ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ....6
1. ОСНОВНЫЕ ПОНЯТИЯ....8
1.1. Алгоритмы и их эффективность....8
1.2. Классы P, NP и труднорешаемые задачи....9
1.3. Основные определения теории графов....12
1.4. Линейное программирование и полиэдральная теория....13
1.5. Матроиды....15
1.6. Выводы по разделу....15
2. ЗАДАЧА КОММИВОЯЖЕРА НА МАКСИМУМ....16
2.1. Определение задачи....16
2.2. Виды задач....17
2.3. Применение задачи коммивояжера на максимум....19
2.4. Теория антиматроидов....21
2.5. Выводы по разделу....25
3. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ MAX TSP....27
3.1. Метод ветвей и границ....27
3.2. Динамическое программирование....29
3.3. Вспомогательные алгоритмы....30
3.4. Алгоритм координатного подъема....41
3.5. Градиентный алгоритм с форой....41
3.6. Алгоритм Сердюкова....42
3.7. Улучшенный алгоритм Сердюкова....43
3.8. Выводы по разделу....46
4. РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ, РЕАЛИЗУЮЩЕЙ АЛГОРИТМЫ РЕШЕННИЯ ЗАДАЧИ КОММИВОЯЖЕРА НА МАКСИМУМ....48
4.1. Программа MAX TSP....48
4.2. Анализ алгоритмов....50
4.3. Выводы по разделу....54
ЗАКЛЮЧЕНИЕ....55
ЛИТЕРАТУРА....57
ПРИЛОЖЕНИЕ 1. Описание программы....60
ПРИЛОЖЕНИЕ 2. Руководство пользователя....64
ПРИЛОЖЕНИЕ 3. Текст программы....70

ВВЕДЕНИЕ

Задачи комбинаторной оптимизации встречаются на практике постоянно. Однако долгое время они не привлекали к себе пристального внимания исследователей, так как в большинстве случаев для их решения находился какой-то естественный алгоритм вроде перебора. Поиск более рациональных алгоритмов в то время не мог представлять интереса для задач малой и большой размерности, поскольку в первом случае такие алгоритмы ненамного лучше естественного, а во втором – они так же, как и естественный алгоритм, не приводят к решению из-за большого объема вычислений. Тем не менее, в последние десятилетия повсеместное развитие вычислительной техники и ее применение существенно изменило положение, так как стало возможным решать задачи большой размерности. Оказалось, что для таких задач различные усовершенствования естественных алгоритмов могут давать существенный выигрыш во времени работы и требуемой памяти. Необходимость практического решения широкого круга комбинаторных задач привела к появлению большого количества усовершенствований естественных алгоритмов, а во многих случаях и к построению принципиально новых алгоритмов. Это в свою очередь потребовало разработки теоретических методов сравнения их качества, а также исследования общих принципов построения хороших комбинаторных алгоритмов [37].

Наиболее известной задачей комбинаторной оптимизации является задача коммивояжера. Практические ситуации, связанные с задачей коммивояжера сопровождали людей тысячелетия, однако математическое значение они впервые приобрели лишь в XIX веке, а подробно изучаться стали только в середине прошлого века. В данной научно-исследовательской работе рассматривается задача коммивояжера на максимум.

Актуальность данной темы заключается в том, что в подавляющем большинстве исследуется задача (причем обычно евклидовая на плоскости) на минимум, поскольку у такой задачи есть очевидное практическое значение. Вследствие этого, в решении задачи на минимум достигнуты уже значительные успехи – разработаны серьезные эвристические методы, специальные программные приложения, решены графы очень большой размерности. Практическое значение задачи на максимум не столь очевидно, однако оно тоже велико – существуют приложения этой задачи к проблемам эффективного сжатия произвольных данных и анализа последовательностей ДНК. В результате по количеству публикаций, исследований и достигнутым результатам задача нахождения максимального обхода находится далеко позади задачи минимизации. Однако в последнее время найдены новые подходы к ее решению. Главные успехи связаны с построением новых приближенных алгоритмов.

Различные практические ситуации естественным образом ведут к задачам комбинаторной оптимизации. Это не удивительно – ведь задачи дискретной оптимизации являются моделями практических ситуаций выбора наилучших вариантов из конечного их множества. Конечно же, в большинстве случаев

математические модели не тождественны самой ситуации, а являются ее приближенным описанием, поэтому и решать задачи комбинаторной оптимизации разумно с той же степенью приближения к оптимуму. Тем более, что принципиальная трудность этих задач делает, по-видимому, невозможным построение эффективных точных алгоритмов для их нетривиальных представителей [8]. Приближенные же алгоритмы имеют своим плюсом довольно высокую эффективность, и вследствие этого, представляют большой интерес то, насколько точными являются эти алгоритмы на практике и возможно ли улучшение существующих методов с точки зрения получения решений, более близких к оптимуму.

Этим вопросам и посвящена данная дипломная работа. В ней для конкретной комбинаторной задачи (задачи коммивояжера на максимум) рассматриваются оценки точности и эффективности самых совершенных на сегодняшний день приближенных алгоритмов. Кроме того, рассмотрена теория антиматроидов. Эта теория была разработана не так давно и пока представляет собой только лишь ряд статей в англоязычных математических журналах. Теория антиматроидов показала неэффективность использования жадных и основанных на жадной эвристике алгоритмов для решения задачи коммивояжера, что в итоге привело к разработке новых алгоритмов и усовершенствованию существующих.

Целью данной работы является разработка программы, реализующей алгоритмы решения задачи коммивояжера на максимум, и проведение сравнительный анализ алгоритмов.

Объектом данной работы является задача коммивояжера на максимум.

Предметом данной работы является разработка улучшенного приближенного алгоритма решения задачи коммивояжера на максимум.

В ходе работы решаются следующие задачи:

- выполнить анализ предметной области;
- изучить известные алгоритмы решения задачи коммивояжера на максимум;
- разработать улучшенный приближенный алгоритм;
- разработать интерфейс пользователя;
- выполнить программную реализацию алгоритмов;
- выполнить сравнительный анализ алгоритмов.

1. ОСНОВНЫЕ ПОНЯТИЯ

1.1. Алгоритмы и их эффективность

Каждая задача характеризуется размером. Размер задачи служит мерой количества входных данных и представляется одним или несколькими целочисленными параметрами. Под алгоритмом понимается общий, шаг за шагом выполнимый способ получения решения данной задачи.

Пусть n – это параметр, определяющий размерность задачи (к примеру, для задачи коммивояжера, в роли n выступает число городов). Тогда через $f(n)$ будем обозначать количество элементарных шагов в алгоритме (временн**а**я сложность алгоритма). Приведем примеры значений временной сложности алгоритмов для некоторых задач.

Для задачи коммивояжера алгоритм полного перебора будет иметь временную сложность $f(n)=(n-1)!$, метод «ветвей и границ» $f(n)=1{,}26^n$. Для задачи «о минимальном стягивающем» дереве алгоритм полного перебора имеет временную сложность $f(n)=(n^{n-2})$, алгоритм Примма - $f(n)=\frac{n^2}{2}$.

Для оценки временной эффективности алгоритмов можно применять различные критерии. Например, время выполнения программы, написанной на основе данного алгоритма. Однако, определенное в таком случае время зависит не только от используемого алгоритма, но также от архитектуры и набора внутренних инструкций компьютера, от качества компилятора, да и от уровня программиста, реализовавшего тестируемый алгоритм. Время выполнения также может существенно зависеть от выбранного множества тестовых данных. Чтобы повысить объективность оценок алгоритмов, принята асимптотическая временная сложность как основная мера эффективности выполнения алгоритма. Определим это понятие.

Функция $T(n)$ имеет порядок (или степень) роста $O(f(n))$, если существуют константы c и n_0 такие, что для всех $n \geq n_0$ выполняется неравенство $T(n) \leq c f(n)$ [49].

Говорят, что алгоритм имеет эффективность (т. е. временную сложность в самом худшем случае) O(f(n)), или просто f(n), если функция от n, равная максимальному числу шагов, выполняемых алгоритмом, имеет порядок роста O(f(n)), причем максимум берется по всем входным данным длины n [49].

Под шагом, производимым алгоритмом, понимается выполнение простой "операции", обычно аппаратно реализованной на любой ЭВМ, а именно, любой арифметической операции, операции сравнения, пересылки и т. п.

В данной работе большое внимание уделяется анализу алгоритмов с полиномиальной и экспоненциальной степенями роста. Полиномиальная степень роста – $O(n^k)$, экспоненциальная - $O(a^n)$, где n – размерность задачи.

Очевидно, что чем меньше степень роста времени выполнения алгоритма, тем больше размер задачи, которую можно решить. Поэтому при решении задач большой размерности предпочтение отдается алгоритмам с полиномиальной

эффективностью, а в свою очередь экспоненциальные и тем более факториальные (степень роста $O(n!)$) алгоритмы являются наиболее бесперспективными с точки зрения увеличения размера задачи.

Легко доказать, что полиномиальный и экспоненциальный характер времени работы алгоритма инвариантен относительно реализации (языка программирования, его уровня, способа кодирования данных, программиста и т. д.). Для этого любой алгоритм может быть рассмотрен как некий "черный" ящик, который по входной последовательности производит выходную последовательность. Если рассматривать кодирование входных и выходных данных последовательностями нулей и единиц (двоичными данными), то алгоритм можно представить как последовательность элементарных двоичных операций, таких, как или, и, не, печать и т. д. Поскольку перевод алгоритма с языка программирования высокого уровня и конкретного способа представления данных на этот уровень двоичных символов может производиться за полиномиальное время, то и общее время работы алгоритма останется либо полиномиальным, либо экспоненциальным [11].

Для приближенных алгоритмов решения какой-то задачи оптимизации наиболее значимой характеристикой является оценка точности приближенного решения, оценка того, как оно соотносится с решением оптимальным. Например, этой характеристикой может быть отношение оптимального и приближенного значений целевой функции в наихудшем случае.

Пусть A – задача оптимизации (минимизации или максимизации) с положительной целочисленной функцией стоимости c , и пусть – алгоритм, который по данной индивидуальной задаче I задачи A выдает допустимое решение $f_A(I)$. Оптимальное решение задачи I обозначим $f^{¿}(I)$. Тогда называется -приближенным (эпсилон-приближенным) алгоритмом для задачи A для некоторого ≥ 0 в том и только том случае, если:

$$¿c(f_A(I))-c(f^{¿}(I))\vee\frac{¿}{c(f^{¿}(I))}\leq\varepsilon \quad , \qquad (1)$$

для всех индивидуальных задач I .

1.2. Классы P, NP и труднорешаемые задачи

При решении очень большого числа естественно возникающих задач оптимизации математики часто сталкивались с тем, что эти задачи были в определенной степени труднорешаемы. Труднорешаемы в том смысле, что эффективных, имеющих полиномиальную сложность алгоритмов для них до сих пор не найдено, и имеются веские доводы, позволяющие предположить, что для этих задач эффективных алгоритмов решения не существует.

Принципиальным результатом исследования таких задач стали понятия классов P и NP .

Класс P (polynomial) – класс задач, которые можно решить детерминированными алгоритмами с полиномиальным временем работы.

Класс NP (nondeterministic polynomial) – класс задач, которые можно решить за полиномиальное время недетерминированными алгоритмами.

Поясним отличие детерминированных и недетерминированных алгоритмов. Предположим, алгоритм доходит до места, в котором должен быть сделан выбор из нескольких альтернатив. Детерминированный исследовал бы одну альтернативу и потом возвращался бы для исследования других, а недетерминированный может исследовать все возможности одновременно, "копируя", в сущности, самого себя для каждой альтернативы. Все копии недетерминированного алгоритма работают независимо и могут, в свою очередь, создавать дальнейшие копии. Если копия обнаруживает, что она сделала неправильный выбор, она прекращает выполняться, а если она находит решение, то объявляет о своем успехе, и все копии прекращают работать [11].

Очевидно, что класс P является подмножеством класса NP, то есть $P \subseteq NP$. Важной проблемой является вопрос совпадения этих классов. Введем понятия NP-трудной и NP-сложной задач:

Задача P_1 преобразуется в задачу P_2, если любой частный случай задачи P_1 можно преобразовать за полиномиальное время в некоторый частный случай задачи P_2, так что решение задачи P_1 можно получить за полиномиальное время из решения этого частного случая задачи P_2 (рис. 1).

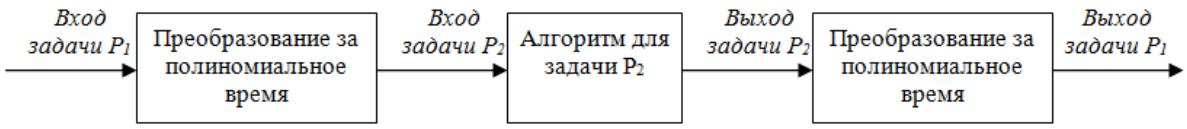

Рис. 1. Схематическая диаграмма, показывающая преобразуемость P_1 в P_2

Задача является NP-трудной, если каждая задача из NP преобразуется в нее, и задача является NP-полной, если она одновременно является NP-трудной и входит в NP [11].

Класс NP-полных задач обладает следующими интересными свойствами, вытекающими из определения:

- никакую NP-полную задачу нельзя решить никаким известным полиномиальным алгоритмом;
- если существует полиномиальный алгоритм для какой-нибудь NP-полной задачи, то существуют полиномиальные алгоритмы для всех NP-полных задач.

Основываясь на этих двух свойствах, многие высказывают гипотезу, что ни для какой NP-полной задачи не может существовать полиномиального алгоритма, однако никому пока не удалось доказать этого, а, следовательно, вопрос о совпадении классов P и NP остается открытым. Исследователи склоняются к мнению, что без развития совершенно новых математических методов доказать эту гипотезу не удастся [9].

Практическое значение понятия NP -полной задачи состоит в том, что такие задачи считаются труднорешаемыми с вычислительной точки зрения, что они не поддаются эффективному алгоритмическому решению и что для любого алгоритма, точно решающего NP -полную задачу, потребуется в худшем случае экспоненциальное количество времени, и, следовательно, он не будет применим на практике ни к каким, за исключением очень малых, задачам. Таким образом, вместо тщетных попыток разработать эффективный алгоритм для какой-то новой комбинаторной задачи, имеет смысл попытаться доказать ее NP -полноту [9].

Исследование NP -полных задач привело к тому, что варианты распознавания фактически всех разумных комбинаторных задач оптимизации лежат в NP . Например, в книге [1] указывается более трехсот NP -полных задач. Ниже приведен список наиболее известных.

1. Задача выполнимости. Для данных m дизъюнктов $C_1,..,C_m$, содержащих переменные $x_1,\ldots,x_n$, требуется определить, выполнима ли формула $C_1C_2\ldots C_m$.

2. Задача целочисленного линейного программирования. Требуется решить следующую задачу оптимизации:

$$\begin{cases} \min c\cdot x, \\ A\cdot x, \\ x\geq 0 \end{cases}, \qquad (2)$$

при условии, что x – целочисленно.

3. Задача независимое множество. Имеется граф, есть ли в нем k попарно несмежных вершин.

4. Задача вершинное покрытие. Имеется ли в графе вершинное покрытие из k вершин.

5. Задача гамильтоновости. Является ли граф гамильтоновым.

6. Задача о максимальной клике. Требуется выяснить, существует ли в графе клика размера k .

Клика – множество попарно смежных вершин графа.

Также к NP -полным задачам относится и задача коммивояжера в варианте распознавания (оптимизационная задача коммивояжера NP -трудна). Она будет определена в следующей главе.

Каков может быть подход к решению NP -полных и NP -трудных задач? До тех пор пока $P\,NP$ размеры решаемых задач из класса NP сильно ограничены. Даже с учетом того, что для точного решения разработан ряд эвристических методов, нет никакой гарантии, что какую-то конкретную задачу (даже небольшой размерности) эти методы смогут решить за приемлемое время. К примеру, в методе ветвей и границ никакая эвристика не может гарантировать, что на каждом шаге не придется рассматривать все ветви и ни одна не будет отсечена. Да, для большей части исходных данных задача может быть точно решена за время близкое к полиномиальному, но для некоторых данных используемые эвристики не смогут помочь избежать экспоненциального времени.

Один из возможных подходов к труднорешаемым задачам заключается в построении алгоритмов полиномиальной сложности для получения "хорошего", но, возможно, не оптимального результата (-приближенных алгоритмов). Правда возникает вопрос: как сильно отличается найденное решение от оптимального? Обычно легче сконструировать быстрый алгоритм, дающий правдоподобное решение, чем оценить его погрешность [1]. Рассмотрению возможных алгоритмов (и точных, и аппроксимационных) решения задачи коммивояжера на максимум посвящена третья глава данной работы.

1.3. Основные определения теории графов

Задача, рассматриваемая в данной работе, относится к задачам оптимизации на графах, поэтому напомним некоторые определения [14] и введем обозначения.

Через (v), где $vV(G)$, будем обозначать множество ребер $E'E(G)$, инцидентных v.

Через (S), где $SV(G)$, будем обозначать множество таких ребер $E'E(G)$, у которых только одна из инцидентных вершин принадлежит S.

Через (S), где $SV(G)$, будем обозначать множество таких ребер $E'E(G)$, у которых обе инцидентные вершины принадлежат S.

Дадим определения, касающиеся гамильтоновых циклов.

Маршрутом в графе называется последовательность ребер вида $v_0v_1, v_1v_2, \ldots, v_{m-1}v_m$. Вершина v_0 называется начальной, v_m – конечной.

Длина маршрута – число входящих в него ребер.

Маршрут называется цепью, если все ребра различны.

Маршрут называется простой цепью, если все вершины его различны, за исключением, может быть, начальной и конечной.

Цикл в графе – замкнутая цепь, содержащая по крайней мере одно ребро.

Граф G – гамильтонов, если в нем существует простая замкнутая цепь, проходящая через все вершины графа; указанную цепь при этом называют гамильтоновым циклом.

Эффективного необходимого и достаточного условия гамильтоновости графа не найдено, более того, задача нахождения указанного цикла в графе является NP-полной.

Во многих алгоритмах используется понятия частичного тура, паросочетания, 2-фактора и контурного покрытия.

Частичный тур – подмножество ребер графа, которое можно дополнить до гамильтонового цикла. Под туром будем понимать сам гамильтонов цикл [14].

Пусть M – подмножество E ребер неориентированного графа $G=(V,E)$. Вершина, покрываемая некоторым ребром из M, называется насыщенной относительно M, в противном случае – ненасыщенной.

Подмножество ME ребер неориентированного графа $G=(V,E)$ называется паросочетанием (совершенным паросочетанием), если каждая вершина vV инцидентна не более (соответственно ровно), чем одному ребру в M [5].

Паросочетание M графа $G=(V,E)$ называется почти совершенным, если каждая вершина $v \in V$, за исключением одной, инцидентна ровно одному ребру в M. Эта вершина называется лишенной [37].

Граф $G=(V,E)$ называется гипонасыщаемым (или граф гипонасыщаем), если количество вершин $¿V \vee ¿$ нечетно, и для любой вершины $v \in V$ существует почти совершенное паросочетание в графе G, которое оставляет вершину v ненасыщенной [37].

Предположим, каждой вершине v графа G приписано некоторое неотрицательное целочисленное значение $f(v)$.

Подмножество $M \subseteq E$ ребер неориентированного графа $G=(V,E)$ называется f-сочетанием, если каждая вершина $v \in V$ инцидентна не более $f(v)$ ребрам в M [16].

Подмножество $M \subseteq E$ ребер неориентированного графа $G=(V,E)$ называется f-фактором, если каждая вершина $v \in V$ инцидентна ровно $f(v)$ ребрам в M [16].

Для этих общих определений уточним понятие 2-фактора и 2-сочетания, поскольку они будут играть важную роль в приближенных алгоритмах решения задачи MAX TSP.

Подмножество $M \subseteq E$ ребер неориентированного графа $G=(V,E)$ называется 2-паросочетанием, если каждая вершина $v \in V$ инцидентна не более, чем двум ребрам в M.

Подмножество $M \subseteq E$ ребер неориентированного графа $G=(V,E)$ называется 2-фактором, если каждая вершина $v \in V$ инцидентна ровно двум ребрам в M.

Пусть M – паросочетание в графе G. Простая цепь называется *чередующейся* (или альтернирующей) относительно M, если среди любых двух соседних ребер данной цепи одно принадлежит паросочетанию M, а другое – нет [15].

Пусть M – паросочетание в графе G. Чередующаяся цепь, соединяющая две ненасыщенные вершины, называется увеличивающей (или аугментальной) относительно M [15].

Максимальным паросочетанием и максимальным 2-фактором во взвешенном графе называются соответственно паросочетание и 2-фактор максимальных весов. Под весом соответствующих подграфов понимается сумма весов, входящих в них ребер.

1.4. Линейное программирование и полиэдральная теория

Пусть $v_1,\ldots,v_m$ – векторы n-мерного евклидова пространства R^n. Вектор $v=❑_1 v_1+\ldots+❑_m v_m$, где $❑_i R(i=1,\ldots,m)$, называется линейной комбинацией векторов $v_1,\ldots,v_m$.

Если $❑_1+\ldots+❑_m=1$, то линейная комбинация называется аффинной.

Если в линейной комбинации каждый элемент $❑_i \geq 0$, то она называется конической комбинацией.

Выпуклая комбинация – это аффинная коническая комбинация.

Линейной (аффинной, конической, выпуклой) оболочкой совокупности векторов $\{v_1,\ldots,v_m\}$ называется множество всех линейных (аффинных, конических, выпуклых) комбинаций векторов $v_1,\ldots,v_m$.

Пусть имеется конечное число точек (векторов) пространства R^n . Их выпуклая оболочка называется политопом [10].

Классический результат Минковского и Вейля [7] утверждает, что каждый политоп является пересечением конечного числа полупространств. На алгебраическом языке это звучит так: всякий политоп можно определить как множество точек, удовлетворяющих некоторой системе линейных неравенств.

Предположим, есть политоп, заданный системой линейных неравенств. Фасет – это множество точек, обращающих некоторое неравенство в равенство. Вершины и грани являются частными случаями более общего понятия фасета.

Множество решений системы линейных неравенств называется полиэдром. Каждый политоп есть полиэдр, но не наоборот. Полиэдр является политопом тогда и только тогда, когда он ограничен. Для описания полиэдра, отличного от политопа, перечня его вершин недостаточно.

Система неравенств, описывающая заданный политоп, называется полиэдральной характеристикой [15].

Основная задача линейного программирования состоит в нахождении наибольшего или наименьшего значения линейной целевой функции на полиэдре, то есть при линейных ограничениях.

Для комбинаторных приложений одна из общих форм записи задачи линейного программирования такова:

$$\begin{cases} c \cdot x \to max, \\ x \geq 0, \\ Ax \leq b, \end{cases} \quad (3)$$

где A – матрица $n \times m$, $b R^n$, 0 – нулевой вектор размерности m .

Двойственная задача к задаче (3) описывается следующим образом:

$$\begin{cases} b \cdot y \to min, \\ y \geq 0, \\ A^T y \geq c, \end{cases} \quad (4)$$

Взаимосвязь между двумя этими задачами отражена в следующей теореме (это утверждение впервые появилось в статье фон Неймана (1947), а позднее, как полноценная теорема с подробным доказательством, – в работе Гейла, Куна и Такера (1950)):

Теорема двойственности линейного программирования. Если одна из задач (3) или (4) имеет решение, и оптимальное значение целевой функции конечно, то другая задача также имеет решение и такое же оптимальное значение целевой функции [34].

Следующее следствие из последней теоремы известно как условия дополняющей нежесткости.

Следствие (условия дополняющей нежесткости). Если x и y – допустимые решения прямой и двойственной задач соответственно, тогда они оптимальны тогда и только тогда, когда выполняются условия:

$$x_i 0 \, y \, A_i = c_i \, \partial \text{ля } i = 1, \ldots, n, \quad (5)$$

$$y_j 0 \, a_j x = b_j \, \partial \text{ля } j = 1, \ldots, m, \quad (6)$$

где A_i обозначает i -й столбец матрицы A , а a_j – j -ю строку [1].

1.5. Матроиды

Матроид – упорядоченная пара $M = ¿E, J>¿$, где E – непустое конечное множество, J – совокупность подмножеств множества E , удовлетворяющая следующим условиям (аксиомам независимости):

(J0) J ;

(J1) если $A J$ и $B A$, что $B J$;

(J2) если $A J$, $B J$, и $¿A \vee ¿ \vee B \vee +1$, то существует такой элемент e , принадлежащий A и не принадлежащий B , что $B\{e\} J$.

Элементы множества J называются независимыми множествами.

Базис – это максимальное по включению независимое множество.

Ранг матроида – количество элементов в любом его базисе.

Множество $A E$, которое в матроиде $M = ¿E, J>¿$ не является независимым, называется зависимым.

Цикл – это минимальное по включению зависимое множество.

Пусть дан матроид $M = ¿E, J>¿$. Мощность множества E называют порядком матроида.

1.6. Выводы по разделу

В данном разделе был выполнен обзор литературы по интересующему нас вопросу. Рассмотрены классы P, NP и труднорешаемые задачи, алгоритмы и их эффективность, приведены основные понятия теории графов, которые будут использоваться в дальнейшем.

Задача коммивояжера относится к NP -полным задачам в варианте распознавания (оптимизационная задача коммивояжера NP -трудна). Даже с учетом того, что для точного решения разработан ряд эвристических методов, нет никакой гарантии, что какую-то конкретную задачу (даже небольшой размерности) эти методы смогут решить за приемлемое время. К примеру, в методе ветвей и границ никакая эвристика не может гарантировать, что на каждом шаге не придется рассматривать все ветви и ни одна не будет отсечена.

2. ЗАДАЧА КОММИВОЯЖЕРА НА МАКСИМУМ

2.1. Определение задачи

Математические проблемы, связанные с задачей коммивояжера впервые были сформулированы в позапрошлом веке двумя математиками: Уильямом Гамильтоном и Томасом Киркменом. Гамильтон придумал игру, в которой игрокам необходимо было составить тур через 20 точек, используя только нарисованные связи между точками (фигура соответствовала додекаэдру).

Задача коммивояжера в общем виде (с минимизацией длины тура) впервые начала изучаться в 20-ых годах прошлого века гарвардским математиком и экономистом Карлом Менгером. В течение следующих 20 лет изучением проблемы занимается все большее и большее количество ученых и, в конечном счете, задача приобретают дурную славу в связи со своей "труднорешаемостью". Прогресс в изучении задачи коммивояжера связан с именами ученых Джорджа Данцига, Рэя Фалкерсона и Селмера Джонсона, которые в 1954 году опубликовали свой метод решения и продемонстрировали его на задаче с 49 городами. Прогресс в решении задачи не стоит на месте, размерности решенных графов растут уже по экспоненте, что связано как с растущей мощностью вычислительных систем (производительность которых согласно одной теории тоже растет по экспоненциальному закону), так и с новыми эвристическими методами. В результате, в настоящее время предпринимаются попытки решить метрическую задачу коммивояжера размерности около двух миллионов, отражающей в себе все населенные пункты земного шара.

Часто возникает путаница в связи с тем, что термином "задача коммивояжера" называют и задачу распознавания, и задачу оптимизации. В данной работе под этим названием будем понимать задачу оптимизации.

Задача коммивояжера на максимум в варианте распознавания определяется следующим образом [9].

Даны целое число n, $(n \times n)$ – матрица (c_{ij}) с неотрицательными элементами и число L. Требуется определить, существует ли такая циклическая перестановка (обход) длины n, что $\sum_{j=1}^{n} c_{j\tau(j)} \geq L$.

Оптимизационную задачу коммивояжера на максимум определим несколькими способами.

1. Классическая постановка [14].

Имеется n городов, которые должен обойти коммивояжер с максимальными затратами. При этом на его маршрут накладывается два ограничения:

1) маршрут должен быть замкнутым, то есть коммивояжер должен вернуться в тот город, из которого он начал движение;

2) в каждом из городов коммивояжер должен побывать точно один раз, то есть надо обязательно обойти все города, при этом не побывав ни в одном городе дважды.

2. В терминах графов [1].

Пусть дан обыкновенный связный взвешенный граф (веса ребер заданы матрицей (c_{ij}) или весовой функцией $c(v_i, v_j)$). Требуется найти гамильтонов цикл наибольшего веса, где вес цикла определяется как сумма весов входящих в него ребер.

3. В терминах перестановок [5].

Пусть дана произвольная матрица (c_{ij}) порядка n с неотрицательными вещественными элементами. Требуется найти такую перестановку $(i_1, i_2, \ldots, i_n)$ множества $\{1, 2, \ldots, n\}$, что сумма $c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_n i_1}$ принимает максимальное значение.

В данной работе рассматривается задача коммивояжера на максимум. Для нее будем использовать обозначение MAX TSP (MAX TSP – Maximum Traveling Salesman Problem (задача коммивояжера на максимум)) и в дальнейшем пользоваться определением ее через графы, так как в этом случае легче всего формально описывать задачу.

Задача TSP является задачей комбинаторной оптимизации (так как множество ее допустимых решений дискретно и конечно) и является NP -трудной. Доказательство NP -трудности задачи коммивояжера можно найти в [9, 11].

Очевидно, что общая задача MAX TSP эквивалентна MIN TSP (задаче коммивояжера на минимум), так как максимальный гамильтонов цикл для матрицы весов c , соответствует минимальному гамильтоновому циклу для матрицы весов $-c$. Что отличает MAX TSP и MIN TSP, так это некоторые частные случаи, в которых на матрицу весов c накладываются ограничения (например, неравенство треугольника для метрической задачи коммивояжера), и что не позволяет свести задачу на минимум к задаче на максимум изменением знака элементов матрицы c [37].

2.2. Виды задач

В большинстве случаев, а именно в описанных ниже задачах 1–4 и 6–7, предполагается, что весовая функция неотрицательна: $c(v_i, v_j) \geq 0$ для $i, j = 1, \ldots, n$ [37].

1. Асимметрическая TSP

Пусть $G = (V, A)$ – полный ориентированный граф с n вершинами с заданной весовой функцией c на его дугах. Если на функцию c не накладывается никаких ограничений, то такая задача MAX TSP называется асимметрической.

Это общая задача коммивояжера, а, следовательно, наиболее сложная, как для точного решения, так и для приближенного. Лучший на сегодня приближенный алгоритм для асимметрической задачи коммивояжера имеет гарантированную точность 2/3. Его можно найти в [41].

2. Симметрическая TSP

Пусть $G = (V, E)$ – полный неориентированный граф с n вершинами с заданной весовой функцией c на его дугах. Задача MAX TSP для такого графа

называется симметрической. Поскольку задача решается для неориентированного графа, то $c(v_i, v_j) = c(v_j, v_i)$ для любых вершин $v_i, v_j V$.

Для матрицы весов выполняется следующее условие: $c_{ij} = c_{ji}$ для всех i, j. Алгоритм Сердюкова решает данную задачу с оценкой точности 3/4 [40]

3. Полуметрическая TSP

Пусть $G = (V, A)$ – полный ориентированный граф с n вершинами с заданной весовой функцией c на его дугах, которая удовлетворяет неравенству треугольника: $c(v_i, v_j) + c(v_j, v_k) \geq c(v_i, v_k)$ для любых трех вершин v_i, v_j, v_k из V. Задача MAX TSP, заданная на таком графе, называется полуметрической.

В 2005 году ряд ученых разработал для этой задачи совершенно новый приближенный алгоритм, использующий декомпозицию регулярных мультиграфов, с оценкой точности 10/13 [41].

4. Метрическая TSP

Пусть $G = (V, E)$ – полный неориентированный граф с n вершинами с заданной весовой функцией c на его дугах, которая удовлетворяет неравенству треугольника: $c(v_i, v_j) + c(v_j, v_k) \geq c(v_i, v_k)$ для любых трех вершин v_i, v_j, v_k из V. Задача MAX TSP, заданная на таком графе, называется метрической. Так как задача задана для неориентированного графа, то $c(v_i, v_j) = c(v_j, v_i)$ для любых вершин $v_i, v_j V$.

На сегодняшний день самый точный приближенный алгоритм имеет точность 7/8 при $n \to \infty$ [39].

5. TSP со складами

Пусть $G = (V, A)$ – полный ориентированный граф с n вершинами с заданной весовой матрицей c. В случае задачи MAX TSP со складами матрица c имеет специальную структуру. Пусть нам даны две $r \times n$ матрицы $u = (u_{ij})$ и $v = (v_{ij})$, тогда матрица c определяется следующим образом: $c_{ij} = \max\limits_{k=1,\ldots,r} (u_{ki} + v_{kj}), для всех i, j = 1, \ldots, n$.

Предполагается, что число r небольшое. Наименьшее r, для которого существует такое представление матрицы c через матрицы u и v, называется комбинаторным рангом матрицы c. Такая задача имеет следующую интерпретацию: предположим, что вместе с n городами есть r складов. Перед посещением каждого города коммивояжер должен забрать товары с любого склада. Стоимость пути из города i в склад k равняется u_{ki}, а стоимость пути со склада k в город j равна v_{kj}. Если коммивояжер хочет максимизировать стоимость обхода, он должен использовать заданную таким образом матрицу c.

В данной задаче элементы матрицы c могут быть отрицательными [37].

6. Евклидова TSP

Пусть в евклидовом пространстве R^k задана метрика. И пусть в R^k заданы n точек $x^1, \ldots, x^n$ с указанным попарным расстоянием $d_{ij} = d(x^i, x^j)$ для любых $i, j = 1, \ldots, n$. Таким образом, евклидова TSP определяется следующим образом.

Пусть $G=(V,E)$ – полный неориентированный граф с n вершинами с заданной на его дугах весовой функцией c, равной $c(v_i,v_j)=d(x^i,x^j)$. Задача MAX TSP, заданная на таком графе, называется евклидовой.

Исходя из определения метрики, евклидова задача коммивояжера является метрической. Вопрос о сложностном статусе этого случая MAX TSP до недавнего времени был открыт (и остается открытым при $k=2$). NP-трудность евклидовой MAX TSP в R^k при $k\geq 3$ установлена С. Фекете [30].

7. Полиэдральная TSP

Пусть в евклидовом пространстве R^k со стандартным скалярным произведением $\cdot,\cdot$ заданы векторы $a_1,\ldots,a_n$. Для определения функции расстояния между точками в пространстве введем единичный шар (содержащий начало координат внутри) $B=\{x\,R^k \vee a_i, x\leq 1, i=1,\ldots,r\}$.

Тогда для упорядоченной пары произвольных точек $(x,y)R^k\times R^k$ определим расстояние $c(x,y)=f(y-x)$, где $f:R^k \overset{+¿}{R}$ есть функционал Минковского от множества B, то есть $f(x)=inf\{R\vee x\,B,>0\}$.

Рассмотрим n точек $x^1,\ldots,x^n$ из R^k таких, что расстояние между x^i и x^j равно $c_{ij}ij=f(x^i-x^j)$, $1\leq i,j\leq n$. Тогда задачу коммивояжера для заданного множества точек назовем полиэдральной MAX TSP. Через m обозначим число фасет шара B.

До недавнего времени сложностной статус полиэдральной MAX TSP оставался неопределенным. В последнее время группой исследователей (С. Фекете, А. Барвинок, Д. Джонсон, Дж. Вудингер, Р. Вудруф) установлены следующие результаты для полиэдральной MAX TSP с центрально-симметрическим (относительно начала координат) и ограниченным шаром B.

1. Задача NP-трудна в R^k, $k\geq 3$, если число фасет единичного шара B является частью ее входа [30].

2. В работе Барвинока, Джонсона, Вудингера и Вудруфа предложен точный алгоритм с временной сложностью $O(n^{m-2}\log n)$.

3. В случае норм L_1 ("манхэттенская норма") и L_∞ на плоскости R^2 С. Фекете построил точный алгоритм с временной сложностью $O(n)$.

2.3. Применение задачи коммивояжера на максимум

Главной практической задачей, для решения которой применяется задача коммивояжера на максимум, является проблема нахождения кратчайшей суперстроки. Пусть даны строки $s_1,\ldots,s_n$, записанные на каком-то алфавите. Без потери общности можно считать, что никакая строка s_i не является подстрокой s_j. Проблема нахождения кратчайшей суперстроки заключается в поиске строки s, для которой каждая s_i является подстрокой. Эта проблема лежит в основе задач анализа последовательностей ДНК и сжатия данных.

Задача нахождения кратчайшей суперстроки сводится к задаче коммивояжера на максимум следующим образом. Пусть c_{ij} – длина перекрытия строк s_i и s_j, то есть длина наибольшей строки v, такой, что $s_i=uv$ и $s_j=vw$ для

каких-то строк u и w. Тогда решение задачи коммивояжера на максимум на полном графе с n вершинами и с весовой функцией c даст нам решение задачи нахождения кратчайшей суперстроки. Таким образом, приближенное решение задачи коммивояжера на максимум даст приближенное решение задачи кратчайшей суперстроки.

Применение задачи нахождения кратчайшей суперстроки при распознавании ДНК и сжатии данных.

1. Анализ последовательностей ДНК.

ДНК представляется строкой символов, заданной на алфавите (A, G, C, T).

ДНК вируса имеет длину $5 \times 10 4$ символов (комплементарных пар оснований нуклеиновых кислот). ДНК человека – 3×10^9 символов.

Задача анализа последовательности ДНК заключается в следующем. Пусть у нас есть различные фрагменты ДНК. Требуется по ним восстановить исходную последовательность (таб. 1).

Таб. 1. Анализ последовательностей ДНК

Фрагменты ДНК	ACTGAACTTACGGG AATCCTAC ACGGGCTAAAGCCATA TGAACTTACGGGCTAAAG TAAAGCCATACGAATCCTAC ACTGAACTTACGGGCTA ATACGAATCC TACGAGA
Найденная последовательность	ACTGAACTTACGGGCTAAAGCCATACGAATCCT ACGAGA

2. Сжатие данных.

Пусть дана последовательность двоичных данных.

0010101011101001010111001010100010011100101010010

Требуется разделить эту последовательность на подпоследовательности, найти для них суперстроку и этим самым сжать последовательность (таб. 2).

Таб. 2. Сжатие данных

Подпоследовательности	00101 0101110100 1010111 001010100 010011 100101010010
Найденная суперстрока	0010101110100101010010011

Строка при этом уменьшилась практически в два раза (с 49 битов до 25).

2.4. Теория антиматроидов

Весьма общей является следующая задача дискретной оптимизации.

Пусть каждому элементу e непустого конечного множества E поставлено в соответствие неотрицательное число $\omega(e)$, называемое весом этого элемента. Вес подмножества $X E$ определяется как сумма весов его элементов:

$$\omega(x)=\sum_{e\in X}\omega(e) \quad . \tag{7}$$

Рассматривается некоторая совокупность J подмножеств множества E. Требуется найти максимальное по включению множество $B\in J$ минимального веса.

Подобный вид имеют или сводятся к нему многие задачи: например, задача коммивояжера, задача о рюкзаке, задача о минимальном стягивающем дереве и другие.

Жадный алгоритм решения описанной задачи состоит в последовательном, элемент за элементом, формировании искомого множества, причем на каждом шаге из всех элементов множества E, добавление которых к ранее выбранным возможно (то есть приводит к некоторому множеству из J), выбирается элемент наименьшего веса.

Как известно [14], жадный алгоритм приводит к оптимальному решению в случае, когда E,J – матроид.

В общем случае это не так, однако, широко распространено мнение, что жадная стратегия приводит к приемлемым с точки зрения практики результатам: получающиеся решения "значительно лучше" наихудших и могут служить в качестве начальных точек при поиске точных решений другими алгоритмами.

Это мнение подкреплено многочисленными экспериментальными результатами по "хорошему поведению" жадного алгоритма при решении евклидовой задачи коммивояжера.

Однако уже в случае антисимметричной задачи коммивояжера были найдены примеры графов, для которых жадный алгоритм приводит к наихудшему из возможных решений. Подробно этот феномен был исследован английскими математиками Гатином Г. и Йео А. в 2001–2005 гг [34, 36].

Пусть E – непустое конечное множество, а J – совокупность некоторых его подмножеств, замкнутая по включению. Антиматроид – это упорядоченная пара E,J, для которой существует весовая функция $\omega: E\to R$ такая, что жадный алгоритм решения задачи на минимум находит подмножество максимального веса.

Обычно пара E,J порождается некоторой задачей дискретной оптимизации [10]. Соответствующую задачу также будем называть антиматроидом.

Рассмотрим задачу о назначениях, которая состоит в выделении совершенного паросочетания наименьшего веса в двудольном графе. В данном случае E – множество ребер двудольного графа, а J – множество всех его паросочетаний. Жадный алгоритм на каждом шаге выбирает ребро наименьшего веса, не смежное

с ранее выбранными ребрами. Весовая функция задается матрицей весов ребер графа. Пусть для графа $K_{3,3}$ матрица весов такова:

$$\begin{pmatrix} 4 & 5 & 5 \\ 5 & 8 & 9 \\ 5 & 9 & 12 \end{pmatrix} .$$

Очевидно, что жадный алгоритм последовательно выберет ребра веса 4, 8 и 12, в результате чего формируется паросочетание веса 24. Легко проверить, что любое другое совершенное паросочетание имеет меньший вес. Таким образом, жадная стратегия привела к наихудшему из возможных решений.

Гатин Г. и Йео А. [34] нашли весьма общее условие, достаточное для того, чтобы пара E, J была антиматроидом.

I – независимое семейство – это упорядоченная пара E, J, где E – непустое конечное множество;

J – совокупность подмножеств множества E (элементы J будем называть независимыми множествами), удовлетворяющая следующим условиям:

(J0) $\varnothing \in J$;

(J1) если $A \in J$ и $B A$, то $B \in J$;

(J2) все базисы (максимальные по включению независимые множества) имеют одинаковую мощность.

Пусть S – независимое множество. Обозначим через $I(S)$ множество всех таких элементов из $¿ E$, добавление любого из которых к S дает вновь независимое множество. Итак,

$$\begin{matrix} ¿ \\ x \in E \\ ¿ \\ I(S) = ¿ \end{matrix} . \qquad (8)$$

I – независимое семейство E, J называется I-антиматроидом, если

(J3) существует такой базис $B' = \{x_1, \ldots, x_k\}$, что для любого другого базиса B выполнено условие

$$\sum_{j=0}^{k-1} \left| I\left(\{x_1, x_2, \ldots, x_j\}\right) \cap B \right| < \frac{k(k+1)}{2} . \qquad (9)$$

Заметим, что в условии важен порядок элементов множества B'. Будем считать, что элемент x_i появляется в множестве B' на i-м шаге работы жадного алгоритма.

Мощность базиса – ранг I-антиматроида. Если ранг не меньше 2, I-антиматроид – нетривиальный.

Гатин и Йео установили, что нетривиальный I-антиматроид является антиматроидом.

Теорема. Для любого нетривиального I-антиматроида E, J существует такая весовая функция $\omega: E \to R$, что жадный алгоритм формирует наихудшее решение (т.е. решение наибольшего веса) задачи нахождения базиса наименьшего веса.

В случае матроида ранга k для любых двух базисов B и B' выполняется неравенство, противоположное неравенству (9). Действительно, из аксиом

матроида вытекает, что в базисе B найдется $k-j$ элементов, которыми можно дополнить множество $\{x_1,\ldots,x_k\}\in B'$ до базиса матроида. Каждый из этих $k-j$ элементов входит в множество I_j. Поэтому $|I_j\cap B|\geq k-j$ и

$$\sum_{j=0}^{k-1}|I_j\cap B|\geq k+(k-1)+\ldots+1=\frac{k(k+1)}{2}\quad . \qquad (10)$$

Неравенство (9) из условия (J3) является точным в следующем смысле: нельзя уменьшить его правую часть, оставив в силе утверждение о том, что I-антиматроид является антиматроидом.

Теорема. Задача о назначениях является I-антиматроидом.

Рассмотрим теперь задачу коммивояжера на минимум. В задаче коммивояжера в роли множества E выступает множество дуг (или ребер) графа, а независимые множества образуют дуги (соответственно ребра), которые можно дополнить до гамильтонова цикла. Выполнение условий (J0), (J1) и (J2) очевидно. Покажем, что имеет место и (J3).

Существует стандартная процедура сведения задачи коммивояжера к задаче о назначениях с дополнительными ограничениями. Она состоит в следующем. По взвешенному ориентированному графу G без петель и кратных дуг с множеством вершин $V=\{1,2,\ldots,k\}$ построим (неориентированный) двудольный граф G' с долями $V_1=\{x_1,x_2,\ldots,x_k\}$ и $V_2=\{y_1,y_2,\ldots,y_k\}$. Каждая дуга ij исходного графа превращается в ребро x_iy_j двудольного графа с сохранением веса. Гамильтонов цикл переходит при этом в некоторое совершенное паросочетание. С другой стороны, всякое совершенное паросочетание в G' задает некоторую подстановку на множестве V. Если указанная подстановка представляет собой один цикл (в алгебраическом смысле), то имеем гамильтонов цикл в G. В противном случае совершенному сочетанию в G' отвечает несколько непересекающихся циклов, покрывающие все вершины графа G.

При переходе от асимметрической задачи коммивояжера к задаче о назначениях (без дополнительных требований к структуре совершенного паросочетания) мощность множества $I_j\cap B$ не уменьшается. Поэтому из выполнения неравенства (3) для задачи о назначениях вытекает его справедливость и в интересующем нас случае. Этим доказано следующее утверждение.

Теорема. Асимметрическая задача коммивояжера – I-антиматроид.

Рассмотрим теперь симметрическую задачу коммивояжера. В своих первых публикациях [34] Гатин и Йео допустили ошибку, отнеся данную задачу к I-антиматроидам. В статье [37] ошибка была исправлена. Оказалось, что симметрическая задача коммивояжера не I-антиматроид, но в некотором смысле весьма близка к нему, что позволило, слегка модифицировав доказательство теоремы, показать, что данная задача является антиматроидом. Справедливы следующие теоремы.

Теорема. Симметричная задача коммивояжера (для графа с не менее чем 4 вершинами) не является I-антиматроидом.

Теорема. Симметричная задача коммивояжера является антиматроидом.

Доказательство. Пусть базис $B'=\{e_1,\ldots,e_k\}$ состоит из ребер гамильтонова цикла $1\to 2\to 3\to\ldots\to k\to 1$, взятых в естественном порядке.

Определим функцию w следующим образом. Пусть число M больше количества вершин k. Положим $w(e_1)=2M$ и $w(e_j)=jM$ при $j>1$. Если ребро e не входит в базис B', то для него найдется индекс j такой, что $e\in I_{j-1}\{I_j$, в этом случае определим $w(e)=1+jM$. Заметим, что множество I_0 совпадает с множеством E всех ребер графа, а $I_1=E\{e_1\}$, поскольку в полном графе любые два различных ребра входят в некоторый гамильтонов цикл. Поэтому $I_0\{I_1=\{e_1\}$, откуда вытекает, что вес любого ребра, отличного от e_1 и e_2, больше $2M$.

При такой весовой функции жадный алгоритм сформирует базис B'. На первых двух шагах будут выбраны ребра наименьшего веса e_1 и e_2. На j-м шаге выбор ведется среди элементов множества I_{j-1}. При этом если $e\in I_{j-1}$, но $e\neq e_j$, то $w(e)>jM=w(e_j)$. Вес гамильтонова цикла B' будет считаться по формуле:

$$w(B')=\frac{Mk(k+1)}{2}+M \quad . \tag{11}$$

Лемма. Для любого гамильтонового цикла $B\neq B'$ справедливо:

$$I_j\cap B\leq\begin{cases}\dfrac{k(k+1)}{2}, & \text{если } e_1\notin B\\ \dfrac{k(k+1)}{2}-1, & \text{если } e_1\in B\end{cases}. \tag{12}$$

Доказательство. Оценим мощности множества $I_j\cap B$. Если $e_1\notin T$, то $|I_j\cap B|=k$. Во всех остальных случаях $|I_j\cap B|\leq k-j$. Действительно, для $j=0,1$ и $k-1$ это очевидно. Поэтому будем считать, что $2\leq j<k-1$. Заметим, что ребро графа входит в I_j тогда и только тогда, когда оно отлично от ребра, соединяющего концевые вершины маршрута $1\to 2\to\ldots\to j\to j+1$, образованного ребрами $e_1,\ldots,e_j$, и не инцидентно ни одной из вершин множества $J=\{2,3,\ldots,j\}$ внутренних вершин этого маршрута. Обозначим через Q_j множество ребер из B, каждое из которых покрывает хотя бы одну вершину из множества J. Пусть m_j – количество ребер в B, у которых оба конца входят в J. Тогда $|Q_j|=2|J|-m_j$, поскольку каждая вершина графа покрывается ровно двумя ребрами гамильтонова цикла B, и при таком подсчете m_j ребер подсчитывается дважды. При этом $m_j<J$ (иначе подграф, образованный указанными m_j ребрами, не будет лесом), откуда $|Q_j|>|J|=j-1$, то есть $|Q_j|\geq j$. Поэтому $I_j\cap B\leq k-|Q_j|\leq n-j$.

Пусть a – наименьшее число, такое, что $e_a\notin B'$ (поскольку $B\neq B'$, такое число существует, при этом $1\leq a\leq k-1$). Тогда начальная часть маршрута B выглядит следующим образом: $1\to 2\to\ldots\to a-1\to a\to b\to\ldots$, где $a+1<b\leq k$.

Если $a=1$, то $¿ I_{b-1} \cap B \vee ¿ k-(b-1)$, так как по доказанному выше в B имеется не менее $b-1$ ребер, инцидентных вершинам из Q_{b-1} и не входящих в I_{b-1} . Если $a>1$, то $e_a \notin B$ и $m_{a+1}=(a+1)-3.$ Поэтому $\left|I_{a+1} \cap B\right|<k-(a+1)$. Таким образом, среди неравенств $\left|I_j \cap B\right| \le n-j$ найдется хотя бы одно строгое. При сумми ровании указанных неравенств получем доказуемое неравенство.

Вернемся к доказательству теоремы. Возьмем произвольный гамильтонов цикл $B=\{f_1,\ldots,f_k\}$, отличный от B' . Далее рассматривается два случая, входит ли ребро $e_1=12$ ввв базис B или нет.

Если не входит, для каждого i найдется такой единственный индекс a_i , что $f_i \in I_{a_i-1}\{I ¿_{a_i}$. При этом $w(f_i) \le a_i M+1$. Далее получаем,

$$¿ I_j \cap B \vee ¿$$

$$w(B)=\sum_{i=1}^{k} w(f_i) \le \sum_{i=1}^{k} (a_i M+1) \le k+M \sum_{i=1}^{k} a_i = k+M \sum_{j=0}^{k-1} ¿ \quad .$$

Используя лемму, получаем,

$$w(B) \le k+M \cdot \frac{k(k+1)}{2} < M+M \cdot \frac{k(k+1)}{2} = w(B') \quad . \qquad (13)$$

Если входит, тогда $a_1=2$ и $\sum_{j=0}^{k-1} \left|I_j \cap B\right| = \sum_{i=0}^{k-1} a_i - 1$. Таким образом, получим

$$\sum_{i=0}^{k-1} a_i = 1+\sum_{j=0}^{k-1} ¿ I_j \cap B \vee \le \frac{k(k+1)}{2} \quad . \qquad (14)$$

Следовательно, $w(B) \le w(B')$. Доказано, что вес любого базиса B , отличного от B' , меньше веса B' .

Рассмотрим пример графа с пятью вершинами (рис. 2). Жадный алгоритм, выбирая на каждом шаге ребро максимального веса, построит гамильтонов цикл наименьшего веса (v_1v_2 , v_2v_3 , v_3v_4 , v_4v_5 , v_5v_1). Таким образом. жадный алгоритм нашел наихудшее решение, а значит, задача коммивояжера на максимум является антиматроидом.

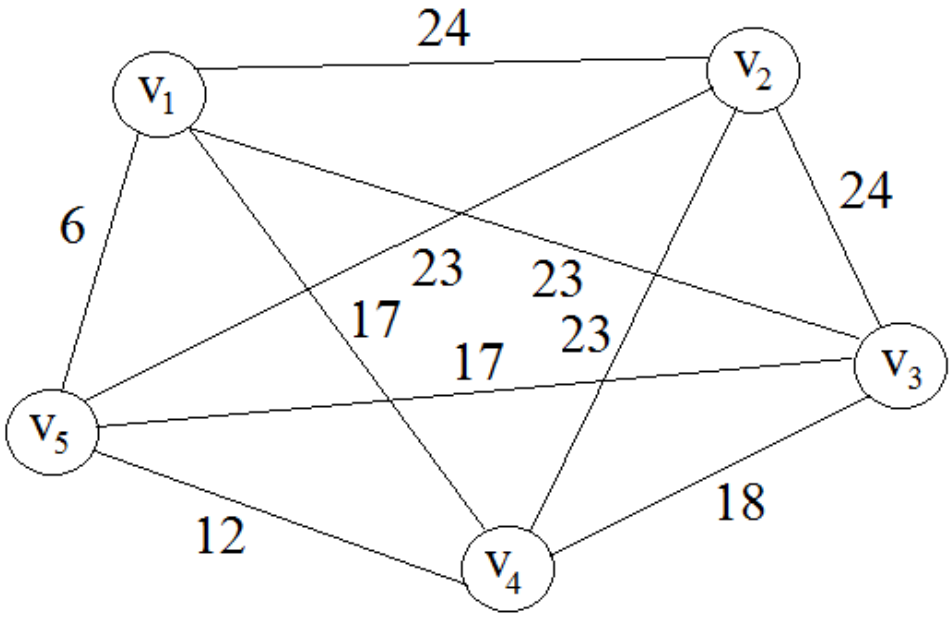

Рис. 2. Пример графа

2.5. Выводы по разделу

В данном разделе описывается задача коммивояжера на максимум, рассматриваются различные виды данной задачи и приводится наилучшая оценка точности, известная на данный момент для конкретного вида задачи.

Главной практической задачей, для решения которой применяется задача коммивояжера на максимум, является проблемы нахождения кратчайшей суперстроки, также существуют приложения этой задачи к проблемам эффективного сжатия произвольных данных и анализа последовательностей ДНК. Также в данном разделе рассмотрена теория антиматроидов – структур, тесно связанных со многими задачами дискретной оптимизации. Данная теория показала неэффективность использования жадных и основанных на жадной эвристике алгоритмов для решения задачи коммивояжера, что в итоге привело к разработке новых алгоритмов и усовершенствованию существующих. В данном разделе приведено доказательство того, что задача коммивояжера на максимум является антиматроидом.

3. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ MAX TSP

3.1. Метод ветвей и границ

Для решения многих NP-полных и NP-трудных задач относительно успешным является применение метода ветвей и границ. Описания его можно найти в [1, 9, 11]. Продемонстрируем применение этого метода к задаче коммивояжера. Будем считать, что граф $G=(V,A)$ является ориентированным и взвешенным.

Для полного графа с n вершинами имеется $(n-1)!$ вариантов маршрута коммивояжера. Естественно, что полный перебор всех вариантов является невозможным даже для не очень больших значений n. Метод ветвей и границ является хорошим способом сокращения полного перебора для получения оптимального решения.

Представим процесс построения маршрута коммивояжера в виде построения двоичного корневого дерева решений, в котором каждой вершине v соответствует некоторое подмножество $M(v)$ множества всех маршрутов коммивояжера. Считаем, что корню дерева решений поставлено в соответствие множество всех маршрутов коммивояжера.

Пусть v – некоторая вершина этого дерева. Выберем дугу (v,w), которая входит хотя бы в один маршрут из $M(v)$. Тогда множество $M(v)$ разбивается на два непересекающихся подмножества, в одно из которых можно отнести все маршруты, содержащие дугу (v,w), а в другое – не содержащие ее. Будем считать, что первое из этих подмножеств соответствует левому сыну вершины x, а второе – правому. Тем самым задано правило ветвления. Вершина дерева решений, для которой строятся сыновья, будет называться активной. Главное достоинство метода ветвей и границ в сравнении с полным перебором заключается в том, что активными объявляются лишь те вершины, в которых может содержаться оптимальный маршрут. Следовательно, необходимо выработать правило активизации вершин, которое будет сводиться к правилу подсчета границ.

Предположим, что для вершин дерева решений вычислено значение $f(v)$ такое, что вес любого маршрута из множества $M(v)$ не больше, чем $f(v)$. Такое число $f(v)$ называется верхней границей маршрутов множества $M(v)$ или границей вершины v (для задачи коммивояжера на минимум определяется нижняя граница). Правило активизации вершин заключается в том, что из множества вершин, не имеющих сыновей, в качестве активной выбирается вершина с наибольшей верхней границей. Вершина, для которой построены оба сына, активной стать не может.

Процесс построения дерева решений продолжается до тех пор, пока активной не будет объявлена вершина v, для которой множество $M(v)$ состоит из одного единственного маршрута, а границы всех других вершин не больше (не меньше для задачи на минимум), чем вес этого маршрута. Понятно, что тогда маршрут, содержащийся в $M(v)$, является оптимальным.

Сформулируем правило вычисления границ.

Процедура вычитания из каждого элемента строки (соответственно столбца) максимального (минимального для задачи на минимум) элемента этой же строки (столбца) называется редукцией строки (редукцией столбца).

Процедура, которая сначала осуществляет редукцию каждой строки, а затем в измененной матрице – редукцию всех столбцов, называется редукцией матрицы, а полученная матрица – редуцированной.

Сумма всех констант, используемых при редукции матрицы, будет границей всех маршрутов коммивояжеров для исходной нередуцированной матрицы весов. Тем самым получено правило вычисления нижней границы корневой вершины дерева решений.

Это правило остается справедливым и для любой другой вершины дерева решений. Действительно, с каждой вершиной v дерева решений однозначно связывается матрица, описывающая все возможные маршруты из $M(v)$. Сумма всех констант, используемых при ее редукции, и граница отца u вершины v дают нужную границу вершины v. А именно:

$$f(v)=f(u)+f, \qquad (15)$$

где $f(v)$ – верхняя граница вершины v, $f(u)$ – верхняя граница отца вершины v, f – сумма всех констант, использованных при редукции матрицы, соответствующей вершине v. В случае задачи на максимум оценка будет не увеличиваться (так как элементы редуцируемых матриц будут неположительны), а в случае задачи на минимум – не уменьшаться.

Предположим, что на некотором этапе рассматривается возможность включения дуги (e_i,e_j), тогда правило получения матрицы $A(v)$ из матрицы $A(u)$ таково:

1) для левого сына (то есть для случая включения дуги в обход) получаем матрицу $A(v)$ удалением из матрицы $A(u)$ строки, соответствующей e_i и столбца, соответствующего вершине e_j;

2) для правого сына (то есть для случая не включения дуги в обход) получаем матрицу $A(v)$ из матрицы $A(u)$ заменой элемента, стоящего на пересечении строки, соответствующей e_i и столбца, соответствующего вершине e_j, на $-\infty$.

Правило выбора дуги (e_i,e_j) можно сформулировать следующим образом: выбрать тот нуль в редуцированной матрице, для которого сумма вторых по минимальности (по модулю) значений в строке и столбце этого нуля наибольшая (так как на эти значения матрица будет редуцироваться для правого сына, и разница границ между правым и левым сыновьями будет наибольшая).

Имеющиеся экспериментальные данные позволяют утверждать, что метод ветвей и границ для TSP является разумной альтернативой полному перебору, так как для случайной матрицы весов число исследуемых вершин дерева решений равно $O((1{,}26)^n)$.

Можно отметить особенности задачи коммивояжера, позволившие реализовать метод ветвей и границ.

1. Ветвление. Множество решений, представляемое вершинами дерева решений, можно разбить на попарно непересекающиеся множества. Каждое подмножество в этом разбиении представляется сыном исходной вершины в дереве разбиений.

2. Границы. Имеется алгоритм для вычисления границы (нижней или верхней) веса любого решения в данном подмножестве.

Поскольку многие NP -трудные и NP -полные задачи обладают двумя перечисленными свойствами, то метод ветвей и границ может применяться для их решения. При этом разбиение множества решений на каждом шаге может осуществляться и более чем на два непересекающихся подмножества, если для подмножеств имеется подходящий алгоритм для вычисления границы входящих в них решений.

3.2. Динамическое программирование

Метод динамического программирования (описан в [8, 9]) близок к методу ветвей и границ в том смысле, что он производит разумный перебор всех допустимых решений задачи, но делает это другим способом. Идея его состоит в том, чтобы идти от последних решений к более ранним.

Пусть для решения некоторой комбинаторной задачи оптимизации нам нужно принять последовательность из n решений, скажем $D_1, D_2, \ldots, D_n$. Тогда, если эта последовательность оптимальна, последние k решений D_{n-k+1} , D_{n-k} ,…, D_n . должны быть оптимальны. То есть завершающая часть оптимальной последовательности решений должна быть оптимальна. Это утверждение часто называют принципом оптимальности.

Рассмотрим применение метода динамического программирования для решения задачи коммивояжера.

Пусть даны множество $S\{2,3,\ldots,n\}$ и kS . Обозначим через $C(S,k)$ оптимальную стоимость пути, начинающегося из города 1, проходящего по всем городам множества S и заканчивающегося в городе k . Задача состоит в вычислении $C(\{2,3,\ldots,n\},1)$. Начнем с нахождения $C(S,k)$ для ¿$S \vee$¿1 , что дает просто

$$C(\{k\},k)=c_{1k}\,1k \text{ для всех } k=2,\ldots,n \quad . \tag{16}$$

Для вычисления $C(S,k)$ при ¿$S \vee$¿1 используем тот факт, что для нахождения наилучшего маршрута из города 1 в город k , проходящего через все города множества S , достаточно рассмотреть для каждого m вариант, в котором m посещается непосредственно перед k , и обратиться к $C(S\{k\},m)$ в предыдущей таблице. Таким образом,

$$C(S,k)=\min_{m \in S\}} \begin{matrix} ¿ \\ ¿ \\ (C(S\}),m \\ ¿+c_{mk}¿¿ \end{matrix} \quad . \tag{17}$$

Этот минимум нужно вычислить для всех множеств S данной мощности и для каждого возможного города m из S . При этом необходимо хранить город

m , для которого достигается минимум, с тем, чтобы можно было восстановить оптимальный обход обратным просмотром. Если считать, что для каждого значения $C(S,k)$ требуется одна ячейка памяти, то в целом требуется память

$$\sum_{k=1}^{n-1} kC_k^{n-1}=(n-1)2^{n-2}=O(n2^n) \quad , \qquad (18)$$

ячеек, а число сложений и сравнений равняется

$$\sum_{k=1}^{n-1} k(k-1)C_k^{n-1}+(n-1)=(n-1)(n-2)2^{n-3}+(n-1)=O(n^2 2^n) \qquad (19)$$

Эти функции, как и временная сложность метода ветвей и границ, экспоненциальны относительно размера задачи n и могут показаться недопустимо большими. Однако если вспомнить тот факт, что при простом переборе имеется $(n-1)!$ различных обходов, можно понять, что на самом деле эти подходы дают огромный выигрыш. Поскольку для TSP не известно алгоритмов, лучших, чем экспоненциальные, то подход с помощью динамического программирования нельзя отбрасывать, хотя доказано, что алгоритмы ветвей и границ в данном случае более эффективны.

3.3. Вспомогательные алгоритмы

Практически все приближенные алгоритмы решения MAX TSP используют вспомогательные задачи. Чаще всего это нахождение максимального паросочетания, максимального 2-фактора и максимального контурного покрытия графа. Для отыскания каждого указанного подграфа (паросочетания, 2-фактора, 2-паросочетания, либо контурного покрытия) максимального веса существуют эффективные алгоритмы с полиномиальной временной сложностью.

Для построения максимального паросочетания можно воспользоваться алгоритмом Габова, приведенном в [17], с временной оценкой $O(n^3)$.

Алгоритм для построения максимального 2-фактора приведен в [19]. Он состоит в построении нового графа, нахождении в нем максимального паросочетания и перехода к первоначальному графу. Поэтому трудоемкость этого алгоритма определяется трудоемкостью и размером подзадачи, состоящей в нахождении максимального паросочетания в построенном графе.

Алгоритм нахождения наибольшего паросочетания.

Сначала рассмотрим задачу нахождения паросочетания наибольшей мощности, поскольку именно она лежит в основе использующегося алгоритма нахождения максимального взвешенного паросочетания.

Известные алгоритмы для нахождения наибольшего паросочетания в произвольном графе, выполняемые за полиномиальное время, занимают место среди самых сложных комбинаторных алгоритмов. Многие из них базируются на процедуре пополнения, осуществляемой с использованием чередующихся цепей. Первый полиномиально временной алгоритм для недвудольных графов построил Эдмондс [41]. В его алгоритме в качестве ключевой была использована идея "сжатия" определенных нечетных циклов. Вплоть до настоящего времени

большинство алгоритмов построения паросочетаний (и в том числе самые удачные их них) базируются – явно или неявно – на этой идее.

Алгоритмы нахождения наибольшего паросочетания основываются на следующей теореме.

Теорема. Паросочетание M является максимальным тогда и только тогда, когда не существует увеличивающей относительно M цепи [41].

Приведем неформальное описание [29] алгоритма Эдмондса для нахождения наибольшего паросочетания. Допустим, что дан граф G, в котором построено паросочетание M. Если M – совершенное паросочетание, то задача решена. Поэтому считаем, что множество S вершин, не покрытых паросочетанием M, не пусто. Построим такой лес F, чтобы каждая его связная компонента содержала в точности одну вершину из S, каждая вершина из S принадлежала только одной компоненте леса F и каждое ребро из F, находящееся на нечетном расстоянии от вершины, входящей в S, принадлежало паросочетанию M. Очевидно, что каждая вершина леса F, находящаяся на нечетном расстоянии от S, имеет в F степень 2. Такие вершины назовем внутренними, а остальные вершины леса F – внешними (в частности все вершины множества S внешние). Такой лес называется M-чередующимся лесом. Очевидно, что лес с множеством вершин S, не имеющий ребер, является M-чередующимся лесом.

Обратимся теперь к соседям внешних вершин. Если найдется внешняя вершина x, смежная с вершиной y, не лежащей в F, то выберем ребро (y, z) $\in M$ и положим $F' = F + (x, y) + (y, z)$. Этот подграф, очевидно, является M-чередующимся лесом, большим, чем лес F.

Если лес F имеет две внешние вершины x и y, принадлежащие различным его компонентам связности и являющиеся смежными в G, то корни этих компонент связаны M-чередующейся цепью, состоящей из ребра (x, y) и двух цепей, идущих от вершин x и y к этим корням. Поскольку эта цепь чередующаяся, то можно получить паросочетание, большее, чем M. Затем используем большее паросочетание.

Допустим, что лес F имеет в одной связной компоненте две внешние вершины x и y, смежные в графе G. Тогда обозначим через C цикл, образованный ребром (x, y) и (x, y)-цепью в F. Пусть P обозначает (единственную) цепь в F, соединяющую цикл C с корнем леса F (цикл C может проходить и через корень, в этом случае цепь P состоит из одной вершины). Очевидно, цепь P является M-чередующейся цепью. Поэтому, если осуществим переключение (т.е. альтернанс) на P, то получим другое паросочетание M_1 того же размера, что и M. Теперь M_1 и C удовлетворяют условиям следующей леммы.

Лемма. Пусть даны граф G, паросочетание M в G и цикл Z длины 2k + 1, содержащий k ребер паросочетания M и вершинно непересекающихся с остальными ребрами из M. Построим новый граф G' из графа G, сжимая цикл Z до единственной вершины. Тогда паросочетание M' = M – E(Z) является наибольшим тогда и только тогда, когда M – наибольшее паросочетание в G [11].

Так как M_1 и C удовлетворяют условиям леммы о сжатии цикла то, если сожмем цикл C до одной вершины, чтобы получить новый граф G', то этим сведем задачу нахождения паросочетания, большего чем M (в графе G), к задаче нахождения паросочетания, большего чем $M_1 – E(C)$, в меньшем графе G'. Пример сжимающегося цикла приведен на рис. 3. Ребра, входящие в паросочетание, изображены толстыми линиями.

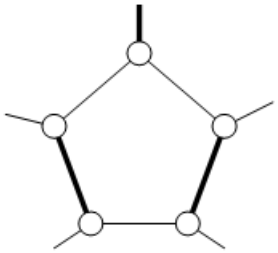

Рис. 3. Цикл, подлежащий сжатию

Наконец, если каждая внешняя вершина имеет соседями только внутренние вершины, то можно утверждать, что паросочетание M уже само является наибольшим. Чтобы убедиться в этом, предположим, что лес F содержит m внутренних и n внешних вершин. Очевидно, $n – m = |S|$. Но паросочетание M пропускает в точности $|S|$ вершин, и поэтому оно должно быть наибольшим паросочетанием.

Резюмируя вышесказанное, заключаем, что можно сделать всегда один из следующих шагов: либо увеличить лес F, либо увеличить паросочетание M, либо уменьшить $|V(G)|$, либо завершить процедуру, имея наибольшее паросочетание. Таким образом, очевидно, что алгоритм выполняется за полиномиальное время и дает наибольшее паросочетание в графе G.

Приведем формальное описание алгоритма.

Для того, чтобы на каждом шаге алгоритма работать с деревьями, а не с более сложными структурами, граф трансформируется. Граф, с которым работает алгоритм на текущем шаге, называется рабочим графом. Вершины рабочего графа называются цветками. Таким образом, цветок является либо вершиной исходного графа G (в случае если данная вершина не "сжималась"), либо его подграфом (в случае если цветок был получен "сжатием").

На входе: полный неориентированный граф $G = (V, E)$, произвольное паросочетание M.

На выходе: наибольшее паросочетание.

Шаг 0. Обозначим $G' = G$.

Шаг 1. Сделаем каждый ненасыщенный паросочетанием M цветок корнем леса F.

Шаг 2. Выберем в графе G ребро (x, y), где x – внешний цветок. Если такой выбор произвести не удалось, то текущее паросочетание является максимальным, и алгоритм прекращает работу. Выберем в графе G' цветок c, содержащий цветок y. Если c – внешний цветок, то перейдем к шагу 3, иначе – к шагу 4.

Шаг 3. Если x и y принадлежат разным деревьям, то увеличим паросочетание M (увеличивающая цепь проходит через (x, y) и соединяет вершины деревьев, содержащих x и y) и перейдем к шагу 1. В противном случае сформируем новый рабочий граф G' сжатием цикла C, образованного ребром (x, y) и (x, y)-цепью в F. Перейдем к шагу 2.

Шаг 4. Найдем d – цветок, инцидентный c в паросочетании M. Если d – внешний цветок, то перейдем к шагу 5, иначе – к шагу 6.

Шаг 5. Заменим цветок c на составляющие его вершины. Перейдем к шагу 2.

Шаг 6. Поместим ребра (x, c) и (c, d) в дерево. Перейдем к шагу 2.

Вернем паросочетание M.

Конец алгоритма.

Алгоритм нахождения максимального взвешенного паросочетания.

В данной работе важную роль играет алгоритм нахождения наибольшего взвешенного паросочетания. Первый такой алгоритм был построен Эдмондсом в 1965 году и, как и все последующие алгоритмы, представлял собой развитие алгоритма нахождения наибольшего паросочетания. Используемый в данной работе алгоритм Габова [8] формально является идентичным алгоритму Эдмондса, но за счет эффективной реализации улучшает временную оценку с $O(n^4)$ до $O(n^3)$.

Первым шагом Эдмондса к решению задачи о максимальном взвешенном паросочетании была идея сформулировать эквивалентную задачу линейного программирования, то есть найти систему совместных неравенств, которые бы являлись полиэдральной характеристикой выпуклой оболочки векторов инцидентности паросочетаний графа. Используя эту идею, Эдмондс разработал полиномиальный алгоритм для решения задачи о максимальном взвешенном паросочетании. Для любой целевой функции, данный алгоритм находит допустимое решение прямой задачи и доказывает его оптимальность с помощью построения двойственного решения, с тем же самым значением целевой функции.

Допустим, что дан взвешенный граф $G = (V, E)$, в котором построено паросочетание M.

Вектором инцидентности паросочетания M графа G называется вектор $x^M \in \mathbb{R}^{|E|}$, такой что:

$$x_j^M = \begin{cases} 1, \text{если } e \in M, \\ 0, \text{если } e \notin M. \end{cases} \tag{20}$$

Если понятно, о каком паросочетании идет речь, символ M будем опускать.

Отправной точкой нахождения эквивалентной задачи линейного программирования является формулировка задачи целочисленного программирования:

$$\begin{cases} c \cdot x \to max, \\ x(\delta(v)) \le 1, \forall v \in V, \\ x \ge 0, \\ x - \text{целое}. \end{cases} \tag{21}$$

Если граф двудольный, то требованием целочисленности можно пренебречь. Если же убрать это требование аргумента x для общего графа, то вершины полученного политопа $\tilde{P}$ могут оказаться не целочисленными. Именно поэтому задача нахождения максимального взвешенного паросочетания и является "нетривиальной" задачей комбинаторной оптимизации. К примеру, рассмотрим простейший недвудольный граф – треугольник и возьмем вектор инцидентности равный x = (0,5; 0,5; 0,5). Эта точка является вершиной политопа $\tilde{P}$ и является дробной. Чтобы избавиться от требования целочисленности, но при этом перейти к эквивалентной задаче, Эдмондс вывел следующую систему:

$$\begin{cases} x(\delta(v)) \le 1, \forall v \in V, \\ x(\delta(S)) \le \left\lfloor \frac{|S|}{2} \right\rfloor, \forall S \subseteq V, |S| \text{ нечетное}, \\ x \ge 0. \end{cases} \quad (22)$$

Данной системе могут удовлетворять и дробные точки, однако вершины политопа, задаваемого этой системой, являются целыми. Заметим, что данный политоп является ограниченным, и целевая функция для задачи нахождения максимального взвешенного паросочетания являться линейной ($c\ x \to \max$). Из этого можем сделать вывод, что следующая задача:

$$\begin{cases} c \cdot x \to max, \\ x(\delta(v)) \le 1, \forall v \in V, \\ (\delta(S)) \le \left\lfloor \frac{|S|}{2} \right\rfloor, \forall S \subseteq V, |S| \text{ нечетное}, \\ x \ge 0, \end{cases} \quad (23)$$

является задачей линейного программирования, и оптимальное значение ее целевой функции ограничено, а это значит, что оно достигается в вершине политопа, задаваемого ограничениями. В силу этого решения задач (ссылка на последнюю систему) и (ссылка на систему выше) будут совпадать.

Сформулируем прямую и двойственную задачу в терминах гипонасыщаемых графов.

Пусть $\aleph$ - семейство гипонасыщаемых подграфов графа G.

Для любого ребра $e \in E$ пусть $C(e) = \{ H \in \aleph : e \in E(H) \}$.

Пусть y – это *вершинный весовой вектор*, определенный на каждом гипонасыщаемом подграфе H графа G и имеющий неотрицательные значения. Пусть y_x обозначает вес $y(\{x\})$ вершины x (любая вершина также является гипонасыщаемым подграфом).

Тогда прямая и двойственная задача будут выглядеть следующим образом:
прямая задача

$$\begin{cases} c \cdot x \to max, \\ x(\delta(v)) \le 1, \forall v \in V, \\ (\delta(E(H))) \le \left\lfloor \frac{|V(H)|}{2} \right\rfloor, \forall H \subseteq, \\ x \ge 0, \end{cases} \quad (24)$$

двойственная задача

$$\begin{cases} \sum y_v + \sum y_H \cdot \left\lfloor \frac{|V(H)|}{2} \right\rfloor \to \min \quad , \forall v \in V, \forall H \in \aleph \\ y_u + y_v + y(\aleph(e)) \geq c(e) \qquad , \forall e = (u,v) \in E \\ y \geq 0 \qquad . \end{cases} \tag{25}$$

Пусть $Z_y = \{ e = (u, v) \in E : y_u + y_v + y(\aleph(e)) = c_e \}$.

Минимальный подграф графа G, содержащий ребра Z_y, назовем подграфом равенства.

На каждом шаге алгоритма у нас есть допустимое решение x, являющееся вектором инцидентности паросочетания, и допустимое двойственное решение y. Согласно условиям дополняющей нежесткости, выполняется следующее:

$$x_e > 0 \Rightarrow y_u + y_v + y(\aleph(e)) = c(e) \quad , \text{для } \forall e = (u, v) \in E, \tag{26}$$

$$y_H > 0 \Rightarrow x(E(H)) = \left\lfloor \frac{|V(H)|}{2} \right\rfloor \quad , \text{для } \forall H \in \aleph, \tag{27}$$

$$y_v > 0 \Rightarrow x(\delta(v)) = 1 \quad , \text{для } \forall v \in V. \tag{28}$$

Алгоритм построен таким образом, что на каждом шаге алгоритма условия (26) и (27) выполняются. Алгоритм изменяет x и y до тех пор, пока не выполнится условие (28).

Приведем формальное описание алгоритма [141].

В данном алгоритме такие понятия как цветок, внешние и внутренние вершины, "сжатие" подграфа, рабочий граф имеют тот же смысл, что и в алгоритме нахождения наибольшего паросочетания.

На входе: полный неориентированный взвешенный граф $G = (V, E)$.

На выходе: максимальное взвешенное паросочетание.

Шаг 0. Инициализация. Обозначим $G' = G$, $x_e = 0$ для всех $e \in E$, $y_v = 0.5 \max\{ 0, \max\{ c_e : e \in \delta(v) \} \}$ для всех $v \in V$ и $y_H = 0$ для всех $H \in \aleph$.

Шаг 1. Проверка оптимальности и выбор вершины. Если $y_v = 0$ для каждой вершины v, такой что $x(\delta(v)) = 0$, тогда алгоритм заканчивает работу. В противном случае, для каждой вершины v, такой что $x(\delta(v)) = 0$ и $y_v > 0$ сделаем цветок, содержащий v, корнем леса F. Получим лес F, состоящий только из изолированных внешних вершин.

Шаг 2. Выбор ребра. Пусть Z'_y – это множество ребер G', которые принадлежат Z_y, но не принадлежат F. Просмотрим Z'_y в поисках ребра (u, v), где u – внешнее ребро F, а v не является внутренним ребром F. Если такое ребро не существует, то перейдем к шагу 6. Если такое ребро $j = (u, v)$ существует, то рассмотрим следующие 4 случая.

Случай 1: v не является узлом F и $x(\delta(v)) = 1$. В этом случае перейдем к шагу 3.

Случай 2: v не является узлом F и $x(\delta(v)) = 0$. В этом случае перейдем к шагу 4.

Случай 3: v внешняя вершина, и u и v находятся в разных деревьях леса F. В

этом случае перейдем к шагу 4.

Случай 4: v внешняя вершина, и u и v находятся в одном и том же дереве леса F. В этом случае перейдем к шагу 5.

Шаг 3. Увеличение леса. Пусть $k = (v, w)$ – это ребро паросочетания, инцидентное v. Увеличим лес F, добавив к нему ребра j и k, и сделаем v внутренней, а w внешней вершинами. Перейдем к шагу 2.

Шаг 4. Увеличение паросочетания. Установим $x_j = 1$. Если $x(\delta(v)) = 0$, то увеличим паросочетание с помощью цепи, идущей от u к корню дерева, содержащего u, и ребра j. В противном случае, увеличим паросочетание с помощью цепи, проходящей через j и соединяющей вершины деревьев, содержащих u и v. После этого, рассмотрим каждый сжатый цветок s в увеличивающем пути. Он инцидентен ровно одному ребру $h = (s, t)$, такому, что $x_j = 1$. Пусть этому ребру h графа G' соответствует ребро $h = (s', t')$ графа G. Найдем почти совершенное паросочетание в цветке s, такое, что оно оставляет вершину s' ненасыщенной. Изменим соответственно этому паросочетанию вектор инцидентности x. Очистим лес F и перейдем к шагу 2.

Шаг 5. Сжатие. Ребро j замыкает нечетный цикл в дереве, которое содержит u и v. Произведем сжатие цветков этого цикла в один внешний цветок. Получим новый рабочий граф G'. Перейдем к шагу 2.

Шаг 6. Изменение двойственных переменных. Уменьшим значение двойственных переменных y_i на величину σ следующим образом:

$y_i \leftarrow y_i - \sigma$, если цветок, содержащий i, является внешним,

$y_i \leftarrow y_i + \sigma$, если цветок, содержащий i, является внутренним,

$y_H \leftarrow y_H - 2\sigma$, если H соответствует внутреннему цветку F,

$y_H \leftarrow y_H + 2\sigma$, если H соответствует внешнему цветку F.

Значение σ ограничено условиями осуществимости двойственной задачи линейного программирования:

$\sigma < y_i$ для всех узлов i таких, что цветок, содержащий i, является внешним;

$\sigma < y_H$ для всех $H \in \aleph$, которые при сжатии дали внутренний цветок F;

$\sigma < y_u + y_v + y(\aleph(e)) - c_e$ для всех $e = (u, v) \in E$, которые соединяют внешний цветок F с цветком, не принадлежащим F.

Выберем наибольшее σ, удовлетворяющее условиям. Если $\sigma > 0$, то делаем преобразование y. Если σ было ограничено третьим неравенством, то во множестве Z'_y появилось новое ребро. В этом случае перейдем к шагу 2. Если же значение σ было ограничено первым неравенством, тогда цветку i, который является либо реальной вершиной G, либо результатом сжатия нечетного цикла, соответствует $y_i = 0$. Перейдем к шагу 7. В противном случае, если значение σ было ограниченовторым неравенством, то существует внутренний сжатый цветок i со значением двойственной переменной $y_i = 0$. В этом случае перейдем к шагу 8.

Шаг 7. Псевдоувеличение. На этом шаге выбрана вершина i такая, что $y_i = 0$ и цветок v, содержащий вершину i, является внешним в F. Пусть r – это корень дерева, содержащего v. Если $v \neq r$, то альтернируем паросочетание относительно цепи, идущей по ребрам F от v к r. Если v – сжатый цветок (этот случай включает в себя случай $v = r$), то найдем новое почти совершенное паросочетание в v, которое оставляет вершину i ненасыщенной. Также изменим паросочетание (и вместе с ним и вектор x) внутри сжатых цветков, лежащих на указанной выше цепи. Очистим лес F и перейдем к шагу 1.

Шаг 8. Расширение сжатого цветка. Цветок i – это внутренний сжатый цветок. Пусть (i', j) – это ребро F, не принадлежащее паросочетанию, такое, что вершина i' входит в состав сжатого цветка i. Пусть (i'', j) – это ребро F, принадлежащее паросочетанию, такое, что вершина i'' входит в состав сжатого цветка i. Пусть P – это путь из вершины i' в вершину i'', проходящий через вершины цветка i, через который проходит увеличивающая цепь цветка i. Заменим цветок i на вершины, входящие в его состав, и получим новый рабочий граф G'. Сделаем цветки пути P внешними и внутренними таким образом, чтобы цветки i' и i'' стали внутренними. Перейдем к шагу 1.

Вернем паросочетание M, соответствующее вектору инцидентности x.

Конец алгоритма.

Каждый раз, когда на шаге 1 ищется новый лес, в корни деревьев попадают такие вершины, для которых $x(\delta(v)) = 0$ и $y_i > 0$. Лес растет до тех пор, пока либо не увеличивается паросочетание, и для какой-то новой вершины i $x(\delta(v)) = 1$, либо не выполняется изменение двойственных переменных, и для какой-то новой вершины i $y_i = 0$. Алгоритм никогда не увеличит y_v для какой-то вершины v, если значение $x(\delta(v))$ не равно 1 и никогда не уменьшит $x(\delta(v))$ для какой-то вершины v, если значение y_v не равно 0. Следовательно, после, по крайней мере, $|V|$ выполнений роста леса условие дополняющей нежесткости (19) будет выполнено. В связи с этим, эффективность реализации данного алгоритма определяется эффективностью использующихся алгоритмов роста дерева. Для нахождения максимального взвешенного паросочетания в графе в данной работе используются описанные Габовым в работе [38] структуры данных и алгоритмы, позволяющие в итоге получить временную оценку $O(|V|^3)$.

Алгоритм нахождения 2-фактора сведением к задаче о паросочетании.

Естественным обобщением задачи о нахождении паросочетания является так называемая задача об f-факторе [26]. Предположим, что каждой вершине $v \in V(G)$ присваивается целочисленное значение $f(v)$. Существует ли некий f-фактор, то есть остовный подграф H из G, такой, что $|\delta_H(v)| = f(v)$ для каждой вершины $v \in V(G)$?

Известна задача, тесно связанная с задачей об f-факторе и эквивалентная ей в случае, когда $f(v) = 1$ для всех $v \in V(G)$, однако в общем случае эти две задачи следует различать [26]. Она называется задачей об f-паросочетании. Как и прежде задается целочисленная функция f, определенная на $V(G)$. Спрашивается, можно

ли сопоставить неотрицательное целое значение $x(e)$ каждому ребру e так, чтобы равенство

$$x(\delta(v))=\sum_{\{e|v\in e\}} x(e)=f(v) \quad , \tag{29}$$

выполнялось для каждой вершины v? Так осуществленное присвоение значений $x(e)$ называется совершенным f-паросочетанием. Очевидно, что f-факторы могут идентифицироваться с теми совершенными f-паросочетаниями, которые удовлетворяют дополнительному ограничению $x(e)\le 1$. С другой стороны, совершенное f-паросочетание может рассматриваться как граф G', у которого $V(G') = V(G)$, каждое ребро параллельно некоторому ребру из G и $|\delta_{G'}(v)| = f(v)$ для всякой вершины v.

Опишем метод сведения задачи об f-факторе к задаче об 1-факторе (или о совершенном паросочетании). Начнем с того, что покажем, как задача об f-паросочетании может быть сведена к задаче о паросочетании. Пусть G – граф и f – целочисленная функция, определенная на $V(G)$. Чтобы исключить тривиальные случаи, предположим, что $f \ge 0$.

Конструируем граф G' следующим образом. Для каждой вершины $v \in V(G)$ пусть U_v – множество, состоящее из $f(v)$ таких элементов, что U_v и U_w не пересекаются, если $v \ne w$. Пусть

$$V(G')=\bigcup_{v\in V(G)} U_v \quad . \tag{30}$$

Соединим всякий элемент из U_v с каждым элементом из U_w, если вершины v и w являются смежными в графе G. Для графа G' справедлива следующая теорема.

Теорема**.** Граф G' обладает совершенным паросочетанием тогда и только тогда, когда граф G обладает совершенным f-паросочетанием [38].

Теперь предположим, что интересует следующий вопрос: обладает ли граф G f-фактором? Построим новый граф G'', подразбив каждое ребро из G двумя новыми вершинами. Расширим область определения функции f до $V(G'')$, положив $f(v) = 1$ для каждой новой вершины. Для графа G'' справедлива следующая теорема.

Теорема**.** Граф G'' обладает совершенным f-паросочетанием тогда и только тогда, когда граф G обладает совершенным f-фактором [38].

Основываясь на этих двух теоремах, можно свести проблему существования и нахождения f-фактора к проблемам существования и нахождения совершенного паросочетания соответственно. Поскольку при таком преобразовании графа каждому f-фактору графа G'' соответствует единственное паросочетание в графе G и наоборот, то данное сведение можно использовать для нахождения f-фактора максимального веса.

Пример нахождения 2-фактора показан на рис. 4-5:

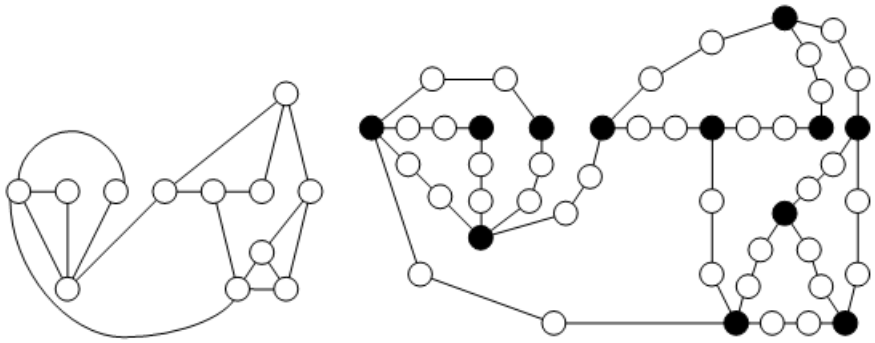

Рис. 4. Граф G и G'

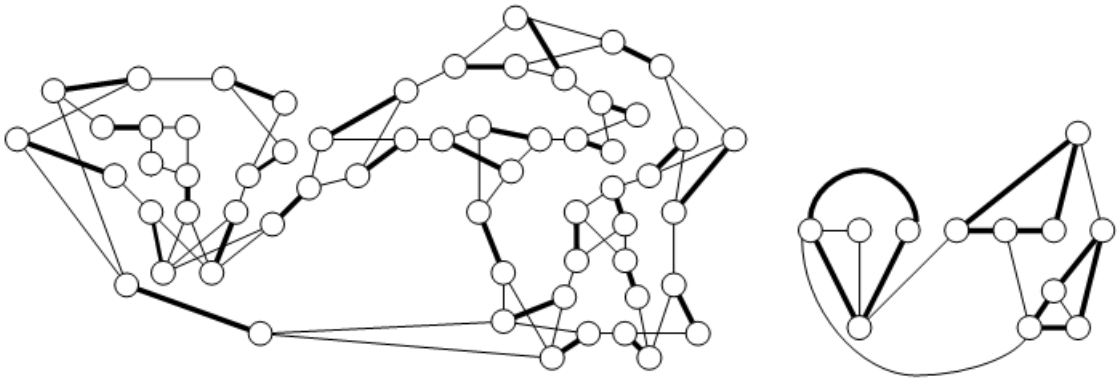

Рис. 5. Граф G' и найденное паросочетание, граф G и найденный 2-фактор

На рисунках выше черными вершинами обозначены вершины, для которых $f(v) = 2$. Белыми вершинами обозначены вершины, для которых $f(v) = 1$.

В связи с тем, что в графе G'' количество вершин зависит от количества ребер исходного графа, можем сделать вывод, что данный алгоритм не позволяет добиться временной оценки $O(n^3)$ (хоть он и остается полиномиальным), однако, несомненным его плюсом является несложная реализация.

Алгоритм Хартвигзена для нахождения максимального 2-паросочетания.

При поиске в графе максимального 2-фактора результатом является множество циклов длины не менее 3, покрывающих вершины данного графа. Если же в графе искать максимальное 2-паросочетание, то некоторые вершины могут покрываться циклом длины 2. Для приближенных алгоритмов решения задачи коммивояжера, использующих поиск подобных подграфов, предпочтительным вариантом является поиск 2-фактора (поскольку циклы в дальнейшем следует разрезать и, следовательно, чем больше ребер в циклах, тем выше вероятность получить большую сумма тура). Однако, для нахождения 2-паросочетания существует алгоритм с временной эффективностью $O(n^3)$. Он был разработан в 1984 году Хартвигзеном [1].

Приведем основные идеи Хартвигзена, использующиеся в его алгоритме. Алгоритм во многом похож на алгоритм Эдмондса. В нем тоже строится чередующийся лес, сжимаются подграфы, есть шаг изменения двойственных переменных. Однако, вместо гипонасыщаемых подграфов (то есть тех подграфов, которые подвергаются сжатию) используются специальная структура, называемая

цветком (заметим, что у Эдмондса тоже есть понятие цветка, но эти понятия следует различать).

Цветок B – это пара (S, T), где $S \subseteq V$ и T – это подмножество $\delta(S)$ нечетной мощности такое, что никакие два ребра T не смежны. Множество S называется центром цветка, ребра T – лепестками. Вершины $V \backslash S$, смежные с лепестками, называются концами лепестков. Цветок изображен на рис. 6.

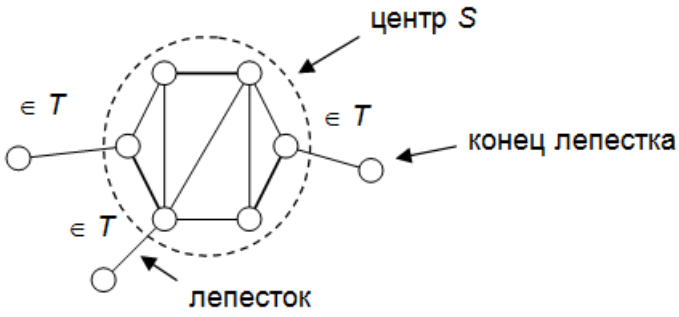

Рис. 6. Цветок

Пусть β – это совокупность цветков $B = (S, T)$ графа G. Для любого ребра $j \in E$ пусть $\beta(j) = \{ B \in \beta : j \in E(B) \}$, и для каждого $B \in \beta$ пусть $s_B = |S| + (|T| - 1) / 2$.

Подобно алгоритму Эдмондса Хартвигзен формулирует задачу линейного программирования, политопом которой является выпуклая оболочка векторов инцидентности 2-паросочетания.

$$\begin{cases} c\,x \to \max & \\ x(\delta(v)) \le 1 & , \forall v \in V \\ x(\gamma(S)) + x(T) \le s_B & , \forall B \in \beta \\ x_e \le 1 & , \forall e \in E \\ x_e \ge 0 & , \forall e \in E, \end{cases} \tag{31}$$

также формулируется двойственная задача:

$$\begin{cases} \sum_{v \in V} y_v + \sum_{B \in \beta} s_B \pi_B + \sum_{e \in E} z_e \to \min & \\ y_u + y_v + \sum_{B \in \beta(e)} \pi_B + z_e \ge c(e) & , \forall e = (u,v) \in E \\ y \ge 0 & , \forall v \in V \\ \pi_B \ge 0 & , \forall B \in \beta \\ z_e \ge 0 & , \forall e \in E. \end{cases} \tag{32}$$

Условия дополняющей нежесткости для прямой и двойственной задач записываются следующим образом:

$$x_e > 0 \Rightarrow y_u + y_v + \sum_{B \in \beta(e)} \pi_B + z_e = c(e) \quad , \quad \forall e = (u,v) \in E \quad , \tag{33}$$

$$\pi_B > 0 \Rightarrow x(\gamma(S)) + x(T) = s_B \quad , \text{ для } \forall B = (\mathrm{S}, \mathrm{T}) \in \beta, \tag{34}$$

$$z_e > 0 \Rightarrow x_e = 1 \quad , \text{ для } \forall e \in \mathrm{E}. \tag{35}$$

$$y_i > 0 \Rightarrow x(\delta(i)) = 2 \quad , \text{ для } \forall \mathrm{i} \in \mathrm{V}. \tag{36}$$

Алгоритм построен таким образом, что на каждом шаге алгоритма условия (33)-(35) выполняются. Алгоритм изменяет x и y до тех пор, пока не выполнится условие (36).

Так как этот алгоритм с помощью идей алгоритма Эдмондса решает еще более сложную задачу, следует ожидать, что он будет сложнее. Так и есть, алгоритм Хартвигзена является очень объемным и труднореализуемым, и, поэтому, в настоящей работе приводится не будет. Подробное его описание можно найти в [1].

3.4. Алгоритм координатного подъема

Алгоритм координатного подъема [6] для решения симметрической задачи коммивояжера.

На входе: полный неориентированный граф $G = (V, E)$ с весовой функцией c.

На выходе: гамильтонов цикл.

Шаг 0. Положим $i = 1$, $T_0 = \oslash$.

Шаг 1. Выберем ребро e_i, имеющее максимальный вес среди допустимых ребер для частичного тура T_{i-1}, и полагаем $T_i = T_{i-1} \cup e_i$. Если $i = n$ завершить работу алгоритма, в противном случае взять $i = i + 1$ и выполнить шаг 1.

Вернем гамильтонов цикл T_n.

Конец алгоритма.

Оценка точности получена в следующей теореме [6].

Теорема. При решении симметрической задачи коммивояжера алгоритмом координатного подъема справедливы следующие достижимые оценки погрешности:

$$\varepsilon = \frac{\left(1 - \frac{1}{\alpha}\right)}{2}, \tag{37}$$

где $\alpha = n$, при нечетном n и $\alpha = n - 1$ при четном n.

Заметим, что в основе данного алгоритма лежит жадная эвристика. Так как симметрическая задача коммивояжера является антиматроидом, то можно сделать вывод, что данный алгоритм на конкретном графе может найти наихудшее из возможных решений.

Трудоемкость алгоритма координатного подъема – $O(n^2)$.

3.5. Градиентный алгоритм с форой

В градиентных алгоритмах с форой на первом этапе с помощью эвристических правил строится некоторый начальный частичный тур (фора), а на втором этапе этот тур достраивается алгоритмом координатного подъема до гамильтонового цикла [6]. В качестве начальных частичных туров могут использоваться решения задачи о паросочетании максимального веса, и частичные туры, получаемые при разрыве циклов, соответствующих решениям задач о назначении и 2-факторе максимального веса. Большинство ε-приближенных алгоритмов (в том числе и

самые удачные) решения задачи коммивояжера на максимум являются алгоритмами с форой. Эвристические правила построения начального частичного тура обычно используют анализ частичных туров, получаемых как при построении паросочетания, так и при построении 2-фактора. Исходя из этого, трудоемкости алгоритмов определяются трудоемкостями решаемых подзадач, а именно задач нахождения максимального 2-фактора, 2-паросочетания или паросочетания. Исходя из этого, при необходимости можно реализовать эти алгоритмы с оценкой времени порядка $O(n^3)$ - $O(n^4)$.

На входе: полный неориентированный граф $G = (V, E)$ с весовой функцией c.

На выходе: гамильтонов цикл.

Шаг 1. Найдем в графе G паросочетание M максимального веса.

Шаг 2. С помощью алгоритма координатного подъема, положив T = M, достроим паросочетание M до гамильтонова цикла.

Вернем гамильтонов цикл T.

Конец алгоритма.

Оценка точности данного алгоритма является результатом следующей теоремы.

Теорема. Неулучшаемая оценка точности данного градиентного алгоритма с форой для симметричной задачи коммивояжера равна 5/8 [6].

3.6. Алгоритм Сердюкова

Для симметрической задачи коммивояжера одним из лучших алгоритмов является алгоритм Сердюкова с оценкой точности 3/4 [4].

В этом алгоритме используется нахождение максимального паросочетания, и необходимо рассмотреть 2 случая: в графе четное число вершин, либо нечетное:

1) n – четно. В этом случае может быть найдено максимальное паросочетание, которое будет покрывать все вершины графа.

На входе: полный неориентированный граф $G = (V, E)$ с весовой функцией c. $|V|$ – четно.

На выходе: Гамильтонов цикл.

Шаг 1. Вычислим максимальный 2-фактор $C = \{C_1, \ldots, C_s\}$. Вычислим максимальное паросочетание M.

Шаг 2. Для каждого цикла C_i, $i = 1, \ldots, s$: Переместим из C_i в M ребро, которое сохранит M частичным туром.

Шаг 3. Дополним C до гамильтонова цикла T_1. Дополним M до гамильтонова цикла T_2.

Вернем больший по весу тур между T_1 и T_2.

Конец алгоритма.

2) n – нечетно. В этом случае для графа не существует совершенного паросочетания, поэтому после нахождение максимального 2-фактора и максимального паросочетания необходимо определенным образом изменить эти

подграфы и использовать алгоритм, предназначенный для четного размера задачи.

На входе: полный неориентированный граф $G = (V, E)$ с весовой функцией c. $|V|$ – нечетно.

На выходе: Гамильтонов цикл.

Шаг 1. Вычислим максимальный 2-фактор $C = \{C_1, \dots, C_s\}$. Вычислим максимальное паросочетание M.

Шаг 2. Обозначим за v^0 вершину, не инцидентную никакому ребру из M. Пусть $v^0 \in C_1$. Найдем ребро $e_0 = (v^0, v') \notin C$ максимального веса $c(e_0)$. Обозначим за $e_1, e_2 \in C_1$ ребра, инцидентные вершине v^0.

Если $v' \in C_1$, то положим $C_0 = C_1$ и $j = 2$.

Если $v' \in C_2$, то положим $C_0 = C_1 \cup C_2$ и $j = 3$.

В $C_0 \setminus M$ выберем ребро $e' \in \{e_1, e_2\}$ и ребро e'', инцидентное вершине v'. В случае $C_0 = C_1$ следует выбрать такое ребро e', чтобы не образовался цикл через вершину v^0.

Обозначить $C = e_0 \cup C \setminus \{e', e''\}$, $M = M \cup \{e', e''\}$.

Шаг 3. Для $i = j, \dots, s$: Переместим из C_i в M ребро, которое сохранит M частичным туром.

Шаг 4. Дополним C и M до гамильтоновых циклов T_1 и T_2, соответственно.

Вернем больший по весу тур между T_1 и T_2.

Конец алгоритма/

3.7. Улучшенный алгоритм Сердюкова

Улучшенный алгоритм Сердюкова строит три тура и выбирает из них тот, который имеет наибольший вес.

На входе: полный неориентированный граф $G = (V, E)$ с весовой функцией c, константа ε.

На выходе: гамильтонов цикл.

Шаг 1. Вычислим максимальный 2-фактор $C = \{C_1, \dots, C_s\}$.

Шаг 2. Вычислим гамильтонов цикл T_1 с помощью алгоритма A_1 ($T_1 = A_1(G, C, \varepsilon)$).

Шаг 3. Вычислим гамильтоновы циклы T_2 и T_3 с помощью алгоритма A_2 ($(T_2, T_3) = A_2(G, C)$).

Вернем больший по весу цикл между T_1, T_2 и T_3.

Конец алгоритма.

Первый тур построен по алгоритму A1, как в [39]. Алгоритм использует параметр $\varepsilon > 0$. Он рассматривает по-разному 2-факторы, которые удовлетворяют условию $¿C_i \vee \le \varepsilon^{-1}$ и нет. В первом случае он вычисляет максимальный гамильтонов путь на его вершинах. Во втором случае удаляется ребро минимального веса. Из получившихся гамильтоновых путей формируется тур T_1.

Алгоритм A_1 (новый алгоритм).

На входе: полный неориентированный граф $G=(V,E)$ с весовой функцией c, 2-фактор $C=\{C_1,\ldots,C_s\}$, константа ε.

На выходе: гамильтонов цикл.

Шаг 1. Для $i=1,\ldots,s$: Если ¿$C_i \vee$ ¿❑$^{-1}$, то вычислим максимальный взвешенный гамильтонов путь H_i в подграфе графа G, индуцированном вершинами цикла C_i. В противном случае, если $|C_i| \geq \varepsilon^{-1}$, то найдем ребро минимального веса e_i в цикле C_i. Обозначить $H_i=$¿$C_i\{e_i\}$.

Шаг 2. Соединим H_i, любым способом (например, алгоритмом координатного подъема) для получения гамильтонового цикла T_1.

Вернем гамильтонов цикл T_1.

Конец алгоритма.

Второй алгоритм является модифицированной версией алгоритма Сердюкова. Он передает ребра из C в W и генерирует два частичных тура. Множество, образованное из W с перенесенными ребрами, увеличивается произвольно до тура T_2. Другое множество, состоящее из оставшихся ребер из C, сначала дополняется новыми ребрами, инцидентные вершины которых принадлежат разным циклам С. Затем он произвольно дополняется до тура T_3.

Алгоритм A_2 (модифицированная версия алгоритма Сердюкова).

На входе: полный неориентированный граф $G=(V,E)$ с весовой функцией c, 2-фактор $C=\{C_1,\ldots,C_s\}$.

На выходе: гамильтоновы циклы T_2 и T_3.

Шаг 1. Пусть E' – это такие ребра графа G, концы которых находятся в разных циклах 2-фактора C.

Шаг 2. Вычислим максимальное взвешенное паросочетание $M' \subseteq E'$.

Шаг 3. Вычислим максимальное взвешенное паросочетание W в графе G.

Шаг 4. Для $i=1,\ldots,s$: Вычислим непересекающиеся непустые паросочетания M_i и M'_i на ребрах цикла C_i такие, что $M_i \cup W$ и $M'_i \cup W$ это туры и каждая вершина C_i инцидентна хотя бы одному ребру из $M_i \cup M'_i$. Перенести ребра либо M_i, либо M'_i (паросочетания выбираются равновероятно) из C_i в W.

Шаг 5. Дополним W до гамильтонового цикла T_2 (любым методом, например, алгоритмом координатного подъема).

Шаг 6. Пусть P – это множество путей, полученных из С после переноса ребер. Пусть множество ребер $M=\{(i,j)\in M': \vee$❑$_P(i) \vee$ ¿$\vee$ ❑$_P(i) \vee$ ¿$1\}$. Рассмотрим $M \cup P$. Пусть это множество состоит их путей $P^¿=\{P^¿_1,\ldots,P^¿_s\}$ и циклов $C^¿=\{C^¿_1,\ldots,C^¿_t\}$.

Шаг 8. Для $i=1,\ldots,t$: Наугад выберем ребро $e C^¿_i \cup M$. Изменим $P^¿: P^¿=P^¿ \cup$ ¿$C^¿_i)$.

Шаг 9. Дополним $P^¿$ до гамильтонового цикла T_3 (любым методом,

например, алгоритмом координатного подъема).

Вернем гамильтоновы циклы T_2 и T_3 .

Конец алгоритма.

Лемма**.** Когда Алгоритм А2 рассматривает C_i , можно построить непустые паросочетания M_i и M'_i такие, что $M_i \cup W$ и $M'_i \cup W$ являются частичными турами, а каждая вершина из C_i инцидентна хотя бы одному ребру из $M_i \cup M'_i$.

Доказательство. Пусть $e_1,\ldots,e_k$ – ребра из C_i , составляющие цикл.

Перебираем ребра из C_i , начиная с e_1 . Поочередно добавляем ребра из C_i в M_i и M'_i . Если такое добавление (например, e_j в M_i) создаст цикл в $M_i \cup W$ (в частности, если это ребро уже есть в W), то ребро e_j пропускается и берется следующее e_{j+1} . Одно из добавлений всегда возможно, поэтому никогда не пропустятся два последовательных ребра.

При рассмотрении последнего ребра в C_i может возникнуть конфликт: добавление двух ребер e_1 и e_k к множеству M_i . Решение может быть следующим: если e_2 было присвоено M'_i , то ребро e_1 просто пропускается. Иначе, если было пропущено ребро e_2 , т.е. оно не добавилось в M'_i , тогда можно добавить ребро e_1 к M'_i . Таким образом, в этом случае ребро e_1 добавляется к множеству M'_i , а не к M_i .

Второй конфликт может произойти, если оба ребра e_1 и e_k будут пропущены. Таким образом, ребро e_1 не вошло ни в одно из множеств M_i или M'_i . В этом случае добавляем ребро e_1 к M'_i .

При этом сохраняется свойство, что каждая вершина из C_i инцидентна по крайней мере одному ребру из $M_i \cup M'_i$, данное объединение содержит по крайней мере одно ребро. □

Стоит отметить, что оба множества M_i и M'_i являются непустыми, поэтому после переноса любого из этих паросочетаний к W по крайней мере одно ребро из каждого цикла будет перенесено, а остальные ребра образуют частичный тур.

Лемма. Для каждой вершины из C_i вероятность того, что одно из ребер, инцидентных ей в C_i , будет добавлено в W по алгоритму А2, не меньше 1/2.

Лемма. Для каждого ребра $e \in M'$ вероятность того, что оно есть в M (т.е. две инцидентных вершины имеют степень 1 в P), по крайней мере 1/4.

В силу того, что каждый цикл из $C_1^i,\ldots,C_t^i$ содержит по крайней мере два ребра из M, получаем:

Лемма. Для каждого ребра $e \in M$ вероятность того, что оно будет удалено в алгоритме А2, не больше 1/2.

Теорема.

$$max\{w(T_1), w(T_2), w(T_3)\} \geq \frac{25(1-\varepsilon)}{33-32\varepsilon}\mu \quad . \tag{38}$$

Доказательство. Пусть T - оптимальный гамильтонов цикл. Определим $\int_{T_i}^{i}(T_{ext})$ как ребра из T, у которых концевые вершины находятся в одной и той же (в разных) компонентах связности C. Пусть $\int_{T_i}^{i}$ w i. Рассмотрим тур T_1. Для каждого цикла из C алгоритма A1 вычисляется гамильтонов путь максимального веса и, следовательно, его вклад в вес T_1 равен по крайней мере весу $\int_{T_i}^{i}$ в подграфе, индуцированном его вершинами. Так как C – максимальный 2-фактор, то $w(C_i)$ является по крайней мере весом $\int_{T_i}^{i}$ в подграфе, индуцированном вершинами из C_i. В каждом образованном цикле удаляется ребро минимального веса, вычитая, таким образом, из его суммарного веса не более коэффициента ε. Следовательно, $\int_{T_i}^{i}$ $w(T_1)\geq(1-\varepsilon)w$ i.

Теперь рассмотрим T_2 и T_3. Пусть $\delta\mu$ – общий вес ребер, перенесенных из C в W. Так как исходный вес W не менее $\frac{1}{2}\mu$, то $w(T_2)\geq\left(\frac{1}{2}+\delta\right)\mu$.

Вес множества P, состоящего из путей, сформированных из C после переноса ребер, равен по крайней мере $(1-\delta)\mu$. Далее ребра добавляются следующим образом: сначала вычисляется максимальное паросочетание M'. $w(M')\geq\frac{1}{2}w(T_{ext})$, так как T_{ext} можно покрыть двумя непересекающимися паросочетаниями в G'. Затем формируется множество M, удаляя все ребра из M', кроме тех, у которых инцидентные вершины имеют степень 1 в P. По лемме 3, каждое ребро из G' имеет с вероятностью 1/4 два конца, которые имеют степень 1 в P. Следовательно, $w(M)\geq\frac{1}{4}w(M')\geq\frac{1}{8}w(T_{ext})=\frac{1}{8}(1-\alpha)\mu$.

Далее рассматриваются ребра из M в множестве $M\cup P$ и удаляются ребра $e\in M$ с вероятностью не более 1/2. Ожидаемый вес оставшихся ребер не менее $\frac{1}{2}w(M)\geq\frac{1}{16}(1-\alpha)\mu$. Наконец, соединяя оставшиеся ребра с P, получится тур T_3. Таким образом, $(T_3)\geq\left((1-\delta)+\frac{1}{16}(1-\alpha)\right)\mu$.

В заключение, получаем

$$max\left\{w(T_1),w(T_2),w(T_3)\right\}\geq max\left\{(1-\varepsilon)\alpha,\frac{1}{2}+\delta,\frac{17}{16}-\delta-\frac{\alpha}{16}\right\}\mu. \quad (39)$$

Минимальное значение правой части неравенства получается при $\alpha=\frac{25}{33-32\varepsilon}$, и тогда оно равно $\frac{25(1-\varepsilon)}{33-32\varepsilon}\mu$.

3.8. Выводы по разделу

В данном разделе были рассмотрены точные и приближенные алгоритмы решения задачи коммивояжера на максимум. Среди методов точного решения было отдано предпочтение методу ветвей и границ, поскольку метод динамического программирования менее эффективен.

Для реализации приближенных алгоритмов решения задачи коммивояжера на максимум используются следующие вспомогательные алгоритмы: алгоритм нахождения наибольшего паросочетания, алгоритм нахождения максимального взвешенного паросочетания, алгоритм нахождения 2-фактора сведением к задаче о паросочетании. Рассмотрены приближенные алгоритмы: алгоритм координатного подъема, градиентный алгоритм с форой, алгоритм Сердюкова.

В основе алгоритма координатного подъема лежит жадная эвристика. Так как симметрическая задача коммивояжера является антиматроидом, то можно сделать вывод, что данный алгоритм на конкретном графе может найти наихудшее из возможных решений. В градиентном алгоритме с форой на первом этапе с помощью эвристических правил строится некоторый начальный частичный тур (фора), а на втором этапе этот тур достраивается алгоритмом координатного подъема до гамильтонового цикла. В данном разделе также приведен алгоритм Сердюкова, затем на основе алгоритмов из [50] и [40] формируется улучшенный алгоритм Сердюкова, решающий задачу коммивояжера на максимум, с оценкой точности 25/33. Несмотря на то, что улучшение оценки невелико, оно, по крайней мере, демонстрирует, что оценка 3/4 не является наилучшей.

4. РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ, РЕАЛИЗУЮЩЕЙ АЛГОРИТМЫ РЕШЕННИЯ ЗАДАЧИ КОММИВОЯЖЕРА НА МАКСИМУМ

4.1. Программа MAX TSP

Для решения поставленной задачи была разработана программа, реализующая точный и приближенные алгоритмы решения задачи коммивояжера на максимум. Программа написана на языке C++ с использование среды разработки Turbo C++. На вход программе подается матрица весов симметричного графа. Результат работы алгоритмов содержит наибольший вес гамильтонова цикла и время работы выбранного алгоритмы. Граф не должен содержать петель и кратных ребер. Матрицу весов можно задать случайным образом или загрузить из файла.

Решить задачу коммивояжера на максимум можно выбранным алгоритмом из выпадающего списка и нажатием кнопки «Получить решение», при этом результат появится в пустом поле в левой части окна (рис. 7). Результаты можно удалить или сохранить в файл нажатием соответствующей кнопки, расположенной ниже поля.

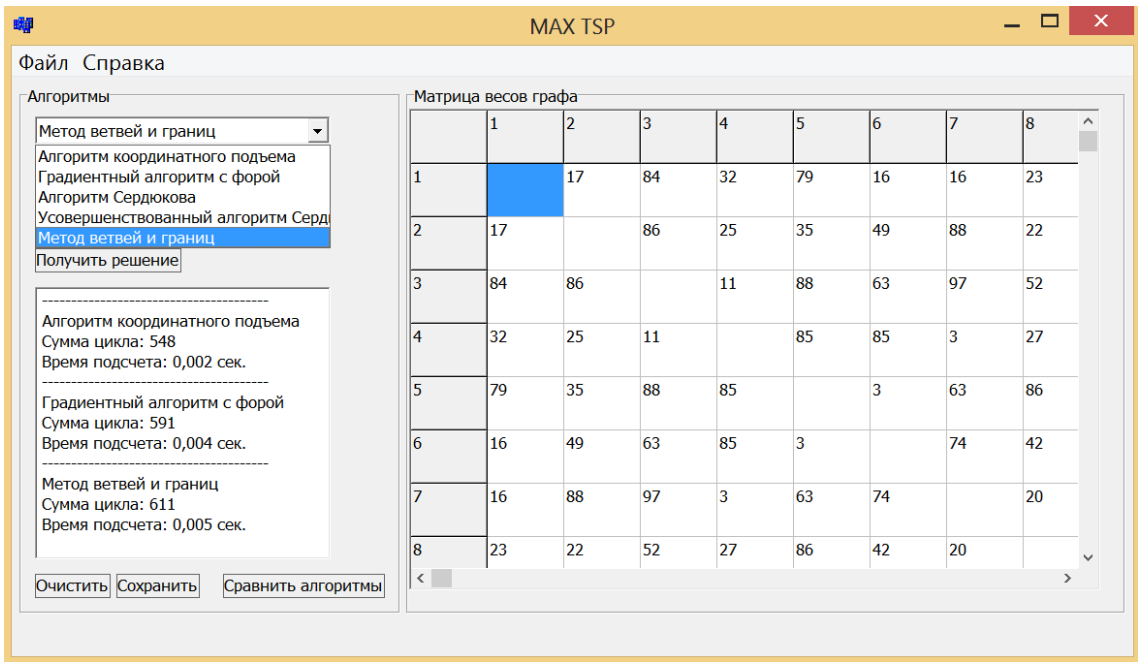

Рис. 7. Результат работы алгоритмов

Для анализа алгоритмов необходимо нажать кнопку «Сравнить алгоритмы», в результате чего появится новое окно. В данном окне требуется выбрать алгоритмы, сравнение которых будет производиться, и график, который необходимо построить, затем нажать кнопку «Пуск». Программа выполнит поиск гамильтонового цикла для выбранных алгоритмов на случайных графах и построит выбранный график, который появится в правой части окна (рис. 8).

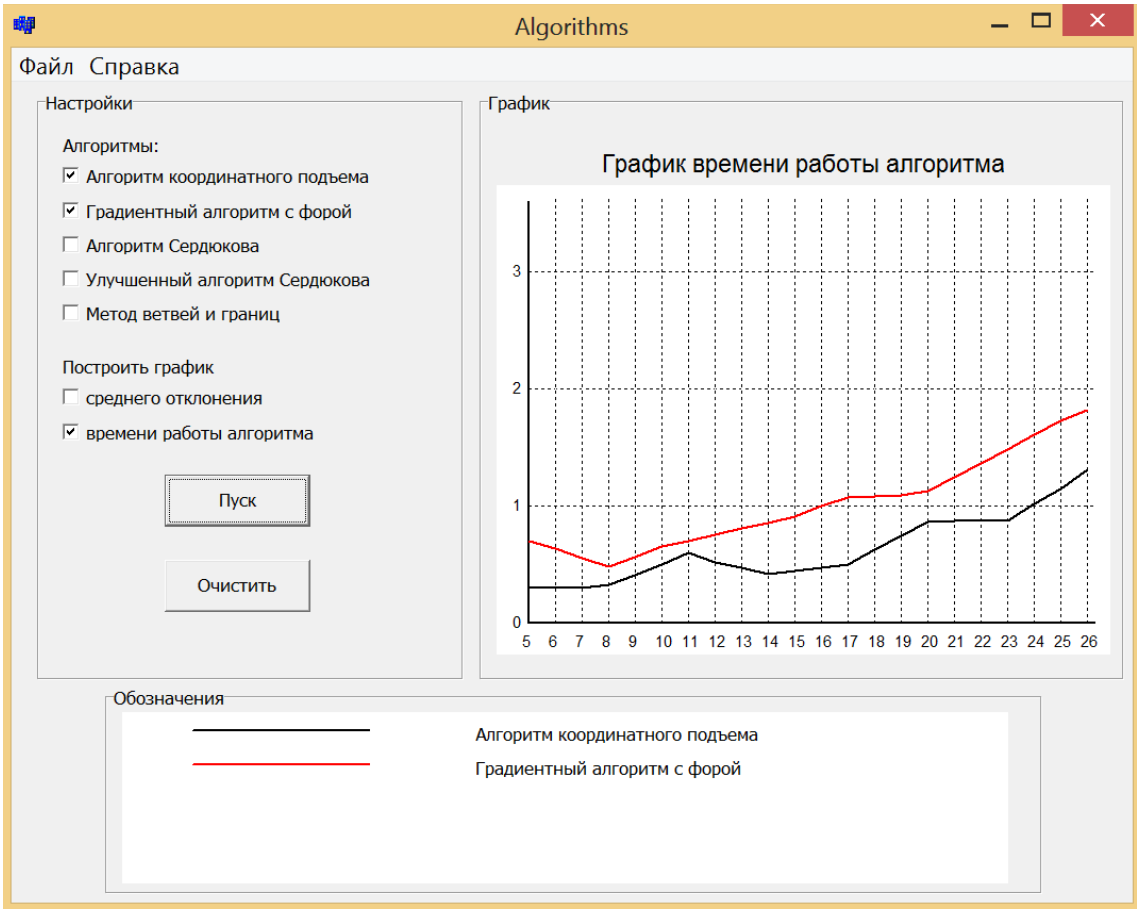

Рис. 8. Построение графика, сравнивающего алгоритмы

В программе для нахождения гамильтоновых циклов реализованы следующие алгоритмы:

- алгоритм координатного подъема;
- градиентный алгоритм с форой;
- алгоритм Сердюкова;
- улучшенный алгоритм Сердюкова;
- метод ветвей и границ.

Помимо этого реализованы следующие вспомогательные алгоритмы:

- нахождение гамильтонового пути максимального веса полным перебором;
- нахождение паросочетания максимального веса;
- нахождение 2-фактора максимального веса.

Нахождение паросочетания максимального веса производится с помощью эффективной реализации алгоритма Эдмондса, разработанной Габовым в работе [44]. Поиск максимального 2-фактора выполняется сведением к задаче нахождения паросочетания максимального веса.

Таким образом, все рассмотренные в данной работе алгоритмы, за исключением метода динамического программирования, реализованы в программе. Среди методов точного решения MAX TSP было отдано предпочтение методу ветвей и границ, поскольку метод динамического программирования менее эффективен.

Программа подсчитывает следующие параметры для каждого алгоритма:

- среднее отклонение от наилучшего (среди всех алгоритмов) найденного решения для каждой размерности графа;
- среднее время работы алгоритма для каждой заданной размерности графа.

Результаты измерения этих параметров программа выдает в виде графиков, в которых осью абсцисс является размерность графа, а осью ординат – либо время в миллисекундах, либо отклонение в процентах.

Более подробная информация о программе приведена в приложениях 1 и 2. Код программы приведен в приложении 3.

4.2. Анализ алгоритмов

Сравнение проводилось на размерностях графов от 5 до 26. Графы размерности больше 26 часто обсчитываются методом ветвей и границ за значительно большее время, поэтому решено было ограничиться этими размерностями. На каждом шаге обсчитывалось 20 графов.

Для начала сравним алгоритм координатного подъема и градиентный алгоритм с форой. На рис. 9 построен график среднего отклонения двух алгоритмов. Среднее отклонение от оптимума для найденных алгоритмом координатного подъема решений не превышало 0,75%, для градиентного алгоритма с форой – 0,68%. На рис. 10 наглядно изображено, что градиентный алгоритм с форой работает быстрее алгоритма координатного подъема.

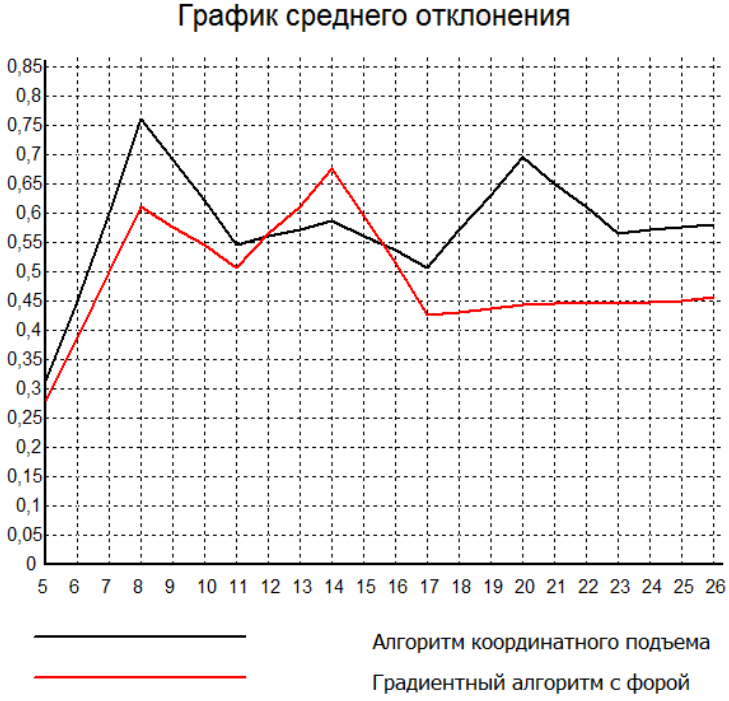

Рис. 9.График среднего отклонения алгоритма координатного подъема и градиентного алгоритма с форой

Рис. 10. График времени работы алгоритма координатного подъема и градиентного алгоритма с форой

При сравнении алгоритма Сердюкова и улучшенного алгоритма Сердюкова, второй алгоритм находит решения более близкие к оптимальному (рис. 11). Среднее отклонение улучшенного алгоритма Сердюкова не превышает 0,3%. Время работы двух алгоритмов отличается несущественно (рис. 12).

Рис. 11. График среднего отклонения алгоритма Сердюкова и улучшенного алгоритма Сердюкова

Рис. 12. График времени работы алгоритма Сердюкова и улучшенного алгоритма Сердюкова

График времени работы метод ветвей и границ приведен отдельно (рис. 13), так как с увеличение размерности графа значительно увеличивается и время работы алгоритма. Для графа с числом вершин 26. время работы алгоритма составляет порядка 475 миллисекунд.

Рис. 13. График времени работы метода ветвей и границ

На рис. 14 приведен график среднего отклонения для всех исследуемых алгоритмов. Наилучшие результаты показал улучшенный алгоритм Сердюкова.

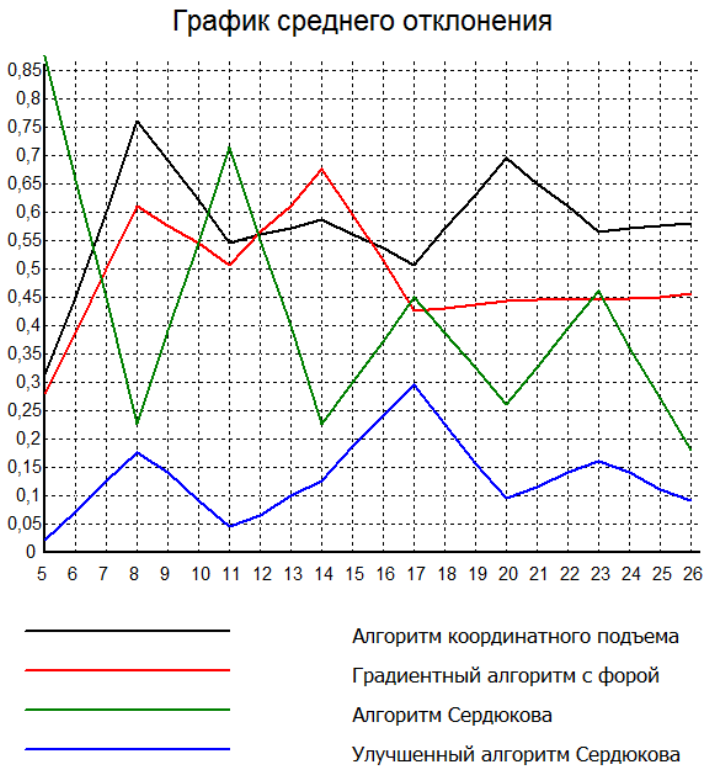

Рис. 14. График среднего отклонения для всех исследуемых приближенных алгоритмов

На графике времени работы всех исследуемых алгоритмов (рис. 15) размерность графов изменяется от 5 до 10, так как время работы метода ветвей и границ на графах большей размерности существенно велико по сравнению с другими алгоритмами.

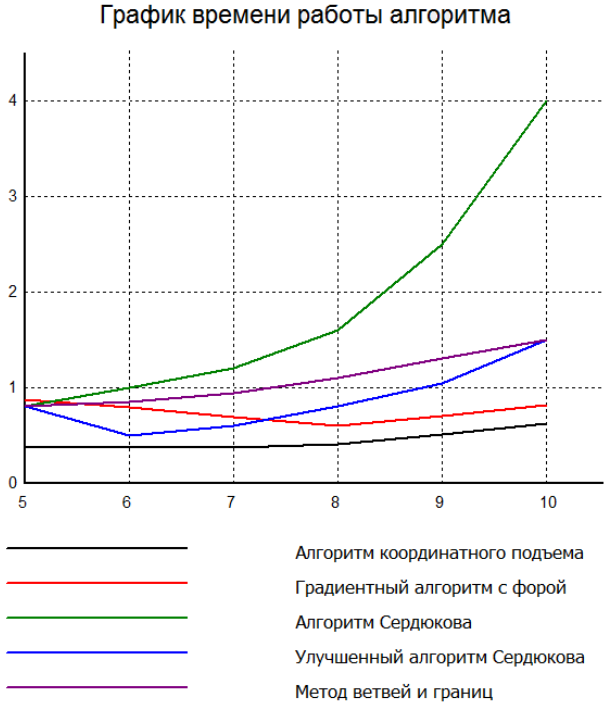

Рис. 15. График времени работы всех исследуемых алгоритмов

4.3. Выводы по разделу

В данном разделе описывается программная реализация изученных алгоритмов для решения задачи коммивояжера на максимум и проводится сравнительный анализ исследуемых алгоритмов. Программа решает поставленную задачу на симметрических графах, заданных случайным образом или загружаемых из файлов. В результате работы программы находится гамильтонов цикл максимального веса, строятся графики среднего отклонения и времени работы алгоритмов.

Самым точным из протестированных ε-приближенных алгоритмов можно признать улучшенный алгоритм Сердюкова. Он показал это при тестировании на случайных симметрических графах. Наибольшее отклонение данного алгоритм от оптимального решения составило 0.3%.

Самым долгим по времени работы оказался метод ветвей и границ на больших размерностях графа, при небольшом количестве вершин он не на много уступает другим алгоритмам.

ЗАКЛЮЧЕНИЕ

В работе был проведен анализ научной и учебной литературы по теме исследования, была поставлена задача коммивояжера на максимум, рассмотрены виды данной задачи. Исследуется теория антиматроидов, приводится доказательство, что задача коммивояжера на максимум является антиматроидом.

Данная работа поволяет изучить различные ε-приближенные алгоритмы решения задачи коммивояжера на максимум. В работе рассмотрены самые последние разработки в этой области. Большинство описанных алгоритмов были разработаны недавно и публиковались только в англоязычных статьях. В работе была рассмотрена теория антиматроидов, позволившая оценить эффективность использования жадных алгоритмов для решения некоторых задач дискретной оптимизации.

Изучение задачи коммивояжера на максимум имеет большое практическое применение. Главной задачей, для решения которой применяется задача коммивояжера на максимум, является проблемы нахождения кратчайшей суперстроки, также существуют приложения этой задачи к проблемам эффективного сжатия произвольных данных и анализа последовательностей ДНК.

В работе разработан улучшенный алгоритм Сердюкова, который эффективнее алгоритма Сердюкова как по точности решения, так и по времени работы. Улучшенный алгоритм Сердюкова решает задачу коммивояжера на максимум с оценкой точности 25/33. Несмотря на то, что улучшение оценки невелико, оно, по крайней мере, демонстрирует, что оценка 3/4 для алгоритма Сердюкова не является наилучшей.

Написанная программа на языке высокого уровня реализует приближенные алгоритмы решения задачи коммивояжера на максимум. В результате задачу коммивояжера на максимум можно решить на симметричном графе, заданном случайным образом или заданным пользователем, следующими алгоритмами: алгоритмом координатного подъема, градиентным алгоритмос с форой, алгоритмом Сердюкова и его модификацией, а также методом ветвей и границ, который является точным алгоритмом. Программа также позволяет находить гамильтоновые циклы в произвольных симметрических графах. Программный интерфейс является интуитивно понятным, что облегчает работу с программой.

В данной работе получен сравнительный анализ алгоритмов по точности решения и времени работы. В результате анализа решений алгоритмов на случайных графах был сделан вывод, что лучшим алгоритмом для приближенного решения задачи коммивояжера является улучшенный алгоритм Сердюкова.

Разработанная программа решает только симметрическую задачу коммивояжера на максимум, веса ребер графа нельзя изменить в самой

программе, необходимо провести изменения в текстовом файле, откуда загружался граф. Также сравнительные графики не будут наглядны для графов большой размерности, при решении задачи методом ветвей и границ при большом количестве вершин программе потребуется сравнительно больше времени, чем для какого-либо приближенного алгоритма. Несмотря на недостатки программы, она решает симметрическую задачу коммивояжера на максимум, находит гамильтонов цикл максимального веса и позволяет сравнить приближенные алгоритмы.

ЛИТЕРАТУРА

1. Асанов, М.О. Дискретная математика: графы, матроиды, алгоритмы / М.О. Асанов, В.А. Баранский, В.В. Расин. – Ижевск: НИЦ "Регулярная и хаотическая динамика", 2001. – 288 с.
2. Ахо, А. Структуры данных и алгоритмы / А. Ахо, Дж. Хопкрофт, Д. Ульман. – М.: Издательский дом «Вильямс», 2001. – 384 с.
3. Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М.: Мир, 1982. – 419 с.
4. Гимади, Э.Х. Алгоритм для приближенного решения задачи коммивояжера и его вероятностный анализ / Э.Х. Гимади, Н.И. Глебов, А.И. Сердюков. – Сиб. журн. исслед. операций, 1994. – №2. – 8-17 с.
5. Гимади, Э.Х. О некоторых результатах для задачи коммивояжера на максимум / Э.Х. Гимади, А.И. Сердюков. – Дискретный Анализ и Исследование Операций, 2001. – №1. – 22-39 с.
6. Ковалев, М.М. Матроиды в дискретной оптимизации / М.М. Ковалев. – Минск: Изд-во «Университетское», 1987. – 222 с.
7. Кормен, Т. Алгоритмы. Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: Вильямс, 2009. – 394 с.
8. Ловас, Л. Прикладные задачи теории графов. Теория паросочетаний в математике, физике, химии / Л. Ловас, М. Пламмер. – М.: Мир, 1998. – 653 с.
9. Панюков, А.В. Применение дополнений паросочетаниями для решения задачи MAX TSP / А.В. Панюков, С.А. Тычинин. – Вестник ЮУрГУ. Серия Математическое моделирование и программирование, 2008. – №15. – 54-63 с.
10. Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц. – М.: Мир, 1985. – 512 с.
11. Рейнгольд, Э. Комбинаторные алгоритмы. Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М.: Мир, 1980. – 478 с.
12. Сергеев, С.И. Вычислительные алгоритмы решения задачи коммивояжера / С.И. Сергеев. – АиТ, 1994. - №6. – 106-114 с.
13. Эвнин, А.Ю. Антиматроиды / А.Ю. Эвнин. – Математическое образование, 2008. – №1. – 2-8 с.
14. Эвнин, А.Ю. Дискретная математика: Конспект лекций / А.Ю. Эвнин. – Челябинск: ЮУрГУ, 1998. – 176 с.
15. Эвнин, А.Ю. Дискретная оптимизация: Избранные вопросы / А.Ю. Эвнин. – Челябинск: ЮУрГУ, 2004. – 78 с.
16. Эвнин, А.Ю. Элементы дискретной оптимизации: уч. пособие / А.Ю. Эвнин. – Челябинск: ЮУрГУ, 2012. – 92 с.
17. Aiello, A. Computational complexity: the problem of approximation / A. Aiello, E. Burattini, M. Massarotti. – In C.M.S.J. Bolyai, Algebra, combinatorics, and logic in computer science, 1986. – Vol. 36. – 51-62 p.

18. Barvinok, A.I. Finding maximum length tours under polyhedral norms / A.I. Barvinok, D.S. Johnson, G.J. Woeginger. – Lecture Notes in Computer Science, 1998. – Vol. 1412. – 195-201 p.
19. Barvinok, A.I. The maximum traveling salesman problem / A.I. Barvinok, S.P. Fekete, D.S. Johnson. – Technical Report University of Koln, 1998. – Vol. 112. – 98-333 p.
20. Barvinok, A.I. Two algorithmic results for the traveling salesman problem / A.I. Barvinok. – Mathematics of Operations Research, 1998. – Vol. 21. – 65-84 c.
21. Blokh, D. Maximizing traveling salesman problem for special matrices / D. Blokh, G. Gutin. – Discrete Appl. Math., 1995. – Vol. 56. – 83-86 p.
22. Chalasani, P. Approximating capacitated routing and delivery problems / P. Chalasani, R. Motwani. – SIAM J. Comput, 1999. – Vol. 28. – 2133-2149 p.
23. Chen, Zh. Improved Deterministic Approximation Algorithms for Max TSP / Zh. Chen, Y. Okamoto, L. Wang. – Inform. Process. Lett., 2005. – Vol. 95. – 333-342 p.
24. Christos, H. The traveling salesman problem with distances one and two / H. Christos. – Mathematics of Operations Research, 1993. – Vol. 18. – 1-11 p.
25. Croes, A. A method for solving traveling salesman problems / A. Croes. – Oper. Res., 1978. – Vol. 5. – 791-812 p.
26. Demange, M. On a approximation measure founded on the links between optimization and polynomial approximation theory / M. Demange, V.T. Paschos. – Theoret. Comput. Sci., 1996. – Vol. 158. – 117-141 p.
27. Engebretsen, L. Approximation hardness of TSP with bounded metrics / L. Engebretsen, M. Karpinski. – Lecture Notes in Computer Science, 2001. – Vol. 2076. – 201-212 p.
28. Fekete, S.P. On minimum stars and maximum matchings / S.P. Fekete. – Discrete & Computational Geometry, 2000. – Vol. 23. – 389-407 p.
29. Fekete, S.P. Good and fast heuristics for large geometric maximum matching and maximum traveling salesman problems / S.P. Fekete, H. Meijer, A. Rohe. Journal of Experimental Algorithms, 2000. – Vol. 17. – 74-90 p.
30. Fekete, S.P. Simplicity and hardness of the maximum traveling salesman problem under geometric distances / S.P. Fekete. – New York: ACM, 1999. – 337-345 p.
31. Fekete, S.P. Solving a "Hard" Problem to Approximate an "Easy" One: Heuristics for Maximum Matchings and Maximum Traveling Salesman Problems / S.P. Fekete, H. Meijer, A. Rohe. – Journal of Experimental Algorithms, 2002. – Vol. 7. – 131-151 p.
32. Fisher, M.L. An analysis of approximations for finding a maximum weight Hamiltonian circuit / M.L. Fisher, G.L. Nemhauser, L.A. Wolsey. – Oper. Res., 1989. – Vol. 27. – 799-809 p.

33. Gabow, H. An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs / H. Gabow. – Journal of the ACM, 1986. – Vol. 4. – 221-234 p.
34. Gutin, G. Anti-matroids / G. Gutin. – Oper. Res. Lett, 2002. – Vol. 30. – 97-99 p.
35. Gutin, G. Exponential neighborhood local search for the traveling salesman problem / G. Gutin. – Comput. Oper. Res., 1999. – Vol. 26. – 313-320 p.
36. Gutin, G. The greedy algorithm for the symmetric TSP / G. Gutin. – Oper. Res. Lett., 2005. – Vol. 33. – 87-89 p.
37. Gutin, G. Traveling Salesman Problem and Its Variations / G. Gutin, A. Punnen. – Kluwer Academic Publishers, 2002. – 848 p.
38. Hartvigsen, D. Extensions of matching theory / D. Hartvigsen. – Pittsburg, PA: Carnegie Mellon Univ, 1984. – 148 p.
39. Hassin, R. A 7/8 -approximation algorithm for metric Max TSP / R. Hassin, S. Rubinstein. – Inform. Process. Lett, 2002. – Vol. 81. – 247-251 p.
40. Hassin, R. An approximation algorithm for the maximum traveling salesman problem / R. Hassin, Sh. Rubinstein. – Inform. Process. Lett, 2000. – Vol. 67. – 125-130 p.
41. Kaplan, H. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs / H. Kaplan, M. Lewenstein, N. Shafrir. – Journal of the ACM, 2005. – Vol. 4. – 602-626 p.
42. Kostochka, A.V. Polynomial algorithms with the estimates 3/4 and 5/6 for the traveling salesman problem of the maximum / A.V. Kostochka, A.I. Serdyukov. – Upravlyaemye Sistemy, 1985. – Vol. 26. – 55-59 p.
43. Kowalik, L. 35/44-approximation for asymmetric maximum tsp with triangle inequality / L. Kowalik, M. Mucha. – Algorithmica, 2011. – Vol. 59. – 240-255 p.
44. Kowalik, L. Deterministic 7/8-approximation for the metric maximum tsp / L. Kowalik, M. Mucha. – Theor. Comput. Sci., 2009. – Vol. 17. – 237-248 p.
45. Lewenstien, M. A 5/8 approximation algorithm for the Maximum TSP / M. Lewenstien, M. Sviridenko. – SIAM Journal on Discrete Mathematics, 2004. – Vol. 2. – 237-248 p.
46. Little, J.D.C. An algorithm for the Traveling Salesman Problem / J.D.C. Little, K.G. Murty, D.W. Sweeney. – Operations Research, 1983. – Vol. 11. – 972-989 p.
47. Oxley, J. Matroid Theory / J. Oxley. – Oxford Univ. Press., 1992. – 459 p.
48. Papadimitriou, C.H. On the approximability of the traveling salesman problem / C.H. Papadimitriou, S. Vempala. – In Proc. STOC'00, 2000. – Vol. 21. – 126-133 p.
49. Rose, J. Algorithmic aspects of vertex elimination on graphs / J. Rose, R. Endre. – SIAM J. Comput, 1985. – Vol. 75. – 266-283 p.
50. Serdyukov, A.I. An algorithm with an estimate for the traveling salesman problem of the maximum / A.I. Serdyukov. – Upravlyaemye Sistemy, 1984. – Vol. 25. – 80–86 p.