Алгоритм Бермана-Карпински приближённого решения задачи (1, 2)-TSP с точностью $\frac{8}{7}$

Введение

Пусть дан полный граф $K_n = (V, E)$ с весами рёбер $c \in \mathbb{R}^E$. *Задача коммивояжёра* (TSP) заключается в том, чтобы найти гамильтонов цикл (обход) в G минимальной стоимости. Если $c_{ij} = c_{ji}\ \forall (i,j) \in E$, говорят, что задача симметрична.

Если веса рёбер удовлетворяют неравенству треугольника, говорят о метрической задаче коммивояжёра. Метрическая задача коммивояжёра — одна из старейших известных NP-полных задач (доказательство NP-полноты см. в [1]). Наилучший известный сейчас результат — приближение с точностью $\frac{3}{2}$[2].

Внимания заслуживает случай симметричной задачи коммивояжёра с весами 1 и 2, называемый (1,2)-TSP. Поскольку $\forall a, b, c \in \{1, 2\}\ a \leqslant b + c$, неравенство треугольника всегда выполняется, и это — подкласс метрической задачи коммивояжёра. (1, 2)-TSP можно рассматривать как обобщение задачи о гамильтоновом пути, требующее наличия в гамильтоновом пути как можно меньшего количества рёбер веса 2.

Было доказано ([3]), что нельзя построить приближение лучше $\frac{741}{740}$. Наилучший известный алгоритм Пападимитриу-Яннакакиса [4] предоставлял приближение 7/6 и был незначительно улучшен до 65/56 [5].

Берман и Карпински в своей статье [6] привели новую методику, позволившую улучшить точность приближения до $\frac{8}{7}$. Этот алгоритм мы и рассмотрим. Поскольку статья очень объёмная, а проект практический, некоторые выкладки будут пропущены, а некоторые утверждения оставлены без доказательства.

NP-полнота

Поставим задачу формально.

TSP $= \{(G, \mathbf{w}, k) : \exists$ цикл во взвешенном графе $(G, \mathbf{w})$, посещающий каждую вершину ровно 1 раз и имеющий вес $\leqslant k\}$
Теорема 1. *TSP — NP-полно.*
Доказательство.

- TSP $\in$ NP[7]:

  Сертификатом является собственно обход. За полиномиальное время проверяется, что такой обход в графе действительно существует, и что его длина $\leqslant k$.

- UHAMPATH $\leqslant_p$ TSP: [8]

  Сопоставим всем рёбрам единичные веса и возьмём $k = n$. Путь длины n, проходящий через все вершины, обязан быть гамильтоновым. □

Эквивалентная формулировка

Представим экземпляр задачи (1,2)-TSP как граф G, вершины которого — точки метрики, а рёбра — пары вершин на расстоянии 1. Пусть в G n вершин, и можно найти его покрытие k путями (простыми и вершинно непересекающимися). Тогда эти пути содержат $n - k$ рёбер и их можно соединить в обход, где ребро из конца одного пути в начало другого будет иметь вес 2. Стоимость обхода будет $n + k$, задача — минимизировать k. Если оптимальное решение содержит $n + k^*$ рёбер, нам нужно найти покрытие путями из $\leqslant \frac{1}{7}n + \frac{8}{7}k^*$ путей. Будем далее решать такую задачу.

Алгоритм малых улучшений

Будем поддерживать текущее решение алгоритма как множество рёбер A ("algorithm"), являющееся 2-паросочетанием (то есть, у вершины может быть не более двух инцидентных ей рёбер из A). Пусть в нём k_A путей и циклов, m_A вершин в циклах, s_A синглтонов (т. е. компонент связности размера 1). Это решение можно преобразовать с помощью множества рёбер C ("change") в новое решение $A \oplus C$.

C улучшает A, если

① $A \oplus C$ — 2-паросочетание;

② $k_{A\oplus C} < k_A$ или

③ $k_{A\oplus C} = k_A$ и $m_{A\oplus C} > m_A$, или

④ $k_{A\oplus C} = k_A$ и $m_{A\oplus C} = m_A$ и $s_{A\oplus C} < s_A$

Алгоритм K-IMPROV:

A = $\emptyset$;
while $\exists C, |C| < K, C$ *улучшает* A **do**
  $A \leftarrow A \oplus C$;
end

Предположим, что для некоторой константы K выполняется

$$(*) \text{ либо } k_A \leqslant \frac{1}{7}n + \frac{8}{7}k^*, \text{ либо } \exists C : C \text{ улучшает } A \textbf{ и } |C| \leqslant K.$$

Заметим, что

- нельзя провести более чем n улучшений вида ②, т. к. число путей и циклов получится $\leqslant 0$;
- нельзя провести более чем n улучшений вида ③ подряд без улучшения вида ②, т. к. в циклах получится $> n$ вершин;
- нельзя провести более чем n улучшений вида ④ подряд, т. к. иначе будет < 0 изолированных вершин.

То есть, можно провести не более чем n^3 улучшений.

Проверка всех пунктов ① — ④ занимает $O(n)$.

Перебор всех допустимых вариантов C в самой простой реализации (перебор всех подмножеств множества рёбер) занимает $O((n^2)^K)$. Авторы статьи утверждают, что время работы поиска улучшения можно сократить до $O(n^K)$, но далее (в разделе про тестирование) описывается, почему это в нашем случае не так важно. Окончательно, время работы K-IMPROV составляет $O(n^{K+4})$.

Доказательство утверждения (*)

В статье [6] утверждение доказывается для $K = 21$, в расширенной версии статьи — для $K = 15$. Зафиксируем оптимальное решение, 2-паросочетание B ("best"), т. ч. $k_B = k^*$. Пусть D — множество рёбер, оба конца которых лежат в одном и том же цикле A. Пусть $\mathcal{G}$, *вспомогательный граф (auxilary graph)*, — граф, содержащий вершины графа G и рёбра $(A \cup B) \setminus D$. Покрасим его рёбра в три цвета. Белые из $A \setminus B \setminus D$, чёрные из $B \setminus A \setminus D$ и серые из $(A \cap B) \setminus D$.

Опр. 1. *Чередующийся путь (alternating path)* (ЧП) — путь, начинающийся и заканчивающийся чёрными рёбрами, в котором чёрные и белые рёбра чередуются.

Опр. 2. *A-объектами* назовём пути и циклы A . Для A-объекта определим *начальную вершину (initial node)* как вершину, которая может быть первой или последней вершиной ЧП и при этом принадлежит A-объекту.

- Для A-пути начальные вершины — его концы.
- Для A-цикла C возьмём за начальные вершины такие две, что в C существует гамильтонов путь из одной в другую, который может быть продолжен двумя чёрными рёбрами до двух других вершин. То, что они всегда существуют в цикле из $\leqslant 7$ вершин, доказано далее.

Опр. 3. *Фишка (или монетка)* — абстрактная делимая единица, которая распределяется по вершинам. Изначальное распределение фишек задаётся весами вершин. Общее количество фишек — сумма этих весов.

Если B состоит из k^* путей, оптимум имеет стоимость $n + k^*$ и множество A даёт достаточное приближение, если оно имеет стоимость $\leqslant \frac{8}{7}(n + k^*)$, т. е. если оно состоит из $\leqslant \frac{1}{7} + \frac{8}{7}k^*$ A-объектов. Возьмём $n + 8k^*$ фишек и докажем, что для того, чтобы A было достаточно хорошим приближением, каждый A-объект должен содержать 8 фишек. Заметим: вершина, инцидентная $2 - a$ вершинам B, содержит $1 + 4$ фишек (сумма их всех $2k^*$).

Изначально фишки пути есть фишки его концов и фишки цикла есть фишки всех его вершин. Оставшиеся фишки содержатся в чередующихся путях. За каждую свою вершину, которая ещё не находится в каком-то A-объекте, ЧП забирает 0.5 фишки. Оставшиеся 0.5 фишек могут быть собраны либо другим ЧП, в который входит эта вершина, либо распределены по каким-то другим правилам (см. далее). ЧП будет отдавать свои фишки тем A-объектам, которые содержат его начальные вершины. Если разбить ЧП на два, каждый из них будет содержать только одну начальную вершину, и поэтому давать свои фишки только одному A-объекту.

Обычно A-путь содержит два конца, и каждый из них — начальная вершина ЧП, который даёт этому пути $2\frac{1}{2}$ фишек. Цикл обычно содержит $4 + a$ вершин и содержит 2 начальных вершины ЧП, которые дают циклу $\frac{3-a}{2}$ фишек.

Могут быть отклонения от обычного случая. Если A-объект содержит менее 2-х начальных вершин, он собирает больше фишек с вершин, которые он содержит. Если A-объект — синглтон, то каждая начальная вершина даёт ему 3 фишки. Если цикл содержит более 6 вершин, в нём не будет начальных вершин. Вершина, инцидентная только одному ребру из B, имеет 4 дополнительных фишки, и мы будем опускать особые случаи, вызванные такими вершинами.

Очень маленькие улучшения

В некоторых случаях возможны улучшения, добавляющие только одно ребро. Обсудим их, а дальше будем предполагать, что они не встречаются.

- Чёрное ребро e, соединяющее две начальные вершины. Если оно соединяет два разных A-объекта, они сливаются в один. Если сливаемый объект — цикл, приходится удалить одно из его рёбер. Если e содержит начальные вершины одного A-объекта, это обязан быть A-путь, и вставка ребра превращает этот путь в цикл.
- Ребро, соединяющее A-синглтон с другим A-объектом (за исключением случая, когда синглтон соединяется со средней вершиной A-пути из трёх вершин — иначе улучшение типа ④).

Предположим, что нет очень маленького улучшения, и есть ЧП R, начинающийся в u, а $\{u\}$ — A-синглтон. Тогда для пути (v, w, x) и некоторого y путь R начинается с (u, w, v, y). Когда мы считаем R возможной частью улучшения, мы берём "сокращённую" версию R, начинающуюся с (v, y). Так как мы предположили, что вставка одного ребра не приведёт к улучшению, это законно. С одной стороны, мы забываем, что R начинается с синглтона и поэтому должен собрать лишнюю $\frac{1}{2}$ фишки. С другой стороны, мы забудем, что R собирает $\frac{1}{2}$ фишки на вершине w.

Начальные вершины циклов

Утв. 1. Пусть C — цикл A с $\leqslant 7$ вершинами с $|C|$ фишками (т. е. у него все вершины смежны с двумя рёбрами из B).

Пусть $\hat{C}$ — множество вершин из C, и $\hat{K} \subset \hat{C}$ — множество вершин, инцидентных двум чёрным вершинам.

Покажем, что какие-то две вершины из $\hat{K}$ согласованы в том смысле, что они — концы гамильтонова пути в $\hat{C}$.

Доказательство.

- Если $|\hat{K}| = 2$, то $\hat{K}$ — согласованная пара, потому что множество рёбер B, содержащихся в C, образует единственный путь.
- Пусть $|\hat{K}| \geqslant 3$.

  Предположим, что две вершины $\hat{K}$, u и v, соседние на цикле C. Тогда они согласованы, т. к. мы можем сделать путь, удалив ребро (u, v) из C.

  Иначе они не смежны, и получаем, что $|\hat{K}| \leqslant |\hat{C}|/2$. Это означает, что $|\hat{K}| = 3$ и $|\hat{C}| = 6$.

  Т. к. $\hat{K}$ содержит 3 вершины, B покрывает $\hat{C}$ двумя путями. Один из этих путей содержит 1 вершину и 0 рёбер, другой 5 вершин и 4 ребра. Следовательно, 2 ребра B содержатся в $\hat{C} \setminus \hat{K}$. БОО, C — цикл $(u_0, \ldots, u_5)$, $\hat{K} = \{u_0, u_2, u_4\}$ и $\{u_1, u_3\}]inB$. Тогда мы можем обойти $\hat{C}$ так: $(u_0, u_5, u_4, u_3, u_1, u_2)$. □

ЧП с дефицитом — общий метод

Пусть R — ЧП. По нашим правилам, он собирает $\frac{1}{2}$ фишек за каждую свою вершину, за исключением концов путей и вершин циклов. Мы не собираем больше потому, что каждая из таких вершин может принадлежать более чем одному ЧП.

Есть несколько методов дать R больше фишек:

- Распределить фишки серых вершин.
- Разбить ЧП P, проходящий через цикл, созданный R. Рассмотрим одну из образовавшихся половинок, P.
  - Если она короткая, можно слить цикл с A-объектом, содержащим начальную точку P.
  - Если она длинная, ей не надо собирать фишки с её рёбер в цикле, и эти фишки можно передать R.

Избегаем плохих случаев

S-дуги — Избегаем их или находим им дополнительные фишки

Опр. 4. *Дуги (arcs)* — чёрные рёбра, содержащиеся в путях.

Дуги потенциально проблематичны, так как могут позволить короткому ЧП создать больше циклов, так как они позволяют получить цикл из одного фрагмента A-пути и одного чёрного ребра (собственно, дуги). В этом случае, в ЧП перед этой дугой и после неё идут белые рёбра (или конец пути).

Опр. 5. Назовём такую конструкцию *S-дугой* (short cycle making arc). Число вершин на фрагменте пути, соединяющем концы дуги, назовём *длиной дуги*.

Избежим создания S-дуг декомпозицией чёрных и белых рёбер в множество ЧП. Когда сделать это будет невозможно, мы наделим такие S-дуги серыми вершинами, что даст им дополнительные фишки.

Чтобы применить эту технику, рассмотрим *цепь* дуг Q — путь, образованный дугами, содержащимися в A-пути P. Для общности продлим P в оба направления одним *фиктивным белым ребром*. Будем принимать решение, как соединить элементы цепи со смежными белыми рёбрами, когда будем строить ЧП. Соединение дуги с фиктивным белым ребром неявно делает её начальной вершиной её ЧП.

Зам. 1. Решения о разбиении внутри цепи могут быть неоднозначными, и зависят от того, как мы захотим сделать.

Как разрешать неоднозначности?

Сначала возьмём дугу $a_0 \in Q$ с наименьшим приоритетом. Далее, удостоверимся, что никакая другая дуга в Q не является S-дугой. Стартуя из любого конца Q, двигаемся к a. На этом конце Q поступаем произвольно. По индукции рассмотрим другой конец дуги $a \neq a_0$. На другом его конце мы приняли произвольное решение. Если это решение соединило a с белым ребром, направленным от него, соединим a с белым ребром, направленным к нему, иначе поступим произвольно.

Далее авторы статьи рассматривают 7 "хороших случаев" для цепей. Это очень громоздко, поэтому здесь это приводиться не будет.

Опр. 6. Цепи, которые не подпадают ни под один из этих случаев, назовём *проблемными*. Они могут образовать *суперцепь* из цепей, соединённых серыми рёбрами. Если такая цепь содержит конец P, назовём её *терминальной*.

Из разбора хороших случаев и определения "плохой" цепи следует, что:

1. Проблемная терминальная цепь не может быть соединена в суперцепь с не-проблемными.
2. Терминальная проблемная цепь состоит только из рёбер, смежных своему концу.
3. Как забрать у терминальной проблемной цепи фишки?
   - Если нетерминальная проблемная цепь имеет только одну дугу, то это дуга длины 5 или более, и она смежна двум серым рёбрам, направленным внутрь. Мы можем взять дополнительную фишку у любого из этих рёбер.
   - Нетерминальная проблемная цепь с более чем одной дугой смежна с двумя серыми рёбрами на своих концах. Решим, что соберём фишку с одного из этих рёбер. Дадим наименьший приоритет дуге длины $\geqslant 6$, которая содержит внутри себя серое ребро.

Избегаем терминальных коротких циклов

Утв. 2. Пусть R — ЧП, начинающийся с конечной вершины A-пути P. Можно избежать ситуации, когда первые 4 ребра R — два черных и два белых — определяют изменение, которое создаёт цикл C.

ЧП с дефицитом — Соединение цикла с циклом

Утв. 3. Если ЧП R соединяет два цикла, он должен собрать $1\frac{1}{2} + 1\frac{1}{2} = 3$ фишек.

ЧП с дефицитом — Соединение большого цикла с концом пути

Утв. 4. Если ЧП R соединяет путь P и цикл C_0 длины 6, не задающий улучшение, он должен собрать $2\frac{1}{2} + \frac{1}{2} = 3$ фишек.

ЧП с дефицитом — Соединение маленького цикла с концом пути

Утв. 5. Если ЧП R соединяет путь P и цикл C_0 длины 4 или 5, он должен собрать $2\frac{1}{2} + 1\frac{1}{2} = 4$ фишек (5 фишек, если длина 5).

ЧП с дефицитом — Соединение конца пути с концом пути

Утв. 6. Если ЧП R соединяет два конца пути, он должен собрать 5 фишек или образовать маленькое улучшение.

Оценка времени работы

Мы должны делать улучшения типа ④, пока существует изолированная вершина, соединённая с чем либо, кроме центральной вершины пути из 3-х вершин. От таких ситуаций можно избавиться за один проход по вершинам.

Число улучшений типа ③ и ② ограничено $O(n^2)$, поэтому время работы ограничивается лишь размером пространства поиска и стоимостью определения того, что малое изменение является улучшением.

Улучшение, которое мы ищем — то, которое не включает ЧП, который отдаёт значительное число фишек соответствующим A-объектам. Наибольшее число фишек (а именно, 5) надо при соединении двух A-путей, поэтому недостаточно длинный путь будет иметь 4 белых ребра и 5 чёрных рёбер. Заметим, что когда мы выбираем чёрное ребро, есть только два варианта выбрать белое, потому что оно должно принадлежать тому же A-пути, что и конец выбранного чёрного ребра.

Такой путь не создаёт улучшение, если он образует по крайней мере 2 цикла. Если цикл создаётся S-дугой, у нас есть дополнительная фишка, поэтому у нас только 3 белых ребра. Есть исключение, Хороший случай 1. Это исключение может произойти только если S-дуга соединяет две вершины, одна из которых соединени с концом своего A-пути. Это соединение должно быть частью улучшения, но есть только два варианта, то есть проверка этого замедлит алгоритм не более чем в константу раз.

Число возможных путей с 4 белыми вершинами есть $O(n^5)$. Если такой путь образует два цикла и мы не можем найти больше фишек, в предыдущем пункте показано, что можно получить улучшение, соединив

1. один из циклов в концом A-пути, где соединение есть ЧП с $\leqslant 2$ белыми вершинами и $\leqslant 3$ чёрными
2. каждый из этих двух циклов с A-циклом при помощи ЧП с $\leqslant 1$ белым ребром.

Строго говоря, ЧП с одним белым ребром, соединяющий два цикла, должен быть расширен рёбрами внутри каждого цикла, чтобы создать улучшение, удовлетворяющее ①. Поскольку эти рёбра могут быть выбраны произвольно, мы можем выразить улучшение множеством, которое не содержит их; поэтому наибольшее улучшающее множество состоит из $9 + 3 + 3 = 15$ рёбер. Если соединить путь с циклом, то соединение будет содержать 5 рёбер, и улучшающее множество будет состоять из $9 + 5 = 14$ рёбер.

Реализация алгоритма

Исходный код

```
def check_if_matching(G, k):
    """Проверка, является ли граф G k-паросочетанием"""
    for n in G.nodes:
        if len(list(G.neighbors(n))) > k:
            return False
    return True

def get_components_count(G):
    """Подсчёт по графу G всех необходимых значений:
    количество путей и циклов, объединением которых он является,
    количество вершин в циклах и
    количество синглтонов"""
    subgraphs = nx.connected_component_subgraphs(G)
    number_vertices_in_cycles = 0
    components_count = 0
    singletons_count = 0
    for g in subgraphs:
        components_count += 1
        if g.number_of_nodes() == 2 and g.number_of_edges() == 1:
            singletons_count += 1
        try:
            if len(nx.find_cycle(g)) == len(g.nodes):
                number_vertices_in_cycles += len(G.nodes)
        except nx.NetworkXNoCycle:
            pass
    return components_count, number_vertices_in_cycles, singletons_count

def get_chain_on_nodes(G, C_nodes):
    """Возвращает граф, который есть цепь из G на данных C_nodes вершинах"""
    return set([(C_nodes[i], C_nodes[i+1])
                for i in range(0, len(C_nodes)-1)
                if (C_nodes[i], C_nodes[i+1]) in G.edges])

def improv(K, G, draw=False, desired=None):
    """Алгоритм K-IMPROV
    draw может работать плохо из-за каких-то багов с colormap'ами в networkx"""
    if K > G.number_of_edges():
        K = G.number_of_edges()

    # Изначально A пусто
    A = nx.Graph()
    A.add_nodes_from(G.nodes)
    A_components_num = sys.maxsize
    A_vertices_in_cycles_num = sys.maxsize
    A_singletons_num = sys.maxsize

    # Перебираем все возможные варианты для C
    log.info('Граф с %d вершинами и %d рёбрами'%(G.number_of_nodes(), G.number_of_edges()))
    max_iter = sum([comb(G.number_of_edges(), i) for i in range(0, K+1)])-G.number_of_edges()-1
    log.info('Число комбинаций для перебора %d при K=%d'%(max_iter, K))

    n_global_iter = 0
    something_changes = True
    last_modification_iter, last_modification_global_iter = 0, 1
    while something_changes:
        n_global_iter += 1
```

```
log.info('             GLOBAL ITERATION %d'%n_global_iter)
something_changes = False
C_edges_s = itertools.chain.from_iterable(itertools.combinations(G.edges, i)
                                          for i in range(1, K+1))

if log.getLogger().isEnabledFor(log.INFO):
    pbar = tqdm.tqdm_notebook(total=max_iter)
exiting = False
for n_iter, C_edges in enumerate(C_edges_s):
    if log.getLogger().isEnabledFor(log.INFO):
        pbar.update(1)

    if last_modification_global_iter < n_global_iter and n_iter > last_modification_iter:
        log.info("Early exit: nothing to check")
        break
    if C_edges == set():
        continue

    C = nx.Graph()
    C.add_nodes_from(G.nodes)
    C.add_edges_from(C_edges)
    G_new = nx.symmetric_difference(A, C)

    cur_components_num, cur_vertices_in_cycles_num, cur_singletons_num = get_components_count(G

    if check_if_matching(G_new, 2):
        if (cur_components_num<A_components_num) \
                or (cur_components_num == A_components_num \
                    and cur_vertices_in_cycles_num > A_vertices_in_cycles_num) \
                or (cur_components_num == A_components_num \
                    and cur_vertices_in_cycles_num == A_vertices_in_cycles_num \
                    and cur_singletons_num < A_singletons_num):
            something_changes = True
            log.info('Changing A: %d %d %d'%( cur_components_num, cur_vertices_in_cycles_num, c
            log.debug('was: %d %d %d'%( A_components_num, A_vertices_in_cycles_num, A_singleton
            last_modification_iter = n_iter
            last_modification_global_iter = n_global_iter

            # Баг networkx: если вставить рисовалку сюда, первый раз будет неправильный цвет.

            A = G_new
            A_components_num = cur_components_num
            A_vertices_in_cycles_num = cur_vertices_in_cycles_num
            A_singletons_num = cur_singletons_num

            # Сделаем выход пораньше, чтобы тестилось побыстрее
            algo_ans = G.number_of_nodes()+A_components_num
            if desired is not None:
                if algo_ans < 8/7*desired:
                    exiting=True
                    log.info('K=%d was enough'%C.number_of_edges())
                    break

            if draw:
                edge_color = [choose_color(edge, A, C) for edge in G.edges ]
                plt.figure(figsize=(5,5))
                draw_weighted_graph(G, edge_color=edge_color, width=4, edge_cmap=rb_cmap)
                plt.show()
if log.getLogger().isEnabledFor(log.INFO):
    pbar.close()
if exiting:
```

```
            break

    algo_ans = G.number_of_nodes()+A_components_num
    if A_components_num == 1 and A_vertices_in_cycles_num > 0:
        algo_ans -= 1

    return algo_ans, A, C.number_of_edges()
```

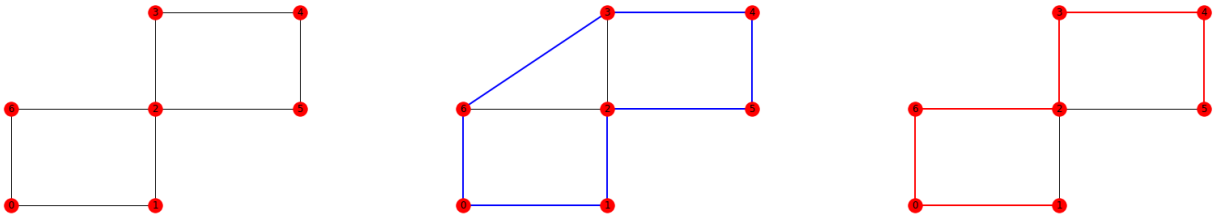
Генерация тестов

Поскольку (см. выше) в (1, 2)-TSP всегда выполняется неравенство треугольника, достаточно сгенерировать граф с рёбрами 1 на каких-то местах, а все остальные рёбра сделать веса 2.

При тестировании результат работы алгоритма сравнивался с результатом работы широко известной программы Concorde, предназначенной для точного решения задачи TSP.

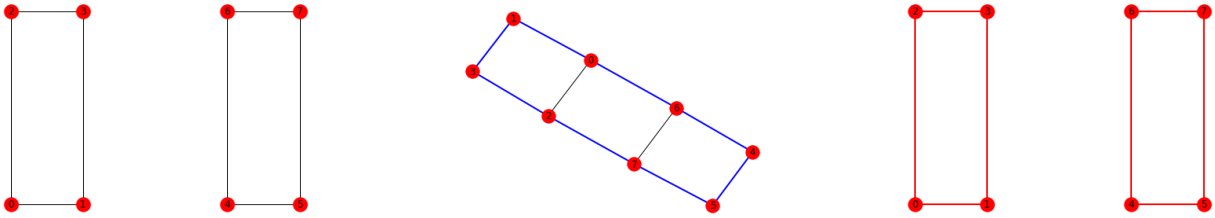
Тестирование на маленьких неслучайных примерах

Были рассмотрены два небольших графа (Рис. 1 и Рис. 2).



(a) n=7, m=8 (b) Concorde=8 (c) K-IMPROV=8, K=2

Рис. 1



(a) n=8, m=8 (b) Concorde=10 (c) K-IMPROV=10, K=1

Рис. 2

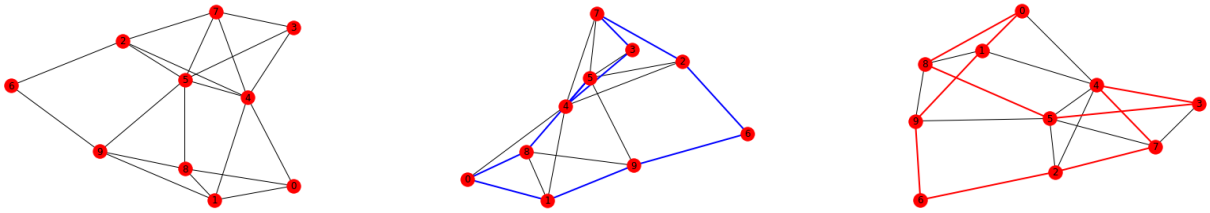
Генерация случайных тестов

Алгоритм тестировался на случайном графе модели Эрдеша-Реньи $G(n, m)$ с разными значениями n и m. Также возможно тестирование на графе модели $G(n, p)$ с разными значениями n и p.

```
def set_edges_weight(G, w):
    for u, v in G.edges():
        G[u][v]['weight'] = w
    return G

def gen_random_graph(n, p=None, m=None):
    if p is None and m is None or not (p is None or m is None):
```

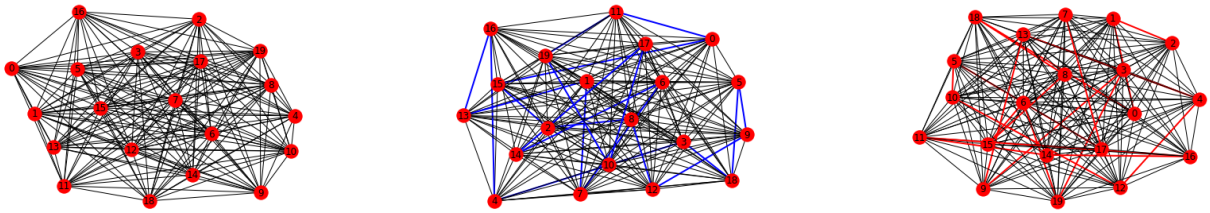
```
        raise Exception("Wrong params")

    if p is not None:
        G = nx.random_graphs.gnp_random_graph(n, p)
    else:
        G = nx.random_graphs.gnm_random_graph(n, m)
    nx.draw(G, with_labels=True)
    plt.show()
    G = set_edges_weight(G, 1)
    return G

G = gen_random_graph(20, m=45)
```

Поскольку в расширенной версии статьи утверждение доказано для $K = 15$, запуски проводились именно при таком значении K.



(a) n=10, m=20 (b) Concorde=10 (c) K-IMPROV=10, K=3

Рис. 3



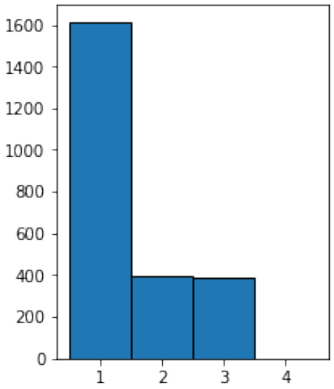
(a) n=20, m=180 (b) Concorde=20 (c) K-IMPROV=22, K=3

Рис. 4

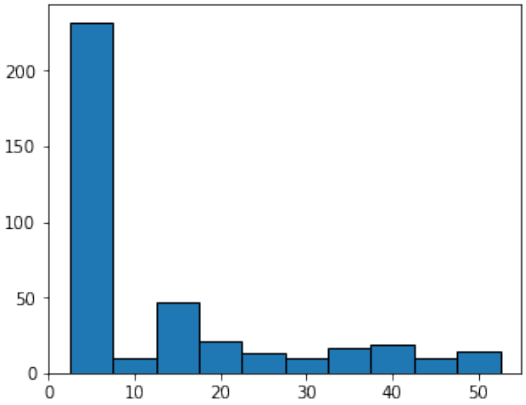
Расчёты показывают, что алгоритм (даже в своей оптимальной реализации) очень неэффективен при $n > 20$ (см. Таблицу 1). Поэтому применялся следующий подход: считалось точное значение при помощи Concorde, запускался алгоритм и останавливался, как только давал достаточное приближение. Это позволило провести очень много тестов. В частности, можно было проверить, какое значение K необходимо, чтобы достичь нужного приближения на случайном графе $G(n, m)$, и для каких параметров n и m это K максимально.

n	Число итераций на 1 шаг	Оценочное время работы, мин
17	131036	1
18	261953	2
19	523108	5
20	1042359	11
21	2069234	22
22	4084225	45
23	7997928	88
24	15505565	172
25	29703650	330
26	56138565	623

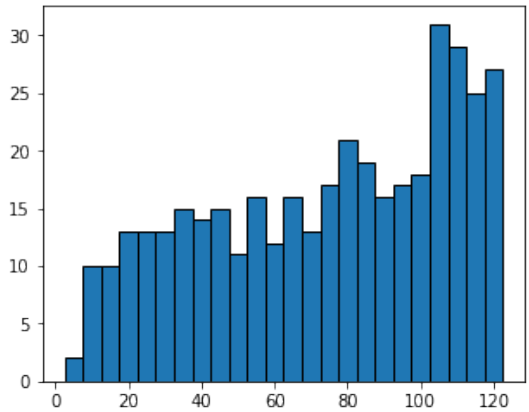
Таблица 1: Ожидаемое время работы



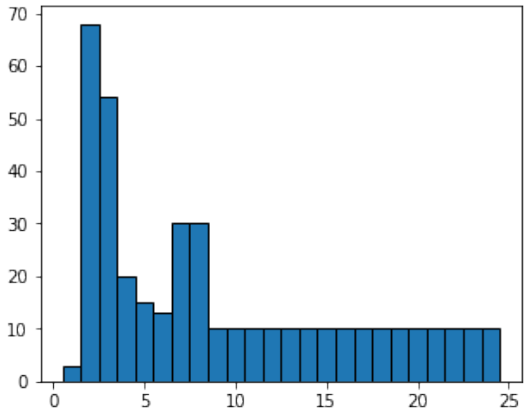
(a) Распределение K



(b) Распределение n при $K \geqslant 3$



(c) Распределение m при $K \geqslant 3$



(d) Распределение $\frac{m}{n}$ при $K \geqslant 3$

Рис. 5

Изучение гистограмм говорит, что на графах, которые рассматривались ($n = 5..50$ и $m = 5..120$ с шагом 5, для каждой пары параметров генерировали и смотрели случайный граф 10 раз), наиболее часто большие K требовалось при $n = \frac{m}{2} = 5$. Если не рассматривать такие маленькие графы, то в целом при $n = \frac{m}{2}$ значения были высокими, равно как они становились высокими при приближении графа к полному. Так или иначе, на рассматриваемых графах для достижения нужного приближения не потребовалось $K > 4$ (хотя если бы мы не останавливали алгоритм, он дал бы более точное приближение, использовав при этом бóльшие значения K).

Выводы

Алгоритм продемонстрировал свою работоспособность, но при этом очень большая константа не позволяет применять его на практике (алгоритм точного решения задачи TSP перебором с отсечениями Concorde был в сотни раз быстрее). Более того, наш алгоритм решает очень специфическую задачу, $(1, 2) - TSP$, которая вряд ли встречается в реальной жизни.

Список литературы

[1] D. S. Johnson and C. H. Papadimitriou. The travelling salesman problem. In E. L. Lawler, J. K. Lenstra, A. H. J. Rinnooy Kan, and D. B. Shmoys, editors, *Computational Complexity*, page 37. John Wiley & Sons, Chichester, 1985.

[2] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[3] Lars Engebretsen and Marek Karpinski. *Approximation Hardness of TSP with Bounded Metrics*, pages 201–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[4] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, February 1993.

[5] M. Bläser and L. Shankar Ram. An improved approximation algorithm for tsp with distances one and two. In *Proceedings of the 15th International Conference on Fundamentals of Computation Theory*, FCT'05, pages 504–515, Berlin, Heidelberg, 2005. Springer-Verlag.

[6] Piotr Berman and Marek Karpinski. 8/7 approximation algorithm for (1,2)-TSP. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 641–648, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[7] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[8] Д. В. Мусатов. *Сложность вычислений.*