Московский физико-технический институт
(национальный исследовательский университет)

Метрическая задача коммивояжёра

Алгоритмический проект
по курсу «Сложность вычислений»

Евгений Абрамов
группа Б05-821

Физтех-школа прикладной математики и информатики
ноябрь-декабрь 2020

Аннотация

Traveling salesman problem, или *задача коммивояжёра* — известная задача комбинаторной оптимизации, связанная с поиском кратчайшего пути, проходящего через указанные пункты ровно по одному разу с последующим возвратом в стартовую позицию. В терминах графа такая задача означает поиск гамильтонова цикла наименьшего веса. В этой работе я покажу, что при условии $\mathbf{P} \neq \mathbf{NP}$ не существует работающих за полиномиальное время алгоритмов константного приближения ее решения. А для метрической постановки задачи, когда граф полон и его ребра удовлетворяют неравенству треугольника, я рассмотрю алгоритмы 2- и 1.5-приближения, докажу их корректность и получу оценку точности приближения на тестовых данных для собственной реализации.

Содержание

1 Введение

Задача коммивояжёра была впервые сформулирована в 1930 году и является одной из самых широко исследованных задач. Она часто используется для теста производительности методов оптимизации. Существует несколько общих постановок задачи в зависимости от использованной метрики и возможности при обходе всех вершин посетить какую-либо больше одного раза, а также различий в стоимости прохода по ребру и обратному к нему. В качестве частных случаев выделяют *геометрическую задачу* (с евклидовой метрикой), *симметричные/асимметричные* и *метрическую*, в рамках которой граф полон, а для его ребер выполняется неравенство треугольника.

Несмотря на то, что задача является вычислительно сложной, известны многие как точные, так и приближающие алгоритмы. Один из первых эффективных алгоритмов был разработан в 1954 году американскими учеными Джоджем Данцигом, Делбертом Реем Фалкерсоном и Селмером Джонсоном и использовал метод отсечений. В 1972 году Ричард Карп доказал **NP**-полноту задачи, обосновав тем самым вычислительную сложность на практике. Последующие продвижения в решении задачи были получены за счет использования методов деления плоскостью, ветвей и границ. Современные решения основаны на методах декомпозиции, которые позволяют эффективно получать хоть и приближенное решение, но с отличием от оптимального не более, чем на 1%.

Задача имеет большое практическое применение. Так, даже решение в рамках исходной математической формулировки может быть использовано для задач планирования и логистики, например, эффективного расположения элементов на материнской плате компьютера, поскольку время передачи информации по шине зависит от ее длины. Для сведения к задаче коммивояжёра требуется сопоставить понятия исходной задачи вершинам и ребрам графа и использовать подходящую метрику. Например, для задачи секвенирования ДНК в биоинформатике цепочке ДНК сопоставляют вершину в полном графе, а «степени сходства» пары таких цепочек — вес на ребре между соответствующими вершинами.

2 Используемые определения и утверждения

Опр. ***Гамильтонов цикл*** *— замкнутый простой путь, проходящий через все вершины графа по одному разу.*

Формулировка задачи:
Пусть задан граф $G = (V, E)$ и весовая функция $\omega : E \to \mathbb{R}_+$. Найти гамильтонов цикл минимального веса.

Опр. *Будем говорить, что алгоритм даёт* α***-приближение*** *задачи коммивояжёра, если он работает за полиномиальное время и результат отличается от оптимального (больше) не более, чем в* α *раз. Значение* α *можно рассматривать как функцию* $\alpha = \alpha(n)$*, где* n *— характеристика размера входа, в частности* $n = |V|$*.*

Теорема. *Если* $\boldsymbol{P \neq NP}$*, то для стандартной задачи коммивояжёра не существует алгоритма* α*-приближения, где* $\alpha = \alpha(n)$ *— константа.*

Доказательство. От противного. Предположим, что для некоторой константы α найдется алгоритм, дающий α—приближение решения задачи коммивояжёра. Покажем, что в таком случае **P**=**NP**. Для этого докажем **NP**-трудность задачи поиска α-приближения, которая, по условию, решается за полиномиальное время.

Сведем **NP**-полную задачу поиска гамильтонова цикла в графе к задаче поиска α - приближения минимального гамильтонова цикла. Для этого сопоставим графу $G = (V, E)$ с весовой функцией ω полный граф $G' = (V, E')$ на том же множестве вершин, для которого:

$$\omega'((v_i, v_j)) = \begin{cases} 1, (v_i, v_j) \in E \\ \alpha \cdot n, (v_i, v_j) \notin E \end{cases}$$

Пусть в исходном графе был гамильтонов цикл. Тогда в новом графе старые ребра будут образовывать гамильтонов цикл веса n. Значит, алгоритм α-приближения находит в новом графе гамильтонов цикл размера не более $\alpha \cdot n$. В силу задания весов ребер в новом графе, найденный алгоритмом гамильтонов цикл состоит только из ребер исходного графа.

В обратную сторону, пусть в исходном графе нет гамильтонова цикла. Тогда в новом графе приближающий алгоритм найдет гамильтонов цикл (поскольку он полный), причем вес такого цикла будет строго больше $\alpha \cdot n$ (так как он должен содержать хотя бы 1 ребро $\notin E$).

Итого: в исходном графе есть гамильтонов цикл $\iff$ найденный алгоритмом α - приближения гамильтонов цикл имеет вес не более $\alpha \cdot n$. Значит, задача поиска α-приближения является **NP**-трудной и имеет полиномиальный алгоритм решения, откуда **P**=**NP**. Противоречие. ■

Рассуждение выше опиралось, в частности, на то, что граф не обязан удовлетворять неравенству треугольника. Поэтому если добавить дополнительные ограничения, например, как в метрической задаче, то такую жесткую оценку трудности сделать уже не получится. Более того, можно показать, что задача аппроксимации в таких ситуациях может иметь эффективное решение.

Опр. ***Остовное дерево*** *— подграф исходного графа, который является деревом и содержит все вершины.* ***Минимальное остовное дерево*** *— остовное дерево минимального веса.*

Опр. ***Эйлеров путь*** *— путь, который проходит по каждому ребру в графе ровно один раз.* ***Эйлеров цикл*** *— замкнутый эйлеров путь.*

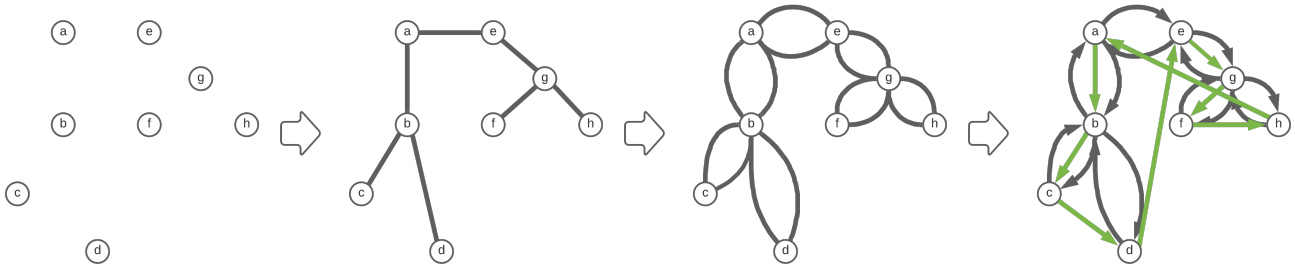
Опр. ***Паросочетание*** *в графе — такое его множество ребер, которое не содержит ребер, инцидентных одной и той же вершине.* ***Совершенное паросочетание*** *— покрывающее все вершины графа. Паросочетание называется* ***максимальным****, если оно не содержится ни в одном другом паросочетании исходного графа и* ***наибольшим****, если оно максимально по числу ребер.*

Из таких определений вытекает, что совершенное паросочетание одновременно является и максимальным, и наибольшим.

3 Алгоритм 2-приближения

3.1 Описание и обоснование корректности

1. Построим по исходному графу минимальное остовное дерево
2. По остовному дереву построим новый неориентированный граф, полученный удвоением ребер этого дерева
3. Построим эйлеров цикл в новом графе
4. Упорядочим вершины в порядке первого появления при некотором обходе этого цикла
5. Простой цикл, образованный упорядоченными вершинами, и будет искомым



Теорема. *Построенный алгоритмом гамильтонов цикл будет 2-приближением решения задачи коммивояжёра.*

Доказательство. Сперва отметим, что эйлеров цикл в пункте 3 можно получить по некоторому обходу дерева в глубину из произвольной стартовой вершины, причем два ребра между соседними вершинами используются при входе и выходе из одной из них.

Заметим, что вес минимального остовного дерева не превосходит веса минимального гамильтонова цикла, поскольку при удалении одного из ребер гамильтонова цикла будет получено остовное дерево. Отсюда, в частности, следует, что вес построенного эйлерова цикла не более чем в 2 раза превосходит решение задачи коммивояжёра.

Теперь для доказательства оценки достаточно показать, что вес построенного гамильтоного цикла не превосходит веса эйлерова цикла. Это выполнено в силу обобщения неравенства треугольника на несколько вершин: одно ребро, соединяющее вершины, обязано иметь меньший вес, чем любой другой путь между ними в графе. ■

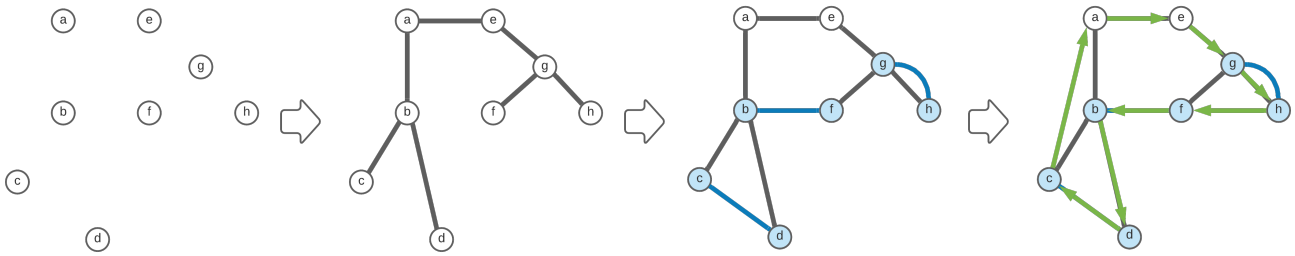
3.2 Анализ сложности работы

Будем считать, что минимальное остовное дерево строится с помощью агоритма Прима, который имеет временную сложность $O(E \log V)$ при использовании бинарной кучи и $O(E + V \log V)$ при использовании фибоначчиевой кучи. Удвоение ребер и поиск эйлерова цикла не влияют на асимптотику, поскольку для этих операций достаточно одного обхода дерева за $O(V + E)$. Для составления гамильтонова цикла также достаточно линейного по размеру графа числа операций. Итоговая сложность составляет $O(E + V \log V)$, а с учетом того, что граф полный — $O(V^2)$.

4 Алгоритм 1.5-приближения

4.1 Описание и обоснование корректности

1. Построим по исходному графу минимальное остовное дерево
2. Построим новый неориентированный граф, полученный добавлением к минимальному остовному дереву ребер совершенного паросочетания минимального веса на вершинах исходного графа, которые в минимальном остовном дереве имеют нечетную степень
3. Построим эйлеров цикл в новом графе
4. Упорядочим вершины в порядке первого появления при некотором обходе этого цикла
5. Простой цикл, образованный упорядоченными вершинами, и будет искомым



Построенный в пункте 2 граф будет эйлеровым в силу того, что все его вершины имеют нечетные степени.

Теорема. *Построенный алгоритмом гамильтонов цикл будет* $\frac{3}{2}$*-приближением решения задачи коммивояжёра.*

Доказательство. Обозначим через W_{opt} решение задачи коммивояжера, а через V' — множество вершин, которые в минимальном остовном дереве имеют нечетную степень. Поскольку сумма степеней вершин в графе чётна, то V' состоит из четного числа вершин, а значит, максимальное паросочетание в полном графе на вершинах V' будет совершенным.

Аналогично рассмотренному алгоритму 2-приближения вес минимального остовного дерева не превосходит W_{opt}. Для доказательства оценки приближения достаточно показать, что минимальный вес совершенного паросочетания на $V' \leqslant W_{opt}/2$.

Рассмотрим гамильтонов цикл минимального веса. Последовательно исключим из него вершины из $V \backslash V'$, заменяя в цикле связку $e_{ij} - v_j - e_{jk}$ на e_{ik}, где e_{ij}—ребро между вершинами v_i и v_j. Вес цикла при этом только уменьшится в силу выполнения неравенства треугольника. Из полученного цикла 2-мя способами можно выделить совершенное паросочетание на множестве V', оставив ребра цикла «через одно». Вес хотя бы одного из этих паросочетаний не будет превосходить $W_{opt}/2$. Отсюда вес совершенного паросочетания на V' минимального веса не превосходит $W_{opt}/2$ и оценка доказана. ■

Рассмотренный алгоритм также известен, как *алгоритм Кристофидеса.*

4.2 Анализ сложности работы

По аналогии с 2-приближением решения задачи коммивояжёра, для нахождения минимального остовного дерева воспользуемся *алгоритмом Прима*, имеющим сложность $O(E \log V)$ при использовании бинарной кучи и $O(E + V \log V)$ с фибоначчиевой кучей. Для поиска максимального паросочетания минимального веса используем *алгоритм Эдмондса* (так же известный как *алгоритм сжатия цветков*). Последний имеет сложность $O(EV^2)$, а при использовании модификации Микали и Вазирани $O(EV^{1/2})$. Остальные шаги алгоритма не влияют на асимптотику, поскольку требуют обхода графа в глубину за линейное время. Итоговая сложность по времени составляет $O(EV^{1/2})$, что для полного графа означает $O(V^{5/2})$.

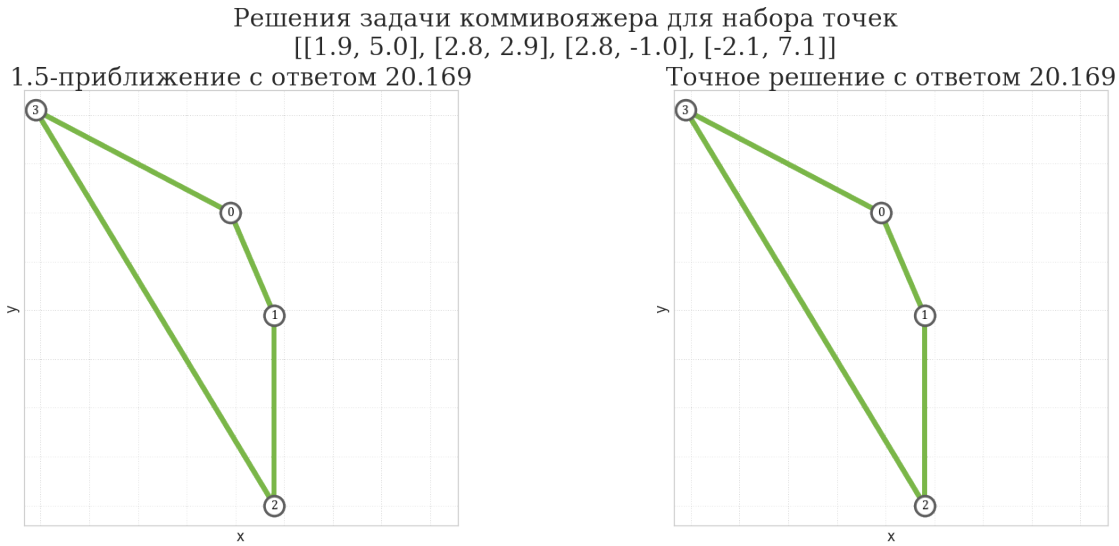
4.3 Особенности реализации

Для хранения и работы с графом используем библиотеку `networkx` языка Python. Для изображения графов и построения графиков сравнения скорости работы точного алгоритма и реализации алгоритма Кристофидеса воспользуемся библиотекой `matplotlib`.

4.4 Результаты тестовых запусков

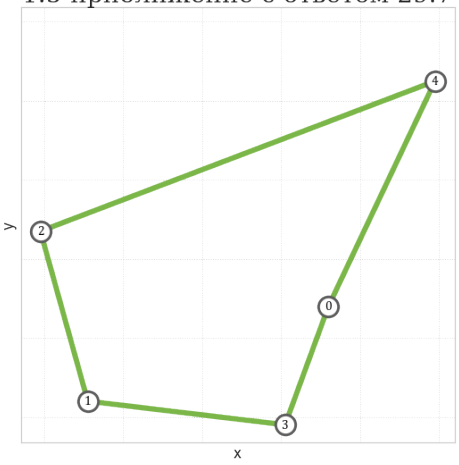
Для исследования реализации алгоритма использовались 3 набора данных. Первый из них, с малым числом точек (от 4 до 10) — для визуализации и наблюдения, на каких данных алгоритм работает точнее. Второй, с графами бóльших размеров — для сравнения времени работы алгоритма с наивной реализацией в зависимости от числа вершин. В качестве третьего набора использованы реальные данные о городах некоторых стран для демонстрации результата и времени работы на больших данных.

Точки первого набора были сгенерированы с округлением из многомерного нормального измерения с помощью библиотеки `scipy.stats`. Для получения точного решения задачи коммивояжёра использовался алгоритм полного перебора, работающий за экспоненциальное время в худшем случае.

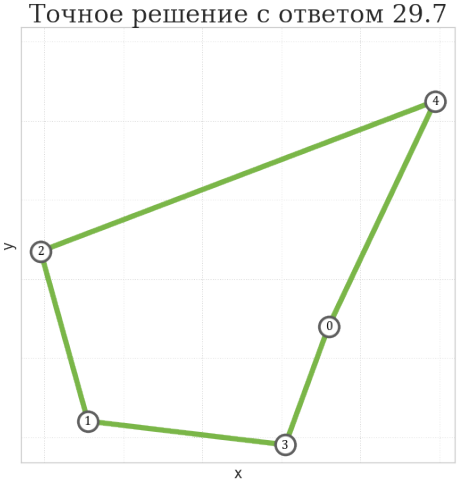


Решения задачи коммивояжера для набора точек
[[5.2, 4.8], [-0.9, 2.4], [-2.1, 6.7], [4.1, 1.8], [7.9, 10.5]]

1.5-приближение с ответом 29.7

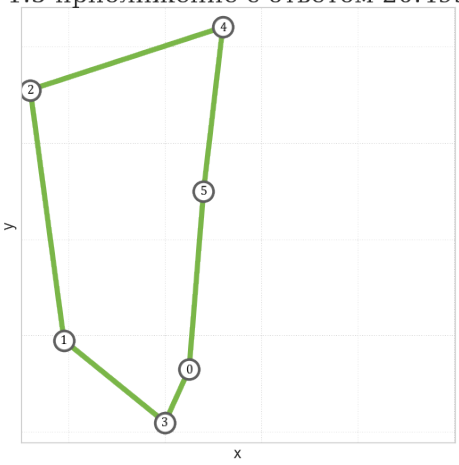


Точное решение с ответом 29.7

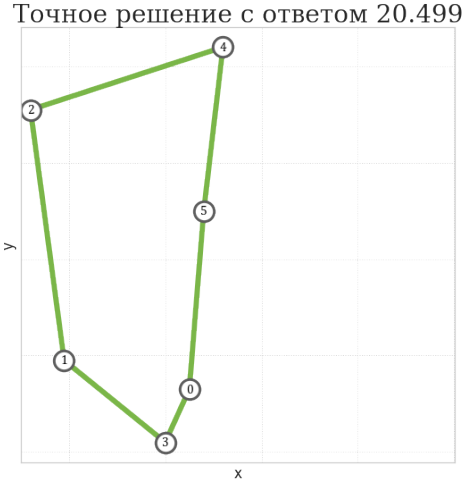


Решения задачи коммивояжера для набора точек
[[4.5, 1.3], [1.9, 1.9], [1.2, 7.1], [4.0, 0.2], [5.2, 8.4], [4.8, 5.0]]

1.5-приближение с ответом 20.499

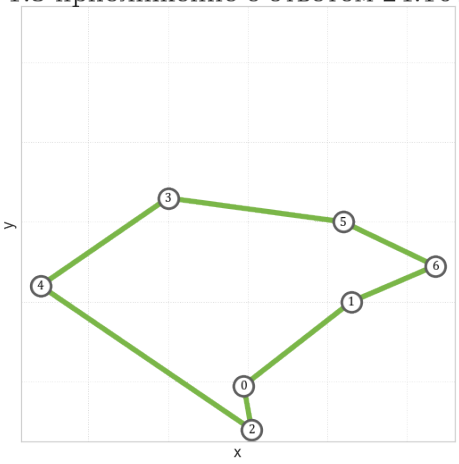


Точное решение с ответом 20.499

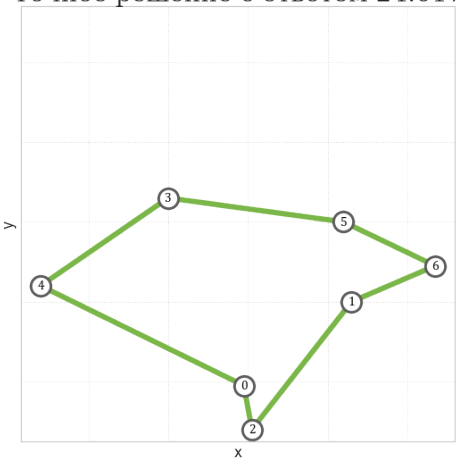


Решения задачи коммивояжера для набора точек
[[3.9, 1.9], [6.6, 4.0], [4.1, 0.8], [2.0, 6.6], [-1.2, 4.4], [6.4, 6.0], [8.7, 4.9]]

1.5-приближение с ответом 24.104

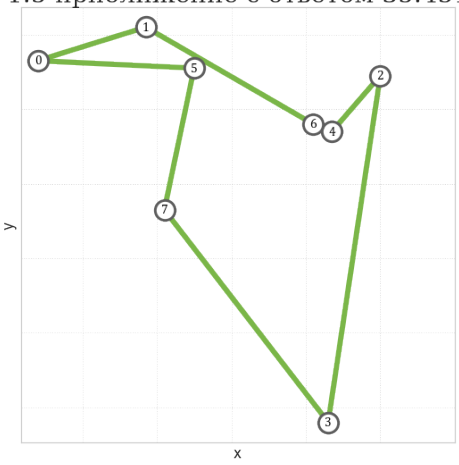


Точное решение с ответом 24.017

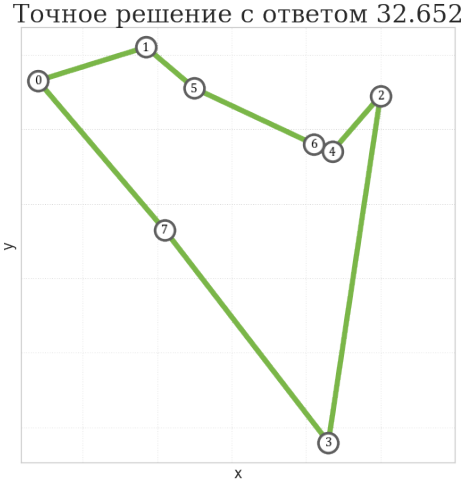


Решения задачи коммивояжера для набора точек
[[-3.2, 5.3], [-0.3, 6.2], [6.0, 4.9], [4.6, -4.4], [4.7, 3.4], [1.0, 5.1], [4.2, 3.6], [0.2, 1.3]]

1.5-приближение с ответом 35.451

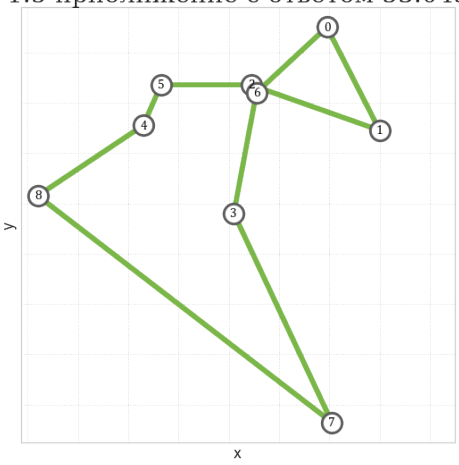


Точное решение с ответом 32.652

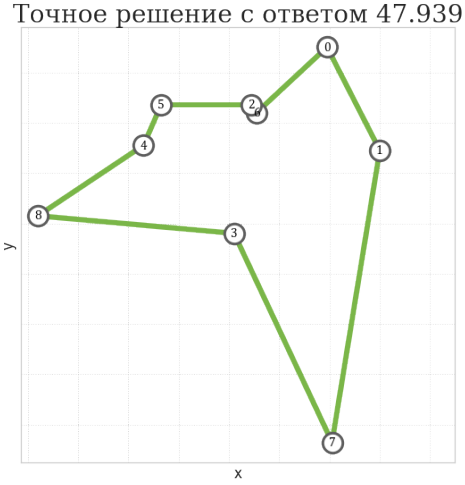


Решения задачи коммивояжера для набора точек
[[3.9, 7.0], [6.0, 2.9], [0.9, 4.7], [0.2, -0.4], [-3.4, 3.1], [-2.7, 4.7], [1.1, 4.4], [4.1, -8.7], [-7.6, 0.3]]

1.5-приближение с ответом 53.045

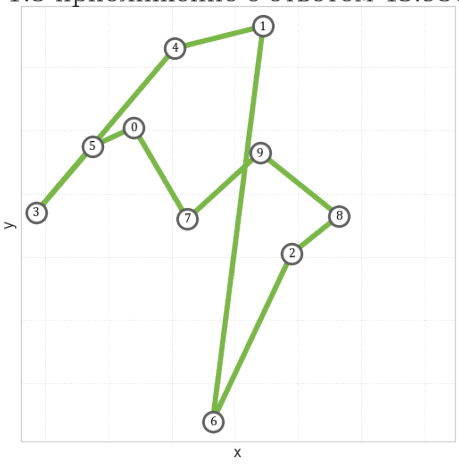


Точное решение с ответом 47.939

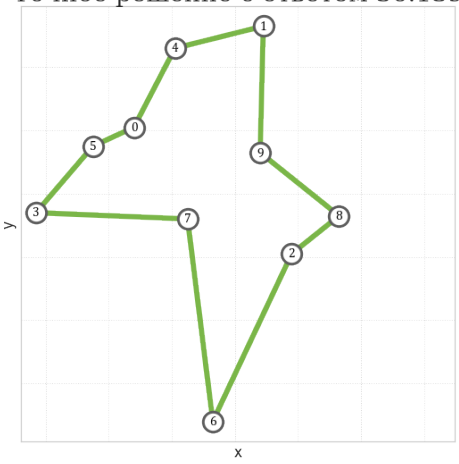


Решения задачи коммивояжера для набора точек
[[0.8, 4.1], [4.9, 7.3], [5.8, 0.1], [-2.3, 1.4], [2.1, 6.6], [-0.5, 3.5], [3.3, -5.2], [2.5, 1.2], [7.3, 1.3], [4.8, 3.3]]

1.5-приближение с ответом 43.956



Точное решение с ответом 36.139

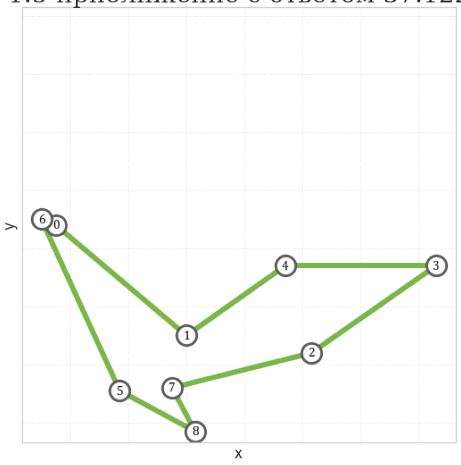


Полученные результаты свидетельствуют о том, что алгоритм Кристофидеса работает точнее на графах, у которых все или почти все вершины лежат на выпуклой оболочке (среди рассмотренных графов это с 1-го по 3-й). Последние 2 графа отражают, что чем больше вершин попадают внутрь выпуклой оболочки, тем менее точным получается ответ. Такое поведение можно объяснить тем, что если почти все вершины попадают на выпуклую оболочку, то минимальное остовное дерево, которое строит алгоритм, будет содержать большую часть ребер выпуклой оболочки, которая довольно точно приближает правильное решение.

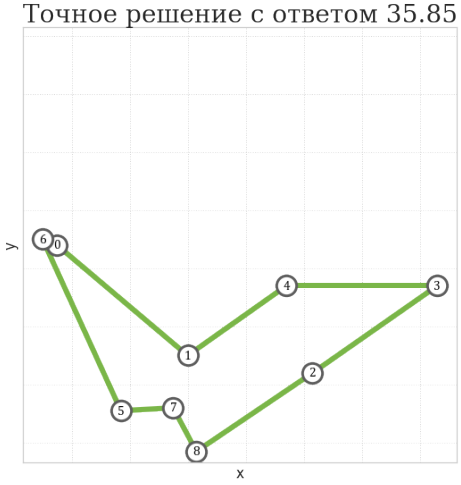
Рассмотрим, как меняется точное решение и 1.5-приближение алгоритма Кристофидеса в зависимости от использованной метрики для двух наборов точек (по порядку — для Евклидовой метрики $d(x,y) = \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2}$, Манхэттенской $d(x,y) = \sum_{i=1}^{d} |x_i - y_i|$ и экстремальной $d(x,y) = \max_{i \in \overline{1,d}} |x_i - y_i|$).

Решения задачи коммивояжера для набора точек
[[-0.5, 6.8], [4.0, 3.0], [8.3, 2.4], [12.6, 5.4], [7.4, 5.4], [1.7, 1.1], [-1.0, 7.0], [3.5, 1.2], [4.3, -0.3]]

1.5-приближение с ответом 37.122

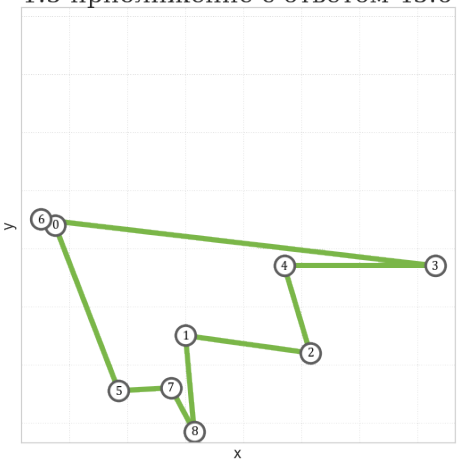


Точное решение с ответом 35.85

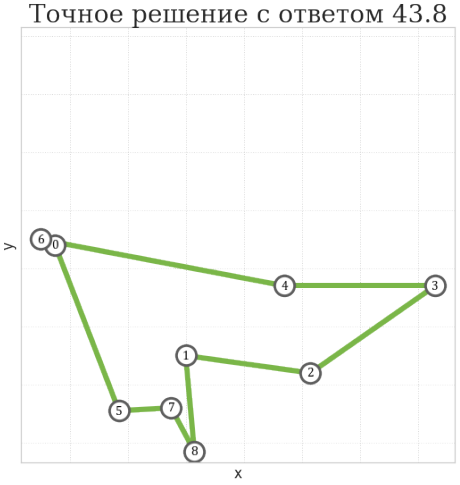


Решения задачи коммивояжера для набора точек
[[-0.5, 6.8], [4.0, 3.0], [8.3, 2.4], [12.6, 5.4], [7.4, 5.4], [1.7, 1.1], [-1.0, 7.0], [3.5, 1.2], [4.3, -0.3]]

1.5-приближение с ответом 45.6

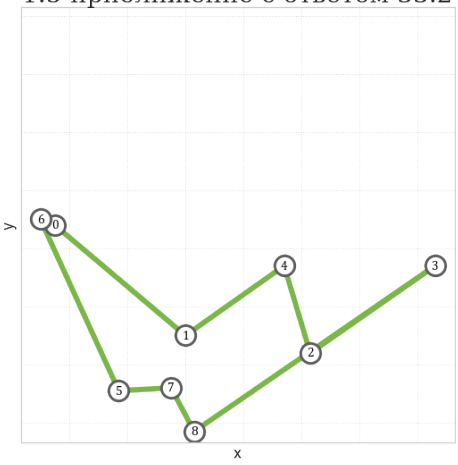


Точное решение с ответом 43.8

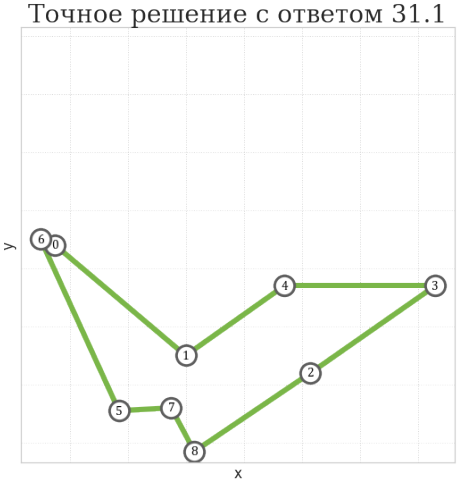


Решения задачи коммивояжера для набора точек
[[-0.5, 6.8], [4.0, 3.0], [8.3, 2.4], [12.6, 5.4], [7.4, 5.4], [1.7, 1.1], [-1.0, 7.0], [3.5, 1.2], [4.3, -0.3]]

1.5-приближение с ответом 33.2

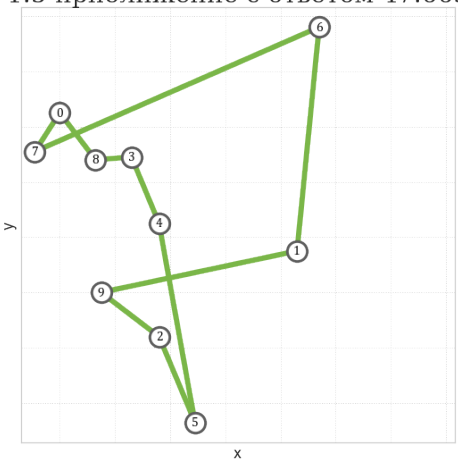


Точное решение с ответом 31.1

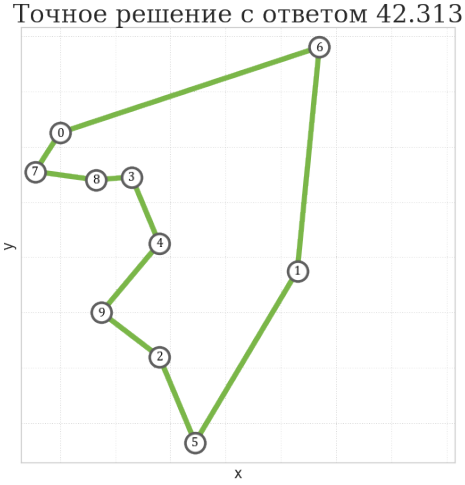


Решения задачи коммивояжера для набора точек
-2.0, 6.5], [6.6, 1.5], [1.6, -1.6], [0.6, 4.9], [1.6, 2.5], [2.9, -4.7], [7.4, 9.6], [-2.9, 5.1], [-0.7, 4.8], [-0.5, -0.0

1.5-приближение с ответом 47.663

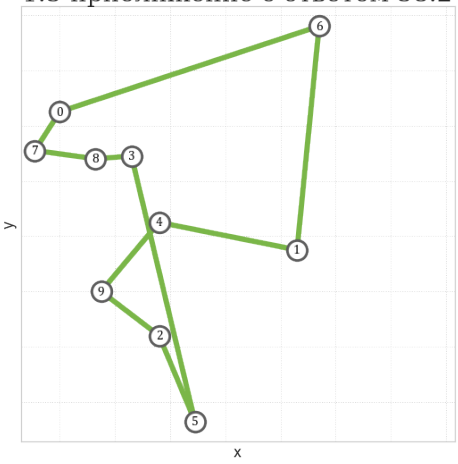


Точное решение с ответом 42.313

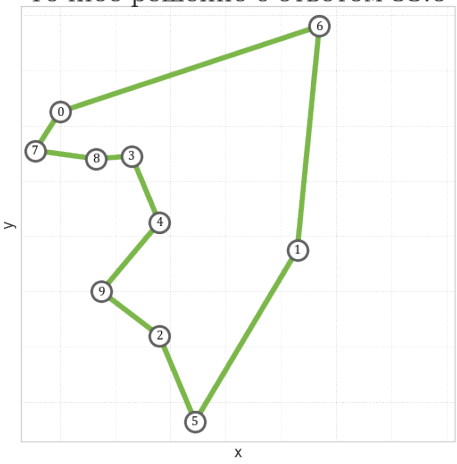


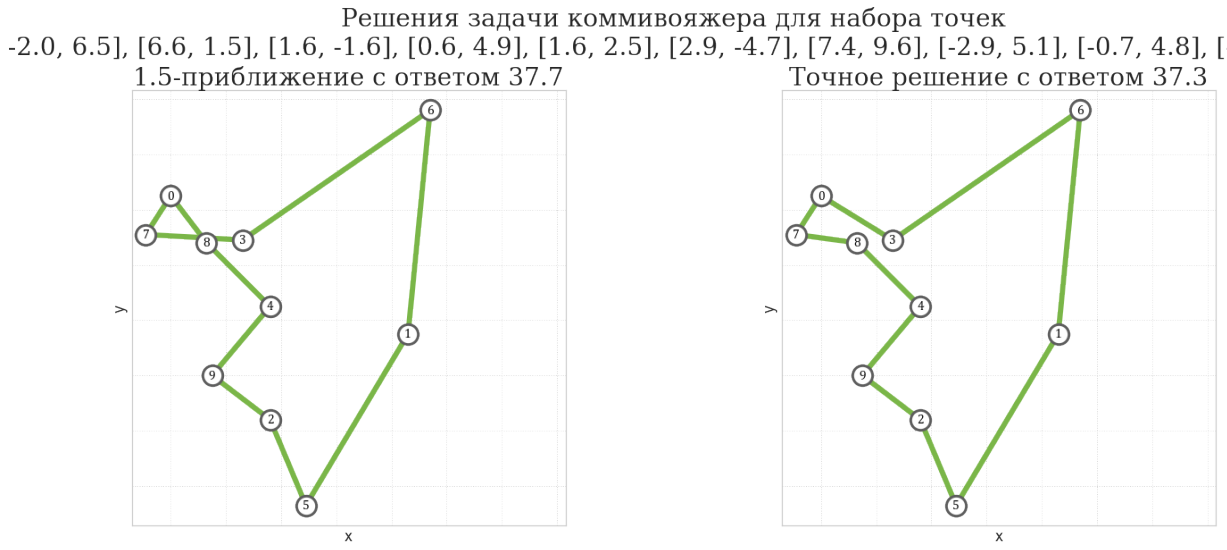
Решения задачи коммивояжера для набора точек
-2.0, 6.5], [6.6, 1.5], [1.6, -1.6], [0.6, 4.9], [1.6, 2.5], [2.9, -4.7], [7.4, 9.6], [-2.9, 5.1], [-0.7, 4.8], [-0.5, -0.0

1.5-приближение с ответом 58.2



Точное решение с ответом 53.6





Рассмотренные примеры отражают, что при изменении метрики на одном и том же наборе точек на плоскости меняется как точное решение задачи коммивояжёра, так и его приближение алгоритмом Кристофидеса. В частности, для обоих наборов точек приближение дало разные результаты в зависимости от метрики.

Результаты запуска на втором наборе данных:

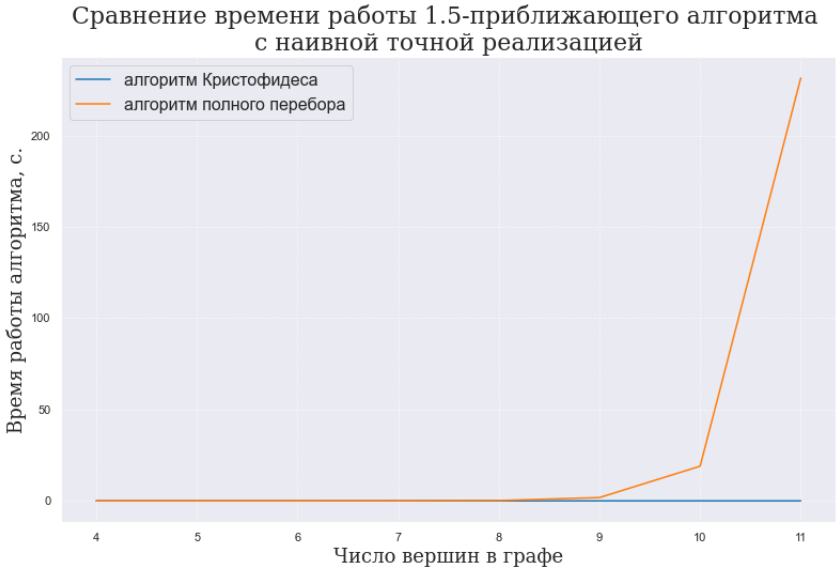
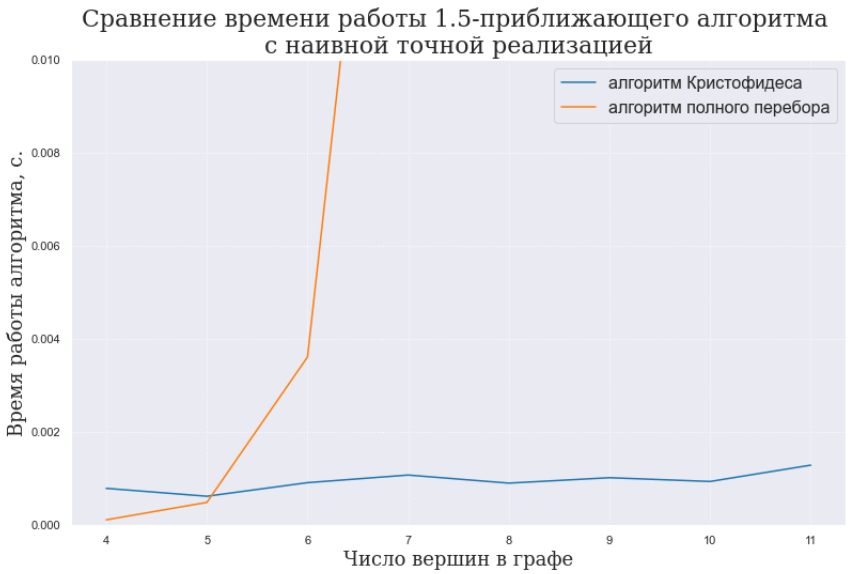
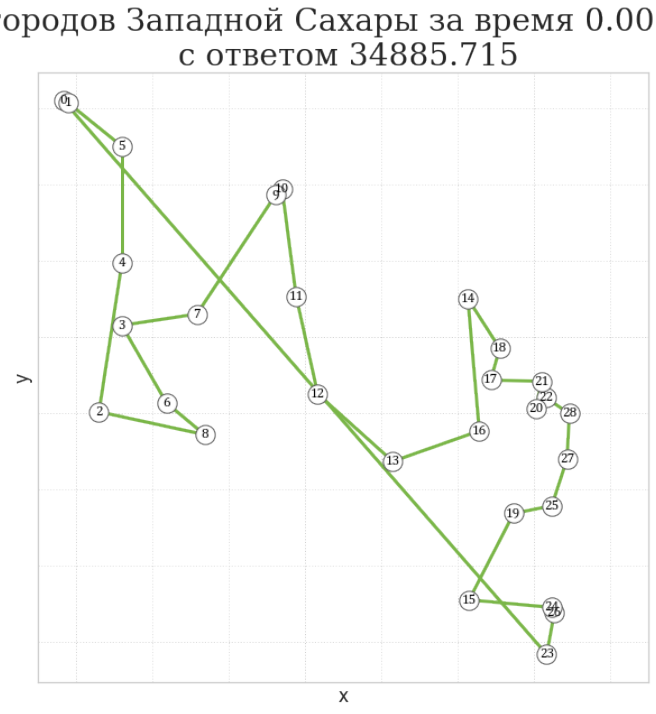


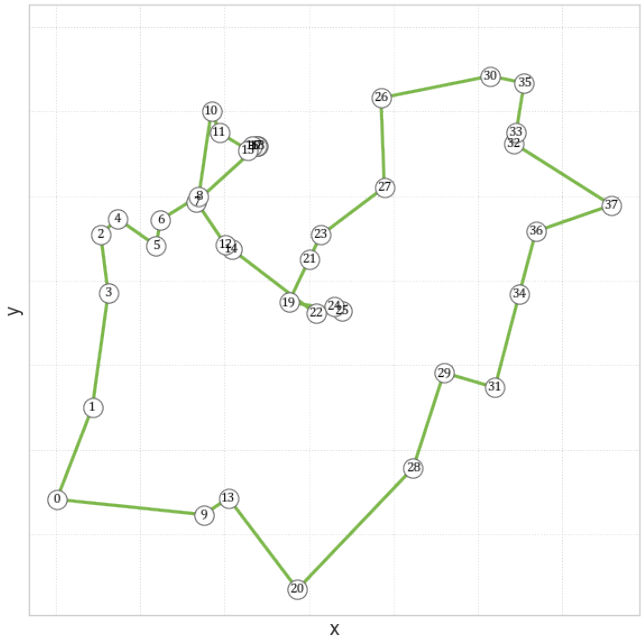
График времени работы приближающего алгоритма сливается в прямую линию около 0, поэтому для определения наименьшего размера графа, на котором наблюдается первое существенное отличие, график был ограничен вдоль оси Oy:



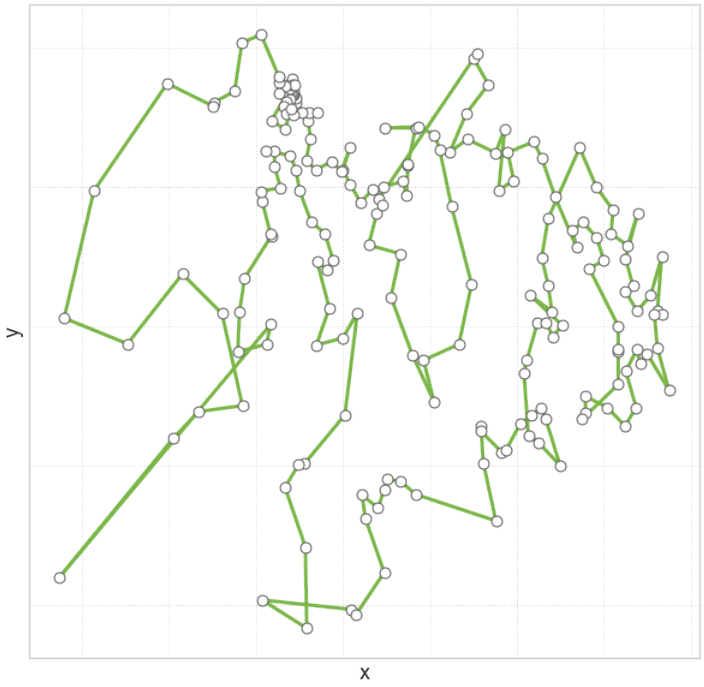
Результаты запуска на третьем наборе данных: от 29 городов в Западной Сахаре до 1979 городов в Омане.



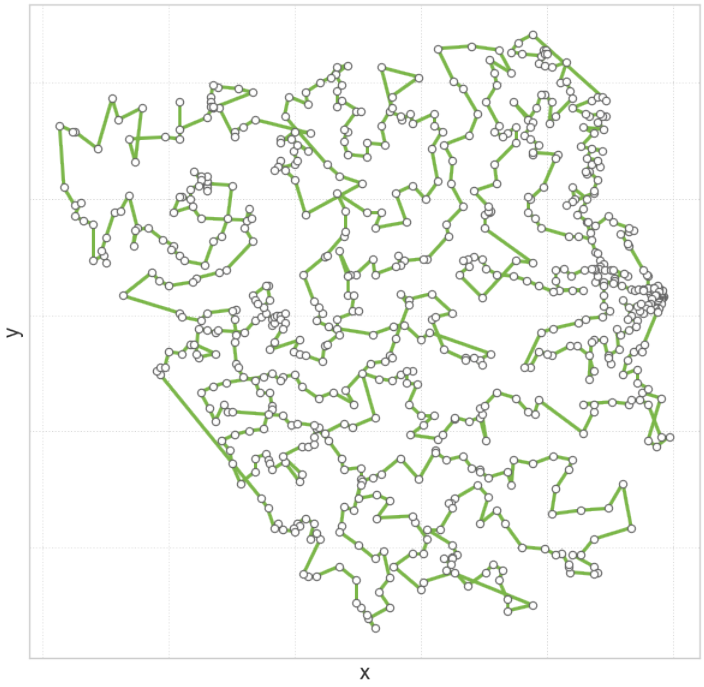
38 городов Джибути за время 0.009 сек.
с ответом 6870.373

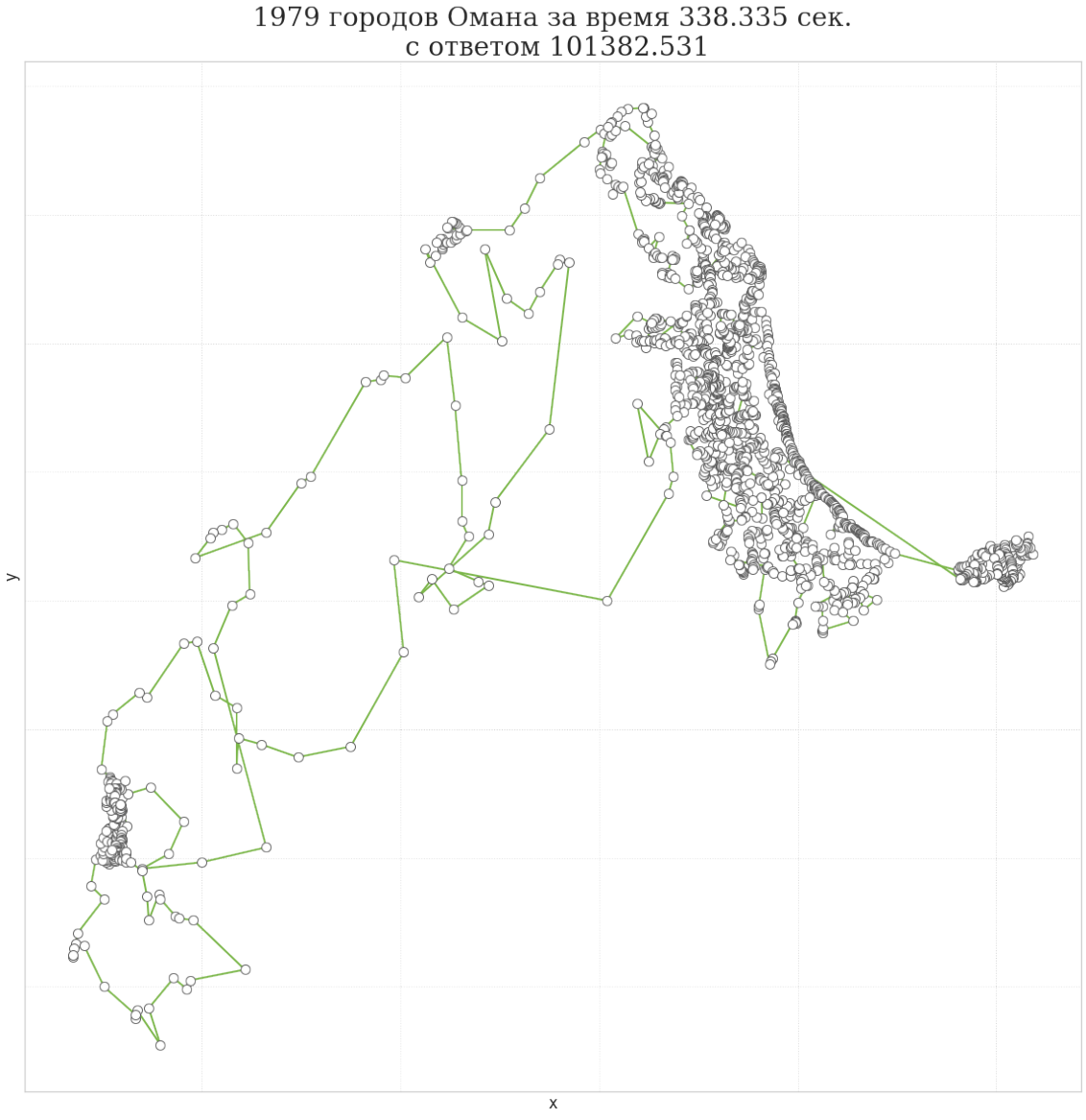


194 города Катара за время 0.752 сек.
с ответом 10685.757



734 города Уругвая за время 25.728 сек.
с ответом 90321.522





Несмотря на то что рассмотренный алгоритм работает за полиномиальное время, третий тестовый набор показывает, что на практике степень многочлена в асимптотике играет ощутимую роль: уже при числе городов порядка 2000 приближающий алгоритм работает более 6 минут. Поэтому на практике часто задействуют алгоритмы, использующие некоторые априорные знания о поставленной задаче, либо алгоритмы, которые оценивают вероятность попадания ребра в решение, как, например, алгоритм Литтла, являющийся частным случаем метода ветвей и границ.

5 Заключение

Несмотря на простоту формулировки и длинную историю, задача коммивояжёра, а именно её метрический вариант, исследуется и по наши дни. При этом алгоритм Кристофидеса до сих пор остается самым точным из известных для метрической задачи без дополнительных условий и ограничений.

6 Приложение: реализация алгоритма 1.5-приближения решения задачи

```
import numpy as np
import pandas as pd
import scipy.stats as sps
import networkx as nx
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import permutations
from matplotlib.font_manager import FontProperties
from tqdm.notebook import tqdm
import time


def get_adjacency_matrix(points, metric='euclidean'):
    '''Для заданного набора точек строит матрицу смежности в соответствии с
    указанной метрикой

    :params:
        - points - набор точек
        - metric - используемая метрика
    '''

    points = points[np.newaxis, :]
    diff = points - points.reshape((-1, 1, 2))
    if metric == 'euclidean':
        matrix = np.linalg.norm(diff, axis=-1)
    elif metric == 'manhattan':
        matrix = np.sum(np.abs(diff), axis=-1)
    elif metric == 'max':
        matrix = np.max(np.abs(diff), axis=-1)
    else:
        raise Exception('metric not found')

    return matrix
```

```
def get_odd_degree_vertices(graph):
    '''Находит в графе вершины с нечетными степенями

    :params:
        - graph - граф, наследник класса nx.Graph
    :return value:
        - vertex_indices - индексы вершин
    '''
    vertices, degrees = np.array(graph.degree).T
    return vertices[degrees % 2 == 1]


def find_min_weight_perfect_matching(graph):
    '''Находит в графе максимальное паросочетание минимального веса

    :params:
        - graph - граф, наследник класса nx.Graph
    :return value:
        - matching - ребра паросочетания
    '''

    second_graph = nx.Graph()
    for first, second, weight in graph.edges.data('weight'):
        second_graph.add_edge(first, second, weight=-weight)

    return nx.max_weight_matching(second_graph, maxcardinality=True)


def find_hamiltonian_path(points, metric='euclidean'):
    '''Решает 3/2-приближенную метрическую задачу коммивояжера с евклидовой
    метрикой с помощью алгоритма Кристофидеса

    :params:
        - points - координаты вершин в графе
        - metric - используемая метрика
    :return value:
        (vertices, length) - список индексов вершин в гамильтоновом пути
        наименьшего веса и длина этого пути
    '''

    adjacency_matrix = get_adjacency_matrix(points, metric)
    graph = nx.from_numpy_array(adjacency_matrix)

    minimum_spanning_tree = nx.minimum_spanning_tree(graph)
```

```
    odd_degree_vertices = get_odd_degree_vertices(minimum_spanning_tree)
    odd_degree_graph = nx.from_numpy_array(
        adjacency_matrix[odd_degree_vertices][:, odd_degree_vertices])

    min_weight_perfect_matching = find_min_weight_perfect_matching(
        odd_degree_graph)

    eulerian_graph = nx.MultiGraph()
    for first, second, weight in minimum_spanning_tree.edges(data='weight'):
        eulerian_graph.add_edge(first, second, weight=weight)

    for first, second in min_weight_perfect_matching:
        first = odd_degree_vertices[first]
        second = odd_degree_vertices[second]
        eulerian_graph.add_edge(first, second,
                                weight=adjacency_matrix[first][second])

    eulerian_cycle = nx.eulerian_circuit(eulerian_graph)

    used = np.zeros(shape=(points.shape[0]))
    length = 0
    result = []
    for v_from, v_to in eulerian_cycle:
        if len(result) == 0:
            result.append(v_from)
            used[v_from] = 1
        if used[v_to] == 0:
            length += adjacency_matrix[result[-1]][v_to]
            result.append(v_to)
            used[v_to] = 1
    length += adjacency_matrix[result[-1]][result[0]]

    return result, length


def find_min_weight_hamiltonian_path(points, metric='euclidean'):
    '''Находит точное решение задачи коммивояжёра алгоритмом полного перебора

    :params:
        - points - точки на плоскости
        - metric - используемая метрика

    :return value:
        - (vertices, length) - список индексов вершин в гамильтоновом пути
```

```
        наименьшего веса и длина этого пути
    '''

    adjacency_matrix = get_adjacency_matrix(points, metric)

    min_length = None
    min_cycle = None

    for permutation in permutations(range(len(points))):
        length = 0

        for i, first in enumerate(permutation):
            second = permutation[(i + 1) % len(permutation)]
            length += adjacency_matrix[first][second]

        if min_length is None or length < min_length:
            min_length = length
            min_cycle = permutation

    return min_cycle, min_length


def plot_hamiltonian_path(points, path, length, ax, title, font=None,
                          node_size=500, font_size=14, width=6, linewidths=3,
                          measured_time=None):
    '''Строит на графике гамильтонов цикл минимального веса для фиксированного
    набора точек

    :params:
        - points - набор точек на плоскости
        - path - список индексов вершин в гамильтоновом цикле
        - length - длина гамильтонова пути
        - ax - ось
        - title - заголовок
        - font - настройки шрифта, наследник класса FontProperties
        - node_size - размер вершины на графике
        - font_size - размер числа-метки вершины
        - width - ширина ребра
        - linewidths - толщина границы вершины
        - measured_time - время работы алгоритма
    '''

    adjacency_matrix = get_adjacency_matrix(points)

    cycle = nx.Graph()
```

```
    for i, first in enumerate(path):
        second = path[(i + 1) % len(path)]
        cycle.add_edge(first, second, weight=adjacency_matrix[first][second])

    with sns.plotting_context('notebook'), sns.axes_style('darkgrid'):
        if measured_time is not None:
            title += ' за время {} сек.\n'.format(round(measured_time, 3))
        ax.set_title(title + ' с ответом {}'.format(length.round(3)),
                     fontsize=28, fontproperties=font)

        nx.draw_networkx(cycle, pos=points, node_color='white', ax=ax,
                         edgecolors='#5E5E5E', linewidths=linewidths,
                         edge_color='#7AB648', font_family='Cambria',
                         node_size=node_size, font_size=font_size, width=width)

        ax.set_xlabel('x', fontsize=18)
        ax.set_ylabel('y', fontsize=18)
        ax.grid(ls=':', b=True)


def compare_solutions(points, metric='euclidean', filename=None):
    '''Рисует на графике 1.5-приближение решения задачи коммивояжера и точное
    решение

    :params:
        - points - набор точек
        - metric - используемая метрика
        - filename - имя файла для сохранения графика
    '''

    correct_path, correct_length = find_min_weight_hamiltonian_path(
        points, metric)
    path, length = find_hamiltonian_path(points, metric)

    with sns.plotting_context('notebook'), sns.axes_style('whitegrid'):
        fig = plt.figure(figsize=(20.7, 8.9))

        font = FontProperties(family='serif', size=28)

        fig.suptitle('Решения задачи коммивояжера для набора точек\n'
                     '{}'.format(points.tolist()), y=0.97,
                     fontproperties=font)

        ax_left = plt.subplot(1, 2, 1, sharex=plt.gca(), sharey=plt.gca())
        ax_left.axis('square')
```

```
        plot_hamiltonian_path(points, path, length, ax_left, '1.5-приближение',
                              font)

        ax_right = plt.subplot(1, 2, 2, sharex=plt.gca(), sharey=plt.gca())
        ax_right.axis('square')

        plot_hamiltonian_path(points, correct_path, correct_length, ax_right,
                              'Точное решение', font)

        fig.tight_layout(rect=[0, 0.01, 1, 1])

        if filename is not None:
            fig.savefig(filename, transparent=True)


'''Генерация и обработка первого тестового набора'''
mean = np.array([3, 3])
cov = np.array([[10, 0],
                [0, 10]])

for i, sample_size in enumerate(tqdm(np.arange(4, 11))):
    sample = sps.multivariate_normal(mean, cov).rvs(size=sample_size).round(1)
    compare_solutions(sample, 'small{}.png'.format(i + 1))

for i, sample_size in enumerate(tqdm(np.arange(9, 11))):
    sample = sps.multivariate_normal(mean, cov).rvs(size=sample_size).round(1)
    compare_solutions(sample, metric='euclidean',
                      filename='eu_metric{}.png'.format(i + 1))
    compare_solutions(sample, metric='manhattan',
                      filename='ma_metric{}.png'.format(i + 1))
    compare_solutions(sample, metric='max',
                      filename='max_metric{}.png'.format(i + 1))


'''Генерация и обработка второго тестового набора'''
mean = np.array([3, 3])
cov = np.array([[100, 0],
                [0, 100]])

num_points = np.arange(4, 12)
time_correct = np.array([])
time_approx = np.array([])

for sample_size in tqdm(num_points):
```

```
    sample = sps.multivariate_normal(mean, cov).rvs(size=sample_size).round(1)

    start = time.time()
    path, length = find_hamiltonian_path(sample)
    time_approx = np.append(time_approx, time.time() - start)

    start = time.time()
    path_correct, path_length = find_min_weight_hamiltonian_path(sample)
    time_correct = np.append(time_correct, time.time() - start)

num_points = np.ones_like(time_approx).cumsum() + 3

with sns.plotting_context('notebook'), sns.axes_style('darkgrid'):
    plt.figure(figsize=(13, 8))

    font = FontProperties(family='serif', size=28)

    plt.title('Сравнение времени работы 1.5-приближающего алгоритма \n'
              'с наивной точной реализацией', fontsize=22, fontproperties=font)

    plt.plot(num_points, time_approx, label='алгоритм Кристофидеса')

    plt.plot(num_points, time_correct, label='алгоритм полного перебора')

    plt.ylim((0, 0.01))
    plt.xlabel('Число вершин в графе', fontsize=18, fontproperties=font)
    plt.ylabel('Время работы алгоритма, с.', fontsize=18, fontproperties=font)
    plt.legend(fontsize=16)
    plt.grid(ls=':', b=True)
    plt.savefig('speed2.png')


'''Пример обработки графа из третьего тестового набора'''
cities = pd.read_csv('./data/wi29.tsp', sep=' ', skiprows=8,
                     names=['id', 'x', 'y'], skipfooter=1, engine='python',
                     index_col='id')

points = cities.to_numpy()

start = time.time()
path, length = find_hamiltonian_path(points)
measured_time = time.time() - start

with sns.plotting_context('notebook'), sns.axes_style('whitegrid'):
    fig = plt.figure(figsize=(10, 10))
```

```
font = FontProperties(family='serif', size=28)

plot_hamiltonian_path(points, path, length, ax=plt.gca(),
                      title='29 городов Западной Сахары', font=font,
                      node_size=300, font_size=12, width=3, linewidths=1,
                      measured_time=measured_time)
plt.axis('square')

fig.savefig('29cities.png', transparent=True)
```

7 Список источников

[1] *Кононов А.В., Кононова П.А*, Приближённые алгоритмы для NP-трудных задач, РИЦ НГУ (2014)

[2] *Куликов А.*, Курс «Алгоритмы для **NP**-трудных задач», лекция 6: приближенные алгоритмы, Computer Science Club при ПОМИ РАН

[3] Решение задачи коммивояжера с помощью метода ветвей и границ, https://habr.com/ru/post/246437/

[4] Решение задачи коммивояжера алгоритмом Литтла с визуализацией на плоскости, https://habr.com/ru/post/332208/

[5] TSP datasets, University of Waterloo website: http://www.math.uwaterloo.ca/tsp/data/index.html