Московский физико-технический институт (ГУ)
Физтех-школа прикладной математики и информатики
Сложность вычислений, весна 2023

Метрическая задача коммивояжёра-2

Велегурина Елена
Б05-121

Долгопрудный, 2023

Содержание

1 Введение

Данная метрическая задача коммивояжёра (далее MTSP) заключается в поиске гамильтонова пути минимального веса между двумя фиксированными вершинами s и t во взвешенном графе $G = (V, E)$ с весами рёбер $w : E \to \mathbb{R}_+$. В метрическом случае граф полный, а функция весов метрическая (то есть для любых трёх вершин x, y и z выполнено $w(x, z) \leqslant w(x, y) + w(y, z)$).

Эта задача является NP-трудной, поэтому не может быть достаточно быстро решена на большом числе вершин. Уже при числе вершин, превосходящем 66, переборные алгоритмы будут работать миллиарды лет. Поэтому начали появляться алгоритмы оптимизации, находящие приближённые решения.

Лучшей оценки достигли Никос Кристофидес и Анатолий Иванович Сердюков, которые независимо друг от друга нашли решение в 1976. Алгоритм Кристофидеса-Сердюкова является аппроксимационным алгоритмом, который гарантирует, что решения находятся в пределах 3/2 от длины оптимального решения. Известны лучшие приближения только для некоторых специальных случаев.

2 MTSP $\in$ **NPh**

В данной секции будет доказана NP-трудность метрической задачи коммивояжёра.

Теорема 1. MTSP $\in$ **NPh**.

Доказательство. Для доказательства воспользуемся следующим известным фактом: UHAMPATH $\in$ **NPh**, где UHAMPATH $= \{(\mathsf{G}, \mathsf{s}, \mathsf{t}) \mid$ в неориентированном графе G есть путь из s в t, проходящий ровно 1 раз через каждую вершину$\}$.

Сведём задачу о гамильтоновом пути UHAMPATH к задаче MTSP, построим новый граф $G' = (V, E, w)$, где w принимает значение 1 для любых рёбер из E.

При таком построении функция w' будет удовлетворять $w(x, z) \leqslant w(x, y) + w(y, z)$.

Таким образом, поиск гамильтонова пути в исходном графе будет эквивалентен поиску гамильтонова пути минимального веса во взвешенном графе.

□

3 Описание работы алгоритма 5/3-приближение

В данной секции описан один из возможных методов оптимизации задачи MTSP - алгоритм 5/3 приближения.

1. Построение минимального остовного дерева (далее MST) алгоритмом Крускала использованием системы непересекающихся множеств(DSU):

   - Отсортируем все рёбра по неубыванию веса.

- Поместим каждую вершину в своё дерево (т.е. своё множество) - на это уйдёт в сумме $O(N)$.
- Перебираем все рёбра (в порядке сортировки) и для каждого ребра за $O(1)$ определяем, принадлежат ли его концы разным деревьям.
- Наконец, объединение двух деревьев с вершинами в разных концах будет осуществляться также за $O(1)$.
- Завершаем работу алгоритма, когда все вершины окажутся в одном дереве.

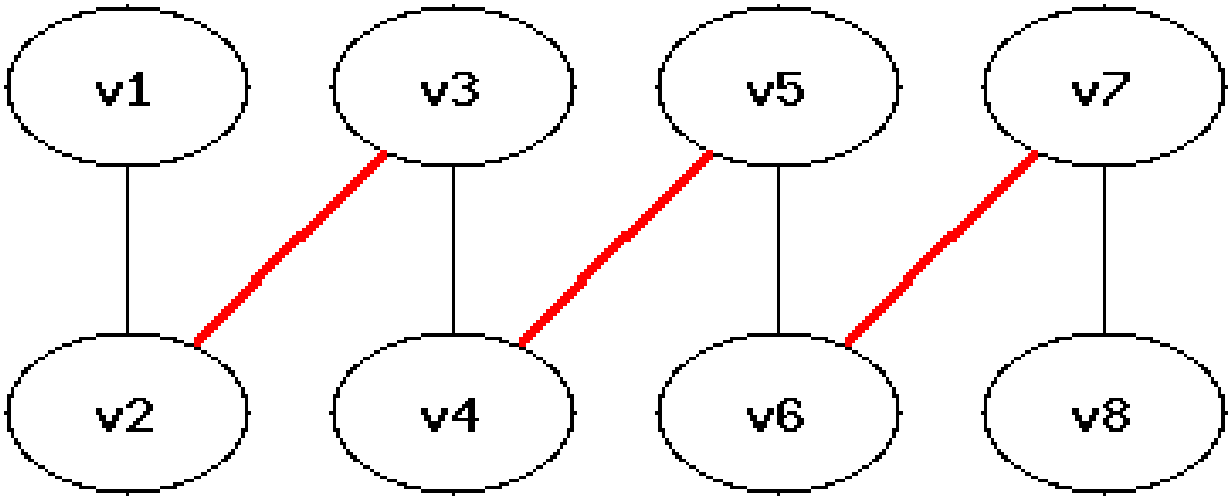
Итого мы получаем асимптотику $O(M \log N + N + M) = O(M \log N)$.

2. Затем сформируем список вершин, содержащий вершины нечётных степеней (для $v \neq s$ и $v \neq t$, и вершины s и t, если их степени чётны.

3. Воспользуемся алгоритмом Эдмонса-Карпа для построения совершенного паросочетания Match в выделенном подграфе. Введём несколько теорем и определений для лучшего описания алгоритма.

Определение 1. *Чередующаяся цепь - простая цепь, для любых двух соседних рёбер которой верно, что одно из них принадлежит паросочетанию, а другое нет.*

Определение 2. *Чередующаяся цепь называется увеличивающей, если её первая и последняя вершины не принадлежат паросочетанию.*

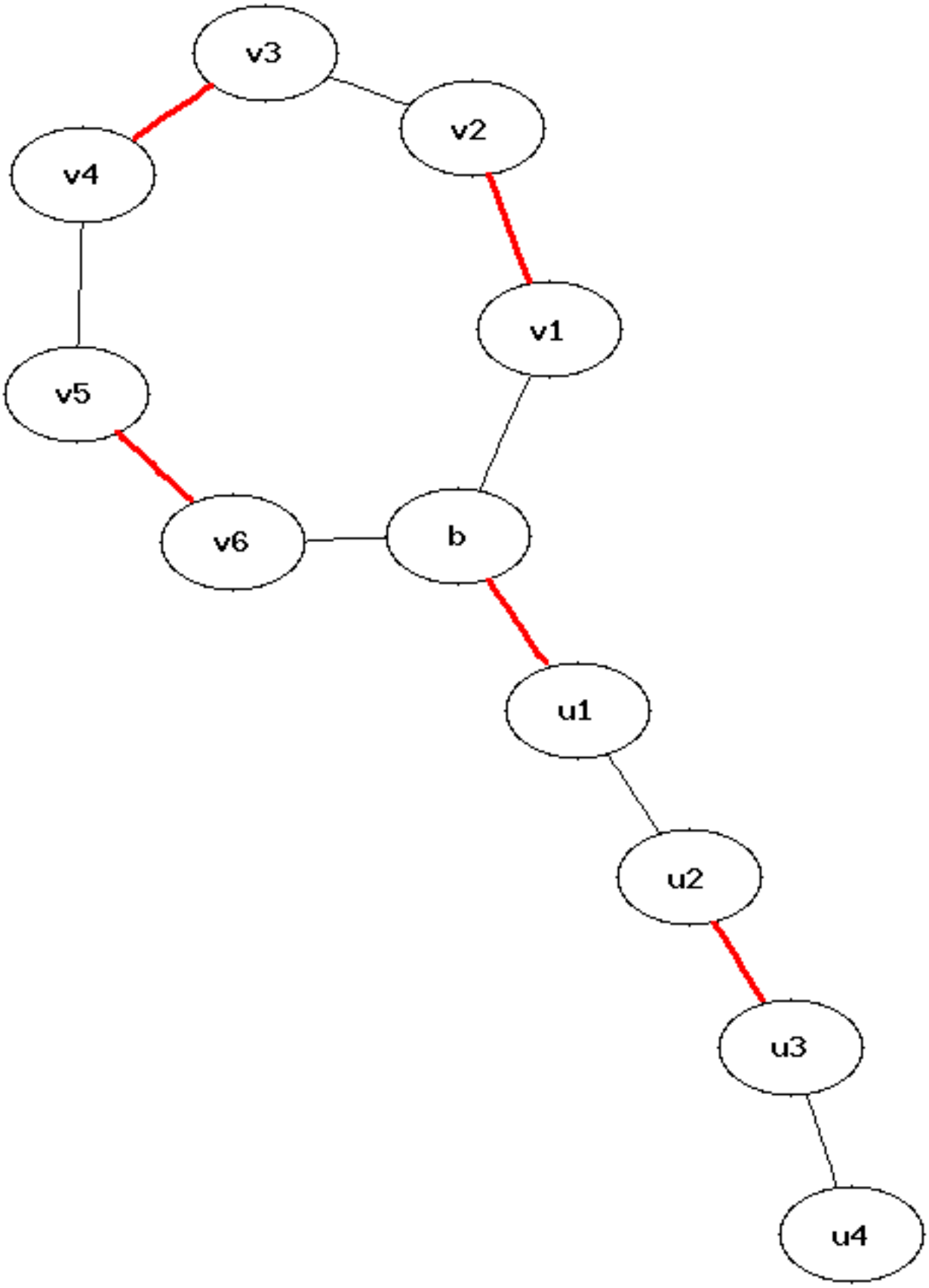


Теорема 2. *Теорема Бержа. Паросочетание является наибольшим тогда и только тогда, когда для него не существует увеличивающей цепи.*

Теорема 3. *Теорема Эдмондса. В графе $\overline{G}$ существует увеличивающая цепь тогда и только тогда, когда существует увеличивающая цепь в G.*

Определение 3. *Цветок - подграф, образованный циклом нечётной длины.*

Определение 4. *Вершину в цветке, не насыщенную рёбрами цикла, назовём стеблем.*



Определение 5. *Сжатие цветка – это сжатие всего нечётного цикла в одну псевдо-вершину (соответственно, все рёбра, инцидентные вершинам этого цикла, становятся инцидентными псевдо-вершине).*

Теперь опишем работу самого алгоритма:

- Найдём все цветки в графе.
- Сожмём их.
- После предыдущего шага не осталось циклов нечётной длины. Теперь алгоритмом DFS ищем увеличивающуюся цепь.
- Разворачиваем сжатые цветки, восстанавливая цепь в исходном графе

4. Строим новый граф, состоящий из рёбер MST и Match. По построению граф – эйлеров с началом s и концом t.

5. Обойдём построенный граф и получим эйлеров путь.

6. Гамильтонов путь будет проходить по вершинам исходного графа в порядке, полученном при эйлеровом обходе в предыдущем пункте.

4 Доказательство оценки 5/3 на вес пути

Построенный гамильтонов путь обозначим $path$, а оптимальный – opt_path. Пусть $w(path)$ возвращает вес пути.

Факт 1. *Из условия на функцию в метрическом пространстве* $w(x, z) \leqslant w(x, y) + w(y, z)$ *следует, что* $w(path) \leqslant w(\mathsf{MST}) + w(\mathsf{Match})$.

Лемма 1. $w(\mathsf{MST}) \leqslant w(opt\_path)$.

Доказательство. Очевидно из определения MST. □

Лемма 2. $w(\mathsf{Match}) \leqslant \frac{2}{3} w(opt\_path)$.

Доказательство. Верно из построения графа. □

Теорема 4. *Вес построенного гамильтонова пути* $path$ *не превосходит* $\frac{5}{3}$ *веса оптимального пути* opt_path.

Доказательство. Воспользуемся доказанными леммами и фактом.

$$w(path) \leqslant w(\mathsf{MST}) + w(\mathsf{Match}) \leqslant w(opt\_path) + \frac{2}{3} w(opt\_path) = \frac{5}{3} w(opt\_path)$$

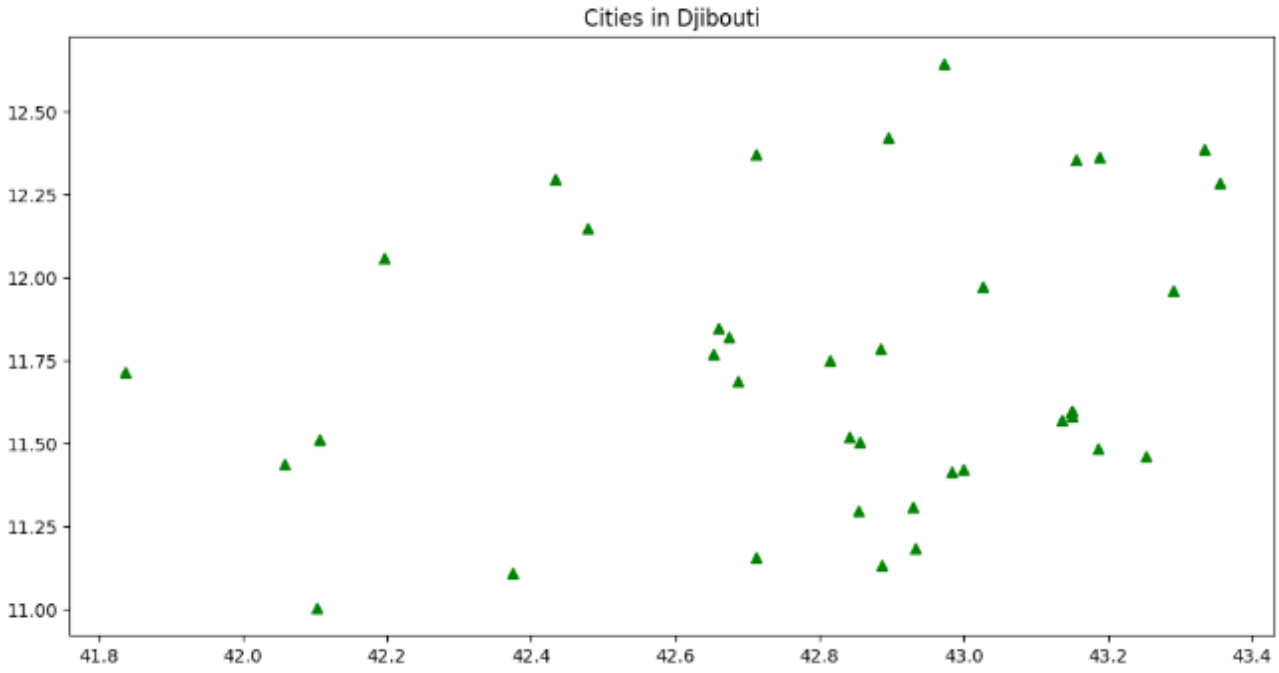
□

5 Реализация и применение описанного алгоритма

Будем тестировать алгоритм на основе реальных данных. Возьмём их с сайта https://math.uwaterloo.ca/tsp/world/countries.html. Он предоставляет координаты городов стран и их оптимальный обход.

Будем рассматривать Джибути. В нём 38 городов.

```
x, y = [], []
i = 0
for line in open('dj38.tsp', 'r').readlines():
    if i < 10:
        i += 1
    else:
        i += 1
        line = line.strip()
        str = line.split(' ')
        try:
            if len(str) > 2:
                x.append(float(str[2]))
                y.append(float(str[1]))
        except ValueError:
            pass
print(i)
plt.figure(figsize=(12, 6))
x = list(map(lambda i : i / 1000, x))
y = list(map(lambda i : i / 1000, y))
print(x)
plt.plot(x, y, 'g^')
plt.title('Cities in Djibouti')
```



Для построения графа используем библиотеку networkx.
Добавим в граф рёбра между всеми вершинами:

```
G = nx.Graph()
for i in range(len(x)):
    for j in range(len(y)):
        G.add_edge(i, j, weight=np.sqrt((x[i]-x[j])**2+(y[i]-y[j])**2))
```

Воспользуемся встроенной функцией библиотеки для построения MST:

```
MST = nx.minimum_spanning_tree(G)
```

Отрисуем MST на реальной карте:

Далее реализуем описанный выше алгоритм построения идеального паросочетания:

```
W = []
s, t = 1, 25
for node in MST.nodes() :
    if (MST.degree[node] % 2 != 0 and not(node==s or node==t)):
        W.append(node)

if (MST.degree[s] % 2 == 0):
        W.append(s)
if (MST.degree[t] % 2 == 0):
        W.append(t)

sub_G = nx.subgraph(G, W).copy()

for u,v,d in sub_G.edges(data=True):
    if d['weight'] != 0:
        d['weight'] = 1/d['weight']

Match = nx.max_weight_matching(sub_G)

Match = [pair for pair in Match]

start = 1
finish = 25

union = Match
for edge in MST.edges():
    union.append(edge)

EulerGraph = nx.MultiGraph()
EulerGraph.add_edges_from(union)
```

Получили Эйлеров граф:

Расположим его вершины в нужном порядке:

```
EP = [edge for edge in nx.eulerian_path(EulerGraph, source=start)]
EP_nodes = []
for edge in EP:
    EP_nodes.append(edge[0])
EP_nodes.append(EP[-1][1])
print(EP_nodes)
```

```
[1, 19, 14, 12, 7, 18, 17, 16, 15, 11, 10, 11, 8, 7, 6, 5, 6, 4, 2, 3,
1, 0, 9, 13, 20, 28, 29, 31, 34, 36, 37, 36, 32, 33, 35, 30, 26, 27, 2
3, 21, 19, 22, 24, 25]
```

Теперь построим искомый гамильтонов путь:

```
AlgoHamPath = []
for node in EP_nodes:
    if not(node in AlgoHamPath) and node != EP_nodes[-1]:
        AlgoHamPath.append(node)

AlgoHamPath.append(EP_nodes[-1])
edges = [(AlgoHamPath[i], AlgoHamPath[i + 1])
         for i in range(len(AlgoHamPath) - 1)]

path = nx.Graph()
path.add_edges_from(edges)
```

Посчитаем длину итогового пути и прибавим расстояние между начальной и конечной точками, чтобы узнать длину гамильтонового цикла:

```
sum_distance=0
for i in range(len(AlgoHamPath)-1):
    sum_distance += G[AlgoHamPath[i] % len(G.nodes())]\
    [AlgoHamPath[(i+1)] % len(G.nodes())]['weight']
print("Длина пути: {}".format(sum_distance))
print("Путь: {}".format(AlgoHamPath))
sum_distance += G[AlgoHamPath[s] % len(G.nodes())]\
[AlgoHamPath[t] % len(G.nodes())]['weight']
print("Длина цикла: {}".format(sum_distance))
```

```
Длина пути: 7.307097641519617
Путь: [1, 19, 14, 12, 7, 18, 17, 16, 15, 11, 10, 8, 6, 5, 4, 2, 3, 0,
9, 13, 20, 28, 29, 31, 34, 36, 37, 32, 33, 35, 30, 26, 27, 23, 21, 22,
24, 25]
Длина цикла: 8.067464666768883
```

Согласно данным с сайта длина оптимального пути - 6656. То есть 6656 * 5 / 3 = 11093 > 8067. Наш алгоритм сработал верно.

6 Заключение

Задачи, связанные с TSP, на данный момент не имеют оптимального решения. Однако, сейчас активно разрабатываются алгоритмы, позволяющие получить приближения. TSP имеет много применений в планировании, логистике и производстве микрочипов. Он появляется как подзадача во многих областях, таких

как секвенирование ДНК. В этих приложениях понятие «город» представляет, например, фрагменты ДНК, а понятие «расстояние» представляет собой время в пути или меру сходства между фрагментами ДНК. TSP также появляется в астрономии, поскольку астрономы, наблюдающие за многими небесными телами, хотят минимизировать время, затрачиваемое на перемещение телескопа между ними. Именно поэтому важно продолжать развитие в этой области и поиск лучших решений. Уже сейчас были достигнуты хорошие результаты и для многих других случаев с миллионами городов можно найти решения, которые гарантированно будут находиться в пределах 2–3% от оптимального тура.

Ссылки

[1] Vazirani, V. Approximation Algorithms, Springer, 2001

[2] https://www.tau.ac.il/ anily/publications/23.pdf

[3] https://en.wikipedia.org/wiki/Christofides_algorithm

[4] https://clck.ru/34aKgo

[5] https://www.lektorium.tv/sites/lektorium.tv/files/additional_files/20091101_algorithmsfornphardproblems_kulikov_lecture06.pdf

[6] https://en.wikipedia.org/wiki/Travelling_salesman_problem

[7] https://e-maxx.ru/algo/mst_kruskal_with_dsu

[8] https://e-maxx.ru/algo/matching_edmonds