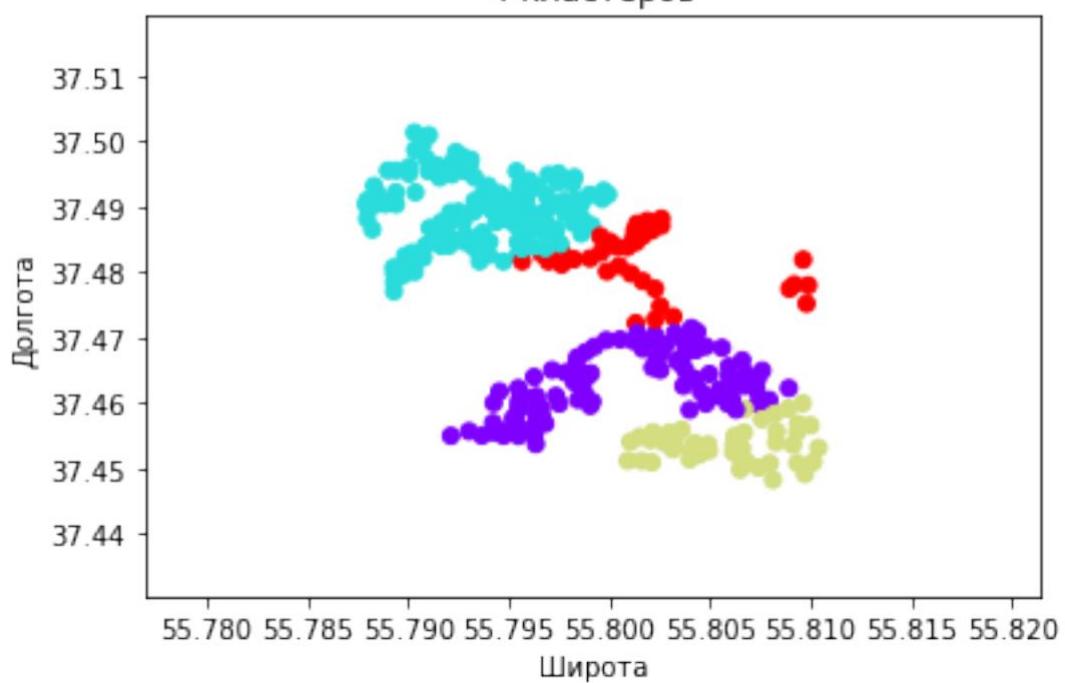
2 кластеров



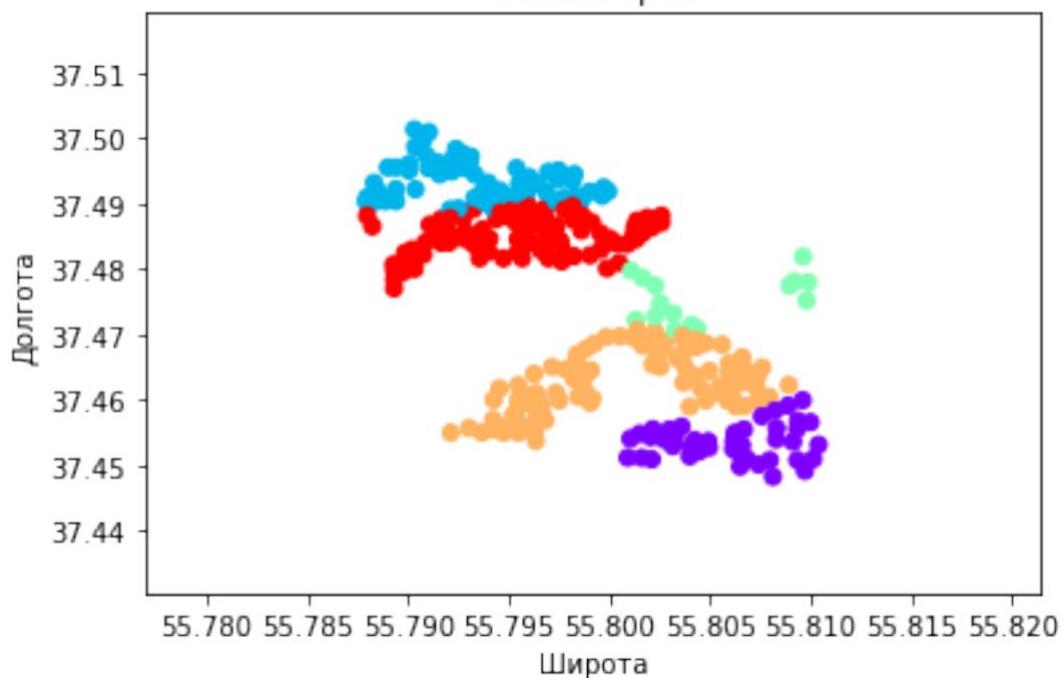
3 кластеров



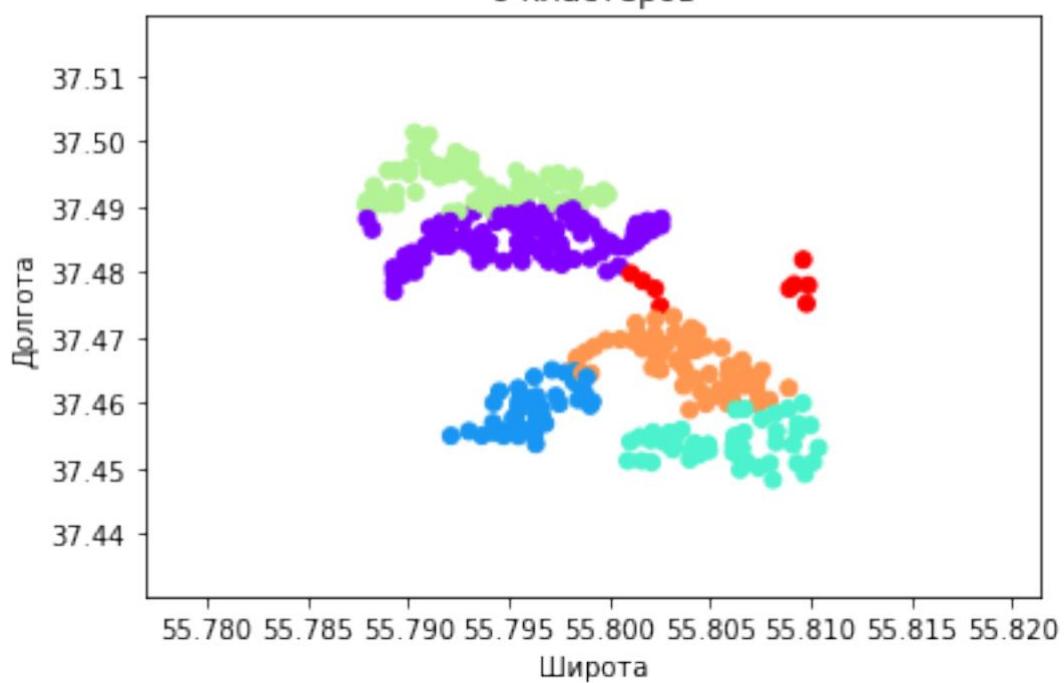
4 кластеров



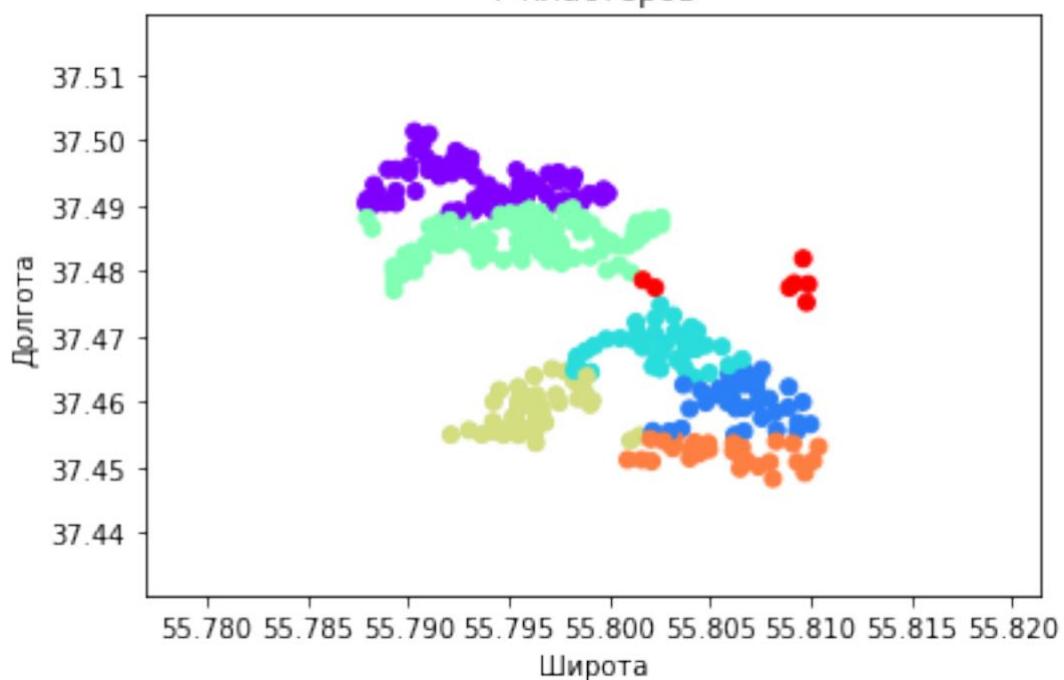
5 кластеров



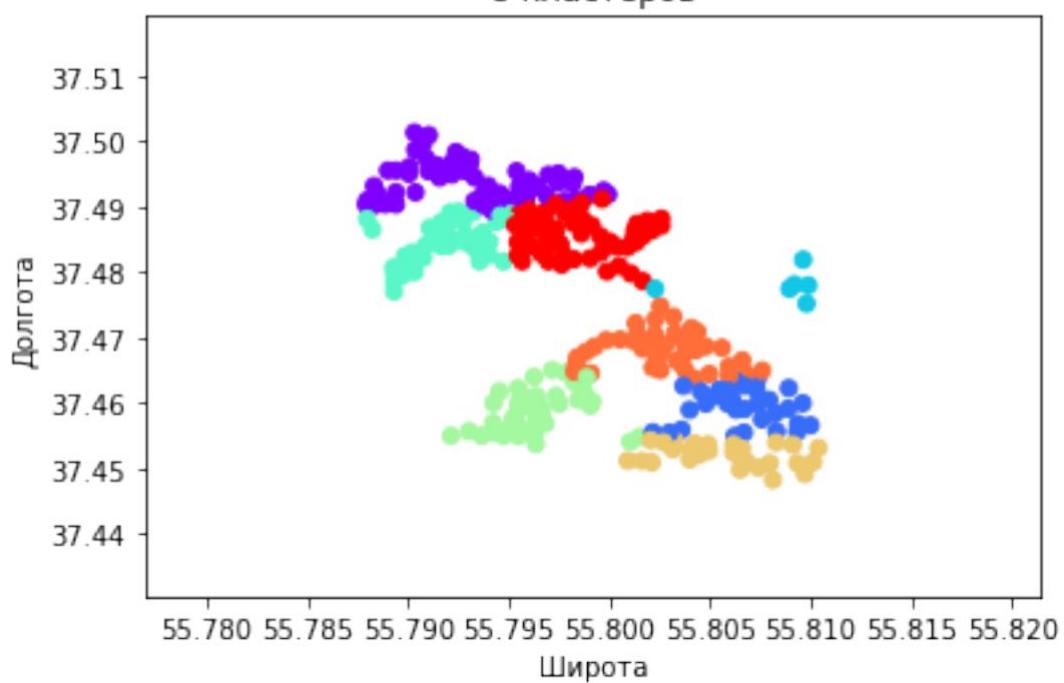
6 кластеров



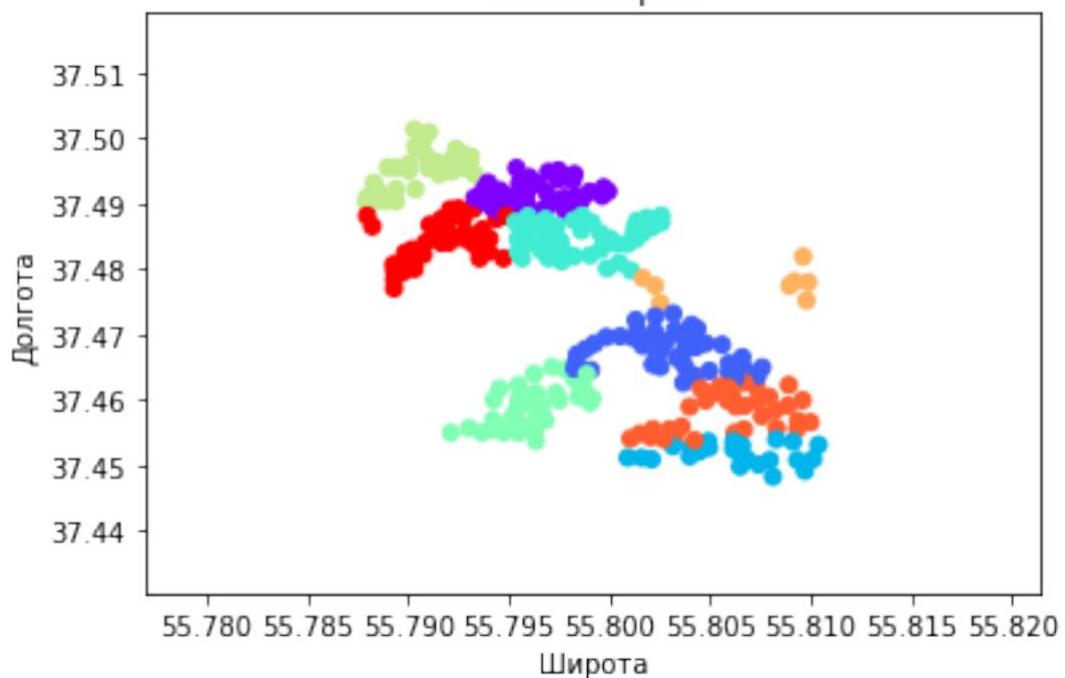
7 кластеров



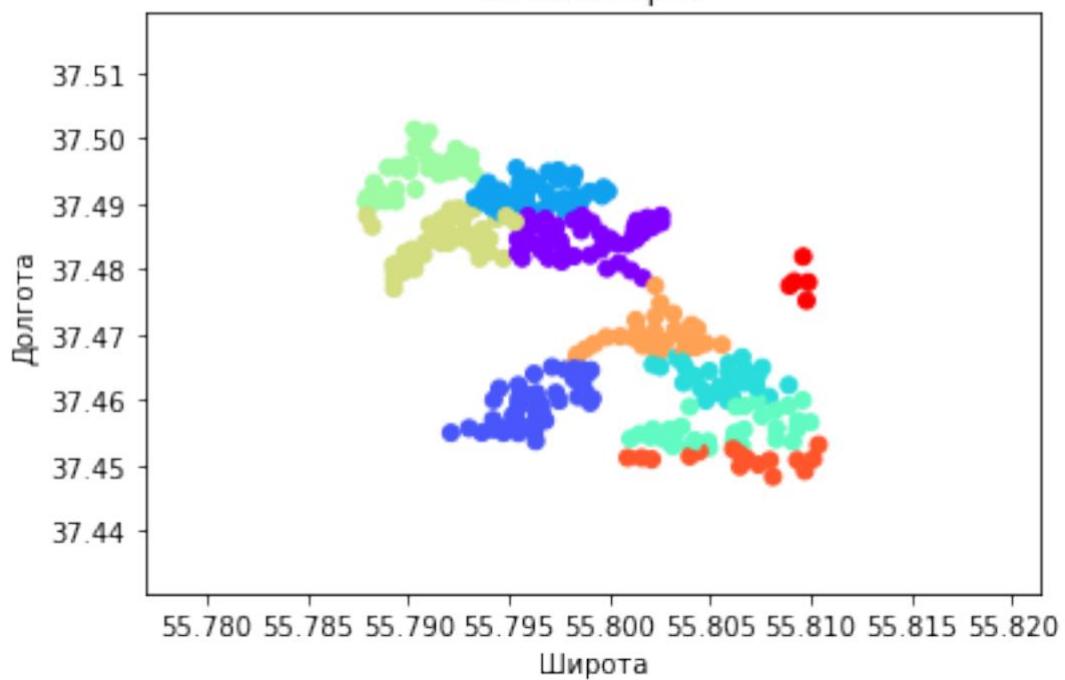
8 кластеров



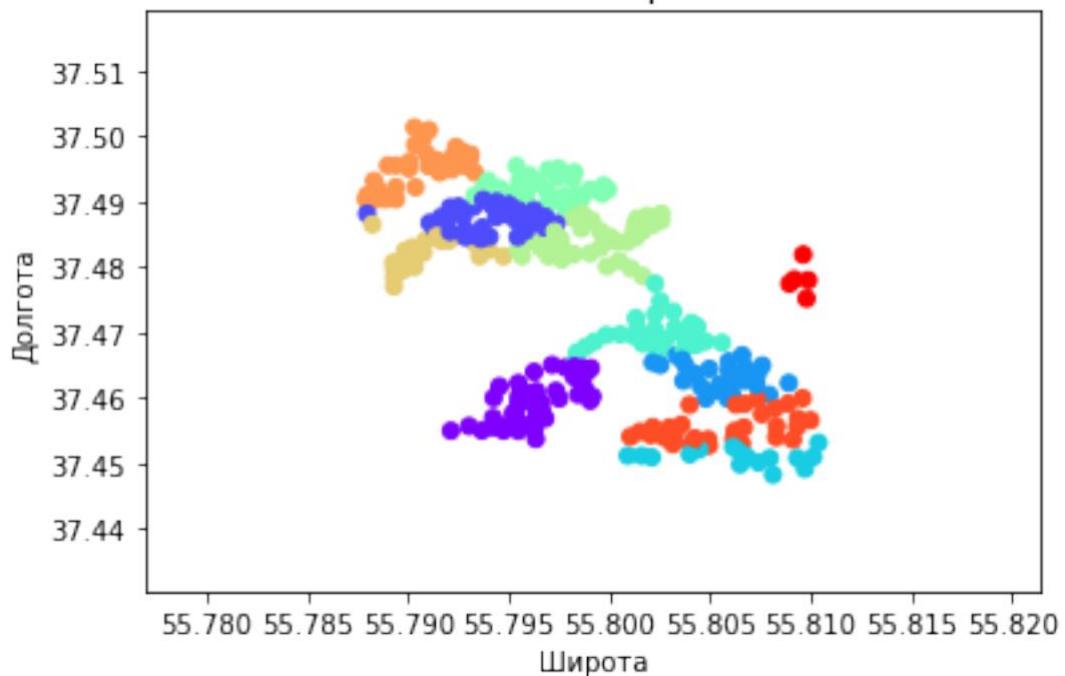
9 кластеров



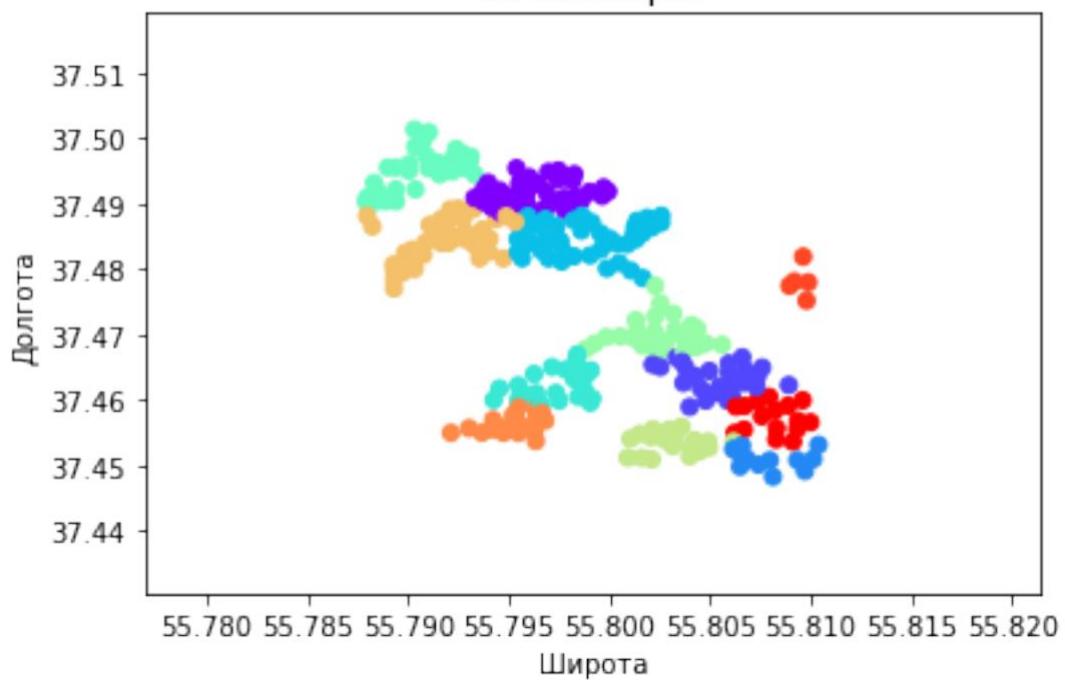
10 кластеров

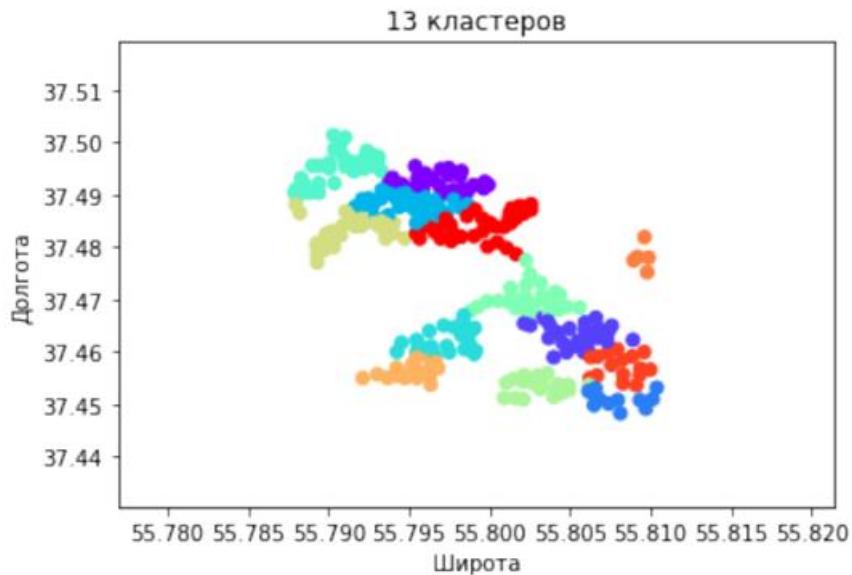


11 кластеров



12 кластеров





```
In [14]: #Сравним со случаем, когда существует лишь единственный вендинговый автомат в районе
def sum_of_nominator_x (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=main_mas[i, 0]*weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)

In [15]: def sum_of_nominator_y (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=main_mas[i, 1]*weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)

In [16]: def sum_of_denominator (P_opt, main_mas, weights=[1]*len(main_mas)):
    summator = [0]*len(main_mas)
    for i in range(len(main_mas)):
        summator[i]=weights[i]/vec_distance(P_opt, main_mas[i])
    return sum(summator)

In [17]: def P_opt(P_start, main_mas, num_steps=20, weights=[1]*len(main_mas)):
    P_opt = np.zeros([num_steps+1, 2])
    P_opt[0]= P_start
    for i in range(num_steps):
        P_opt[i+1] = [sum_of_nominator_x(P_opt[i], main_mas, weights)/sum_of_denominator (P_opt[i], main_mas, weights),
                      sum_of_nominator_y(P_opt[i], main_mas, weights)/sum_of_denominator(P_opt[i], main_mas, weights)]
        print(P_opt[i])
    return
```

```
In [22]: P_opt(P_start, main_mas, 25, weights)
[0. 0.]
[55.80037125 37.4727649]
[55.80066395 37.47293043]
[55.80073682 37.47295724]
[55.80075733 37.47295472]
[55.80076461 37.4729474]
[55.80076816 37.47294054]
[55.80077039 37.47293503]
[55.80077196 37.47293078]
[55.80077312 37.47292753]
[55.800774 37.47292506]
[55.80077467 37.47292319]
[55.80077517 37.47292177]
[55.80077555 37.47292069]
[55.80077584 37.47291987]
[55.80077606 37.47291925]
[55.80077623 37.47291878]
[55.80077636 37.47291842]
[55.80077645 37.47291815]
[55.80077652 37.47291794]
[55.80077658 37.47291779]
[55.80077662 37.47291767]
[55.80077665 37.47291758]
[55.80077668 37.47291751]
[55.8007767 37.47291746]

In [19]: only_optimal_location = [55.80077675, 37.4729173]
sum_only_optimal = 0
for i in range(len(main_mas)):
    sum_only_optimal += vec_distance(only_optimal_location, main_mas[i])
round(sum_only_optimal, 2)

Out[19]: 403.72
```

## **6.8 Приложение 8. Ссылка на папку в Яндекс.Диске с вспомогательными инструментами для необходимых вычислений по настоящей работе**

<https://yadi.sk/d/1vXBxLbiWUmMVg>