Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, _____
_____
_____(И.О. Ф.)
«____»_____2017г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____/А.А.Замышляева
«____»_____2017 г.

Задачи об устойчивых паросочетаниях

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–010402.2017.146.ПЗ ВКР

Руководитель работы, к.п.н.
_____/ А.Ю. Эвнин
«____»_____2017 г.
Автор работы
Студент группы ЕТ-222
_____/ Н.М. Сапронов
«____»_____2017 г.
Нормоконтролер, к.э.н., доцент кафедры ПМиП
_____/Д.А. Дрозин
«____»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Сапронов Н.М.. Задачи об устойчивых паросочетаниях. – Челябинск: ЮУрГУ, ЕТ-222, 55 с., 8 ил., 1 табл., библиогр. список – 50 наим., 1 прил.

В работе рассмотрены основные нерешённые задачи, связанные с устойчивыми паросочетаниями. Проанализированы методы и подходы к решению данных задач. Разработаны алгоритмы решения некоторых актуальных задач об устойчивых паросочетаниях. Улучшен алгоритм поиска устойчивого паросочетания на полном графе. Реализована программа поиска устойчивого паросочетания с помощью оптимизированного алгоритма.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В последнее время часто возникает задача распределения объектов или людей в пары друг с другом. Например, распределение сотрудников по вакансиям, формирование комитетов, распределение абитуриентов по вузам.

Для решения подобных задач удобно воспользоваться теорией графов. Составив некоторый двудольный граф, можно заметить, что задача сводится к известной задаче о поиске паросочетания. Однако на практике зачастую не находится максимального паросочетания, удовлетворяющего всем потребностям в полной мере. В этом случае можно упорядочить предпочтения каждого объекта относительно своей пары. Для решения данных задач требуется алгоритм, находящий такое паросочетание, которое удовлетворяло бы всем предпочтениям, так как стандартные алгоритмы поиска дают неудовлетворительные результаты.

Уже в 1976 Дональд Кнут написал статью о структуре, помогающей в решении подобных задач. В его работе использовались не только связи между объектами (в качестве рёбер), но и предпочтения одних объектов относительно других. Решением данной задачи было названо устойчивое паросочетание (stable matching), которое учитывает пожелания всех объектов и не может быть улучшено относительно удовлетворения предпочтений. В конце своей статьи автор составил список нерешённых проблем, связанных с нахождением устойчивых паросочетаний. Часть этих проблем на сегодняшний день успешно решена, но некоторые проблемы всё так же остаются открытыми.

Целью работы является изучение существующих нерешенных задач об устойчивых паросочетаниях и разработка эффективных алгоритмов решения данных задач. В данной работе рассматривается решение задачи поиска устойчивого паросочетания, оптимальное по времени, а также применение данного алгоритма в практических целях. В работе подробно описаны и программно реализованы все алгоритмы для решения данной задачи.

1. ТЕОРИЯ ПАРОСОЧЕТАНИЙ

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 [6].

Пусть n – это параметр, определяющий размерность задачи (к примеру, в задаче коммивояжера, в роли n выступает число городов). Тогда через $f(n)$ обозначим количество элементарных шагов в алгоритме (временн**а**я сложность алгоритма). Количество параметров может быть различным и варьироваться от типа задачи.

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 (остове) алгоритм полного перебора имеет временную сложность $f(n)=(n^{n-2})$ (по теореме Кэли), алгоритм Прима – $f(n)=\frac{n^2}{2}$, а алгоритм Краскала с использованием системы непересекающихся множеств даст $f(n)=m\cdot\ln(n)$, где m – количество рёбер в исходном графе.

Для оценки временн**о**й 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 $T(n) \leq c*f(n)$.

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*.

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.

Полиномиальным алгоритмом (или алгоритмом полиномиальной временной сложности) называется алгоритм, у которого временная сложность равна

$O(p(n))$, где $p(n)$ – некоторая полиномиальная функция, а *n* – входная длина (размерность задачи).

Приведём примеры алгоритмов для решения некоторых задач, чтобы проиллюстрировать понятие временной сложности.

НОД (наибольший общий делитель) – наибольшее число, которое является делителем одновременно и *a*, и *b* (*a* и *b* – числа, подающиеся на входе). С использованием простого перебора можно находить НОД за *O(min(a,b))*, однако при больших входных данных даже линейный алгоритм даёт неприемлемый результат. **Алгоритм Евклида нахождения НОД** позволяет находить НОД за время, равное *O(log(min(a,b)))*. Данный алгоритм является рекурсивным. Время работы алгоритма оценивается **теоремой Ламе**, которая устанавливает удивительную связь алгоритма Евклида и последовательности Фибоначчи, согласно которой последовательные числа Фибоначчи — наихудшие входные данные для алгоритма Евклида.

Учитывая, что числа Фибоначчи растут экспоненциально (как константа в степени n), получаем, что алгоритм Евклида выполняется за *O(log(min(a,b)))* операций умножения.

В качестве второго примера приведём уже упомянутый выше **алгоритм Прима для поиска остова наименьшего веса**. Сам алгоритм имеет очень простой вид. Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины (или, что то же самое, *n-1* ребро). В итоге будет построен остов, являющийся минимальным.

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.

Самой лучшей временной сложностью полиномиального алгоритма, конечно, является линейная функция вида $O(n)$.При такой сложности время исполнения алгоритма растет линейно с увеличением размерности задачи.

В этой работе рассматриваются графы, значит, размерность задачи будет определяться числом вершин и числом ребер графа. Обозначим их соответственно за num , а их сумму за k . Посмотрим, как будет увеличиваться время решения задачи для различных алгоритмов при увеличении k . Данные приведены в таблице 1.1.

Таблица 1.1. Сравнение нескольких полиномиальных и экспоненциальных функций временной сложности

Функция временной сложности	Размер k					
	10	20	30	40	50	60
n	0,00001 сек.	0,00002сек.	0,00003сек.	0,00004сек.	0,00005сек.	0,00006сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 мин.	5,2 мин.	13,0 мин.
2^n	0,001 сек.	1,0 сек.	17,9 мин.	12,7 дней.	35,7 лет.	366 столетий
3^n	0,059 сек.	58 мин.	6,5 лет	3855 столетий	2×10^8 столетий	$1{,}3 \times 10^{13}$ столетий

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(nondeterministicpolynomial) – 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.

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*-полной задач:

Задача P_1 *преобразуется* в задачу P_2, если любой частный случай задачи P_1можно преобразовать за полиномиальное время в некоторый частный случай задачи P_2, так что решение задачи P_1можно получить за полиномиальное время из решения этого частного случая задачи P_2 (рис. 1.1).

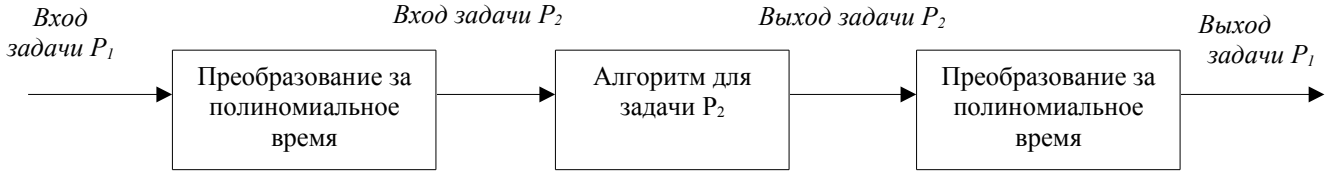

Рис. 1.1. Схематическая диаграмма, показывающая преобразуемость P_1 в P_2

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*.

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

1. Никакую *NP*-полную задачу нельзя решить никаким известным полиномиальным алгоритмом.
2. 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.

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.

Исследование *NP*-полных задач привело к тому, что варианты распознавания фактически *всех разумных* комбинаторных задач оптимизации лежат в *NP*. Приведем список наиболее известных.

1. *Задача выполнимости*. Для данных m дизъюнктов $C_1,\ldots,C_m$, содержащих переменные $x_1,\ldots,x_n$, требуется определить, выполнима ли формула $C_1 \cdot C_2 \cdot \ldots \cdot C_m$.

2. *Задача «независимое множество»*. Имеется граф, есть ли в нем kпопарно несмежных вершин.

3. *Задача «вершинное покрытие»*. Имеется ли в графе вершинное покрытие из k вершин.

4. *Задача гамильтоновости*. Является ли граф гамильтоновым.

5. *Задача о максимальной клике*. Требуется выяснить, существует ли в графе клика размера k [12].

Каков может быть подход к решению *NP-полных* и *NP-трудных* задач? До тех пор пока $P \neq NP$ 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.

1.3. Теория графов и паросочетаний

Задача, решаемая в этой работе, опирается на теорию графов, поэтому напомним некоторые определения из этой теории.

Простым **графом** называется упорядоченная пара $G=\langle V, E\rangle$, где

V –непустое конечное множество (элементы V – вершины графа);

E – конечное множество неупорядоченных пар различных элементов V (элементы E – ребра графа).

Петля – ребро, инцидентное одной и той же вершине.

Кратные ребра – несколько ребер, инцидентных одной и той же паре вершин.

В этой работе будут рассматриваться только простые графы, без петель и кратных ребер.

Пусть, нам дан граф $G=\langle V,E\rangle$, произвольной структуры, не имеющий петель и кратных ребер.

Полный граф – это граф $G=\langle V,E\rangle$, в котором для всех $u\neq v\in V,\{u,v\}\in E$, т.е. граф G полный, если в нем все возможные пары вершин соединены ребрами. Полный граф с n вершинами будет обозначаться символом K_n . K_3 является, конечно, треугольником [11].

Двудольный граф – это граф $G=\langle V,E\rangle$, вершины которого можно разбить на 2 множества $u\neq v\subset V$, так, что в каждом множестве никакие две вершины не соединены друг с другом ребром.

Паросочетанием графа G будем называть множество попарно несмежных рёбер графа. Если паросочетание M не содержится ни в каком другом паросочетании графа *G*, то M называется **максимальным** паросочетанием графа G . **Наибольшее паросочетание** – это паросочетание максимального размера для данного графа.

Примеры паросочетаний показаны на рисунках 1.2 –1.4

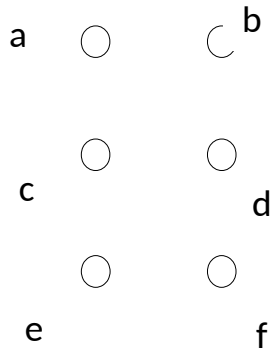

Рис. 1.2. Граф G

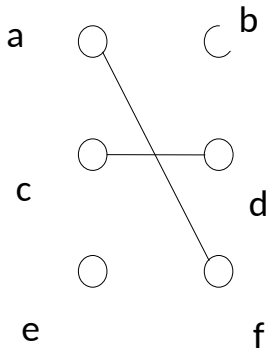

Рис. 1.3. Максимальное паросочетание графа G

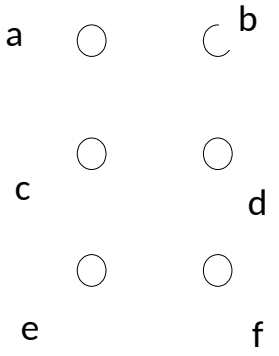

Рис. 1.4. Наибольшее паросочетание графа G

Предпочтения – частный вид бинарных отношений, относительно которых определяется значимость какого-либо элемента множества вершин двудольного графа по сравнению с другими его элементами. Предпочтение удовлетворяет следующим условиям: связанность, транзитивность, ацикличность.

Блокирующая пара – это пара, которая нарушает устойчивость паросочетания и отвечает следующим условиям:

1. $\mu(m)\neq w\,u\,\mu(w)\neq m$

Данное условие означает, что какой-либо мужчина отвергает предписанную ему девушку, и какая-либо девушка отвергает предписанного ей мужчину.

2. $w \succ_m \mu(m) u\, m \succ_w \mu(w)$

Данное условие означает, что для какого-либо мужчины существует девушка, которая является лучше той, которая ему предписана, и для этой девушки данный мужчина также является лучше того, который ей предписан.

Устойчивое паросочетание – паросочетание, в котором нет блокирующих пар, то есть не может возникнуть ситуация, которая не хуже и не лучше для всех элементов паросочетания.

Граф, у которого каждому ребру задано направление, называется **ориентированным графом**. В этом случае каждое ребро называется дугой.

Дополнением G^c графа $G=\langle V, E\rangle$ будем называть граф составленный из того же множества вершин V, в котором ребро между вершинами $u_i u u_j (i \neq j)$ будет проведено тогда и только тогда, когда его не будет в графе G.

Порожденным подграфом будем называть подграф, порождённый множеством рёбер исходного графа. Порожденный подграф содержит не обязательно все вершины графа, но эти вершины соединены такими же ребрами, как в графе.

1.4. Выводы по разделу

В данной главе были рассмотрены классы задач и примеры алгоритмов решения данных задач. Был произведён сравнительный анализ времени работы алгоритмов. В качестве примеров рассмотрены алгоритмы Евклида и Прима, показано их время работы и преимущество использования.

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

Класс NP – класс задач, которые можно решить за полиномиальное время недетерминированными алгоритмами.

Задачи класса *P* могут быть решены с помощью полиномиальных алгоритмов. *NP*-полные же задачи не имеют эффективных алгоритмов решения в общем случае, а потому представляют исследовательский интерес.

Объяснены основные понятия теории графов, которые будут использоваться далее в работе.

Нахождение эффективных структур и алгоритмов для решения *NP*-полных задач позволит расширить область решаемых на практике задач. Одной из таких структур является хордальный граф, рассмотренный в конце данной главы. Благодаря особому строению данного графа возможно решение некоторых *NP*-полных задач на хордальном графе за полиномиальное время.

2. ИСПОЛЬЗОВАНИЕ МОДЕЛИ УСТОЙЧИВЫХ ПАРОСОЧЕТАНИЙ НА ПРАКТИКЕ

2.1. Программа распределения в терапевтической интернатуре США

Первая работа в этой области была посвящена истории развития программы NRMP (National Resident Matching Program). Рот [12] проанализировал историю развития механизмов распределения в данной программе. История программы зачисления начинается с 1945 г., когда впервые были установлены единые правила распределения интернов по больницам. Высокая конкуренция госпиталей за потенциальных интернов привела к тому, что больницы старались сделать предложения заинтересовавшим их студентам как можно раньше. Дело доходило до того, что студенты выбирали распределение в интернатуру за 1,5–2 года до окончания основного курса. Для борьбы с такими «ранними предложениями» ввели ограничение по времени, когда больницы могут предлагать места студентам. Из-за этого ограничения от студентов стали требовать очень быстрого ответа на предложение (вплоть до нескольких часов). Для решения этой проблемы правила распределения неоднократно изменялись. Э. Рот показал, что практики распределения будущих врачей, использовавшиеся до 1952 г., порождали распределения, не лежавшие в ядре соответствующей игры и, соответственно, неустойчивые [12]. Таким образом, было показано, что механизмы, порождающие неустойчивое распределение агентов, не могут долго существовать, поскольку после объявления распределения находятся группы недовольных, желающих изменить распределение. Наконец, в 1952 г. был внедрен механизм, аналогичный механизму Гейла – Шепли. Этот централизованный механизм успешно использовался до 1995 г. Версия механизма 1952 г. работала в интересах больниц, т.е. из всех возможных устойчивых паросочетаний выбирала наилучшее для больниц. Э. Рот показал, что в механизме NRMP-1952 интернам всегда выгодно честно сообщать свой первый госпиталь в списке предпочтений. Кроме того, невыгодно искажать свои предпочтения больницам, имеющим одну вакансию. В остальном как больницы, так и будущие доктора имели возможности манипулирования, т.е. получения лучшего распределения путем сообщения в системе искаженных предпочтений. Позднее, в 1995–1999 гг. Элвин Рот и его коллега Эллиотт Перансон приняли участие в реформировании механизма распределения интернов [9]. Во-первых, была выбрана версия алгоритма отложенного принятия с предлагающими интернами, что сделало механизм ориентированным на интересы будущих докторов, а не больниц, как это было раньше. Во-вторых, были учтены новые проблемы, возникшие в системе распределения интернов с 1950-х годов. В 50-х годах прошлого века, когда создавался первоначальный механизм, подавляющее большинство выпускников

медицинских вузов составляли мужчины. Однако к 1990-м годам женщины составляли значительную долю выпускников медицинских вузов, и это создало неожиданную проблему – среди будущих докторов нередко стали образовываться семейные пары. Молодожены, естественно, хотели получить не просто хорошее место в больнице, но два хороших места в больницах одного города. Это приводило к тому, что семейные пары пытались договариваться с больницами в обход централизованной программы. Помимо появления семейных пар, в системе распределения возникли и другие дополнительные ограничения, связанные с организацией интернатуры в различных больницах. Например, некоторые студенты получали распределение сразу на две программы, первого и второго года обучения, которые были связаны между собой. Ротом и Перансоном была предложена модификация алгоритма [9], которая позволила учесть предпочтения семейных пар с наложением некоторых ограничений на выбор супругов. Их алгоритм был основан на последовательной процедуре Рота и Ванде Вата случайного устранения нестабильностей [17]. Однако первоначально процедура подразумевала случайный выбор объектов. Для выбора способа устранения случайности в процедуре было проведено специальное компьютерное моделирование на имевшихся исторических данных о предпочтениях колледжей и интернов в NRMP. Было показано, что в общем случае, при произвольных предпочтениях супружеских пар и студентов, претендующих одновременно на две программы, относительно желаемых мест устойчивого распределения может не существовать. Однако в случае программы NRMP для всех имевшихся в распоряжении исследователей исторических профилей предпочтений стабильное паросоче- тание существовало и могло быть найдено предложенным алгоритмом.

2.2. Распределение по школам

Еще одно приложение теории устойчивых паросочетаний – распределение учащихся по школам. Особенность этой задачи и ее отличие от классической проблемы состоит в том, что школы при приеме учеников, как правило, не проводят специальных вступительных испытаний (особенно в младших классах). Некоторые дети получают приоритет при поступлении в школу на основании социальных льгот, проживания поблизости от школы или обучения в школе братьев и сестер. Таким образом, школы не могут сообщить строго упорядоченный по предпочтительности список потенциальных учеников, а могут лишь указать группы учеников по убыванию приоритетности обучения. Элвин Рот совместно с Атилой Абдулкадироглу (Университет Дьюк), Парагом Патак (Массачусетский технологический институт) и Тайфуном Сонмезом (Бостонский колледж) приняли участие в реформировании системы распределения учащихся в школьных округах Бостона и Нью-Йорка [3; 4]. До 2001 г. в Бостоне использовался следующий механизм зачисления учащихся. Каждый учащийся (совместно с родителями) составлял список школ по убыванию привлекательности. На первом этапе рассматривались только школы, указанные

первыми в списках учеников. Если поставивших школу на первое место оказывалось больше, чем мест, то учитывались социальные льготы учащихся в данной школе. В спорных случаях зачисляемые выбирались случайным образом. Для учеников, которые не получили места на первом этапе, рассматривались их вторые по предпочтительности школы. Однако если школа, указанная учеником на втором месте, была заполнена другими желающими еще на первом этапе, то ученик снова не получал места. Распределение, к которому приводит такой механизм, вообще говоря, неустойчивое. Кроме того, при такой схеме распределения родителям и ученикам чаще всего было невыгодно сообщать свои настоящие предпочтения. Более того, сами руководители Бостонского образовательного округа в специальных брошюрах рекомендовали родителям «думать стратегически» при составлении списка предпочтений. Как выяснили впоследствии Рот и другие исследователи, такая схема приводила к тому, что «хитрые» семьи получали дополнительное преимущество, в то время как семьи, честно сообщавшие свои предпочтения, значительно проигрывали и получали крайне нежелательные для себя школы. Учеными было предложено два варианта нового механизма распределения, каждый из которых гарантировал неманипулируемость со стороны учеников. Первый вариант представлял собой модификацию алгоритма Гейла – Шепли с введением случайной составляющей [4]. Второй вариант – механизм Top Trading Cycles [19]. Второй механизм порождает распределение, которое, вообще говоря, может доминировать по Парето распределение Гейла – Шепли. Однако в конечном итоге руководителями Бостонского образовательного округа был выбран именно алгоритм Гейла – Шепли. Основной причиной послужило то, что порождаемое им паросочетание всегда является устойчивым, а в контексте распределения в государственные школы устойчивость в некотором смысле эквивалентна справедливости и соблюдению установленных законом приоритетов. Действительно, если распределение устойчиво, то никакой ученик не может заявить, что в школе, куда он хотел бы попасть, учится ребенок с меньшим приоритетом. К сожалению, в случае, когда в предпочтениях школ имеются эквивалентные абитуриенты, могут существовать попарно стабильные паросочетания, которые не являются Парето-эффективными и не принадлежат ядру. Рассмотрим следующий простой пример. Пусть есть два ребенка, Билл и Джоана. Биллу больше нравится школа «Ball», а Джоане – школа «Doll». Однако обеим школам безразлично, кого именно принять – Билла или Джоану. Отправим тогда Билла в школу «Doll», а Джоану – в школу «Ball». Формально такое распределение будет стабильным, однако видно, что оно не является эффективным. Более того, оно не принадлежит ядру, поскольку если Билл, Джоана и две школы объединятся в группу, они смогут составить другое распределение так, что ничье положение не ухудшится, а для некоторых (в данном случае для детей) даже улучшится.

2.3. Пересадка почек

Огромное количество людей во всем мире, и в частности, в США, ежегодно нуждаются в пересадке почки. Часть из них получает пересадку от трупного донора, однако число таких органов значительно меньше, чем потребность в пересадке. Поскольку здоровый человек имеет две почки и в принципе способен жить здоровой жизнью с только лишь одной из них, существует возможность пересадки от живого донора. В США пересадка почки разрешена от любого живого донора, который хочет безвозмездно пожертвовать ее нуждающемуся пациенту. Однако часто случается так, что добровольный донор и его нуждающийся в трансплантации родственник/друг несовместимы друг с другом по группе крови или другим факторам. Таким образом, несмотря на готовность донора помочь, пересадка оказывается невозможной. Э. Рот совместно с Тайфуном Сонмезом (Бостонский колледж) и Утку Унвером (Бостонский колледж) [13; 14] построили научный фундамент системы обмена почками между такими несовместимыми парами (донор из первой пары отдает почку реципиенту из второй, а донор из второй пары – реципиенту из первой). Были получены теоретические результаты, показывающие существование неманипулируемого механизма обмена, а затем, совместно с трансплантологами региона Новая Англия (США), была спроектирована система обмена почками [15]. В дальнейшем похожие схемы были внедрены в нескольких регионах США, Канады и Великобритании. Это позволило существенно увеличить число проводимых трансплантаций и значительно сократить время ожидания трансплантации для больных.

Разработаны различные типы механизмов в зависимости от законодательных и медицинских особенностей в различных регионах и странах. В самой простой схеме донор и реципиент могут либо подходить, либо не подходить друг другу, при этом степень совместимости не оценивается. В таком случае исследователям удалось предложить два неманипулируемых механизма, позволяющих строить максимально возможное паросочетание, т.е. схему перекрестной трансплантации. Под неманипулируемостью в данном случае понимается тот факт, что всем пациентам выгодно раскрывать информацию о своих потенциальных донорах (может быть нескольких родных/друзьях, готовых пожертвовать почку). Один из механизмов основан на традиционном для трансплантологии принципе очередей (приоритетов у больных). Второй предложенный механизм – стохастический, основан на принципе эгалитаризма. Механизм выбирает схему трансплантации таким образом, чтобы самая низкая среди всех пациентов вероятность получить почку была бы как можно выше. В общем случае при оценке совместимости рассчитывается вероятность успеха трансплантации. В этом случае эффективный механизм позволяет пациенту «обменивать» своего живого донора на более высокое место в очереди на ожидание трупной трансплантации.

2.4. Распределение интернов-гастроэнтерологов в США

Э. Рот совместно с Муриэлем Нидерле (Стэнфорд) проанализировали очень интересный пример – систему распределения молодых гастроэнтерологов в США [8]. Распределение гастроэнтерологов с середины 1980-х до конца 1990-х годов осуществлялось с помощью централизованного механизма, организованного аналогично рассмотренному нами выше механизму NRMP. Однако в конце 1990-х годов централизованная схема была отменена, поскольку большинство больниц и интернов предпочитали договариваться задолго до официальной централизованной процедуры. В течение последующих нескольких лет никаких единых правил распределения по больницам для молодых гастроэнтерологов не существовало. В 2006 г. при участии исследователей удалось восстановить централизованную схему распределения. Рот и Нидерле выделили три основные проблемы децентрализованного рынка: 1) перегруженность, 2) широту охвата и 3) «безопасность». Первая проблема заключается в том, что в отсутствие централизованного механизма любое предложение позиции больницей производится «вручную», а ответ молодого доктора требует времени. Таким образом, больница может просто за счет дефицита времени не иметь возможности предложить работу всем кандидатам, в которых она заинтересована. Это приводит к тому, что больницы стараются делать предложения как можно раньше. В результате некоторые студенты вынуждены выбирать место прохождения интернатуры за год до выпускного вечера. При этом в каждом следующем году поступление первых предложений от больниц происходило все раньше и раньше, так как больницы хотели успеть сделать предложения раньше других. Более того, больницы старались делать предложения о работе, требующие немедленного ответа. Таким образом, выбор молодых докторов значительно сужался, так как у них не оставалось возможности рассмотреть все возможные варианты.

Вторая выявленная проблема децентрализованных рынков – сужение охвата. Проанализировав данные о распределении докторов до, во время и после использования централизованной процедуры, ученые выяснили, что при использовании централизованной процедуры студенты имели статистически более высокие шансы получить позицию в госпитале, городе и даже штате, отличном от того, где они получили предыдущее образование. Фактически при использовании децентрализованной процедуры рынок постепенно распадался на отдельные осколки по территориальному признаку. Отчасти причиной такой ситуации стало то, что в условиях конкуренции и дефицита времени на интервью больницы имели больше возможностей получить информацию о молодых докторах из своей местности, чем о кандидатах из других регионов. Третья проблема связана с первыми двумя. Каждая больница опасается стратегических действий других игроков и поэтому сама старается всех «перехитрить». Причем эта проблема возникает и в том случае, если участникам предлагается вернуться к централизованной процедуре. Госпиталь опасается, что если он войдет в программу централизованного распределения, то может пострадать от действий

других больниц, по-прежнему делающих предложения до начала централизованного распределения. Наконец, Рот и Нидерле с использованием статистики заработных плат показали, что децентрализация распределения интернов не приводит к росту их заработных плат. Этот результат был особенно важен в связи с исками некоторых выпускников медицинских вузов США к организаторам централизованной процедуры зачисления касательно неконкурентного назначения заработных плат. Таким образом, было показано, что такие иски не имеют под собой достаточного основания. Авторы провели ряд экспериментов, моделирующих ситуацию, предшествовавшую отказу от централизованной процедуры в 1997 г. Было обнаружено, что произвольный отказ участников рынка от участия в централизованном механизме был вызван сочетанием двух факторов. Во-первых, из-за изменений в системе подготовки гастроэнтерологов сильно сократилось число выпускников, желающих получить место в интернатуре по этой специальности. Во-вторых, сами интерны не подозревали о том, что произошло такое значительное сокращение числа кандидатов, в то время как госпитали были прекрасно осведомлены о дефиците выпускников. Таким образом, в 1997 г., получая предложение до начала официальной централизованной процедуры, молодые врачи, как правило, не отказывались от него, так как ожидали высокой конкуренции за места во время централизованного распределения. В экспериментах было показано, что каждый из таких факторов по отдельности, а также многие другие резкие изменения в соотношении спроса и предложения, в размерах рынка и т.п. не приводят к произвольному отказу от централизованной процедуры, поскольку большинство участников не заинтересованы в этом.

2.5. Распределение помощников федеральных судей

Помощниками федеральных судей становятся молодые юристы после окончания вуза. И молодые выпускники, и судьи заинтересованы в таком распределении: молодежь получает возможность набраться опыта и установить контакты, а профессиональные судьи получают помощников в своей профессиональной деятельности. При этом процесс распределения происходит абсолютно нецентрализованно. На данном рынке неоднократно предпринимались попытки установления единой даты начала процесса распределения, однако всякий раз установленные сроки быстро начинают нарушаться. Э. Рот изучал эту проблему совместно с исследователями в области экспериментальной экономики [7], а также вместе с профессиональными учеными-юристами [5]. В результате проведения масштабных опросов как судей, так и молодых выпускников (было проведено два больших раунда опросов, в 1999–2000 и 2004–2007 гг.) было выявлено, что основные проблемы при распределении помощников судей схожи с проблемами других децентрализованных рынков (см., например, выше рынок гастроэнтерологов). С помощью лабораторных экспериментов и компьютерного моделирования в работе [7] оценивалось влияние тех или иных изменений в

процедуре распределения на его эффективность. Однако вывод неутешителен. Основной причиной невозможности избавиться от ускорения сроков поступления предложений и порождаемой этим неэффективности является вера как судей, так и молодых выпускников в то, что будущему клерку выгоднее всегда соглашаться на первое поступившее предложение. Таким образом, переходу к более эффективному распределению мешает уверенность каждого из участников рынка в том, что другие не согласятся на такой переход.

2.6. Использование устойчивых паросочетаний в России

Впервые на русском языке тематика устойчивых паросочетаний была представлена, по-видимому, в работе [1], где изложены на базовом уровне основные результаты, полученные Гейлом, Шепли и Ротом. Как и в других странах, у нас существует немало ситуаций, в которых необходимо распределение участников двух типов без участия денег. Ярким примером является приемная кампания по зачислению в российские государственные вузы. Действительно, постановка проблемы крайне близка к оригинальной модели Гейла и Шепли: абитуриенты формулируют предпочтения на множестве факультетов, на которые они хотели бы попасть, а вузы имеют предпочтения (определяемые результатами сдачи ЕГЭ) относительно абитуриентов. Используемая в настоящее время в России квазицентрализованная система зачисления абитуриентов была проанализирована в статье [2]. Было показано, что при определенных предположениях существующая в настоящий момент децентрализация, а также ограничение на число вузов, в которые разрешена подача документов, приводят к построению нестабильного и неэффективного паросочетания. Количество вузов, в которые может подавать заявление абитуриент, в настоящее время ограничено пятью, при этом абитуриент не обязан информировать вуз о рассматриваемых им других альтернативах. Таким образом, сильные абитуриенты, подавая заявления сразу в несколько вузов, занимают места всех остальных. В то же время число волн зачисления ограничено, поэтому при соблюдении всех имеющихся правил квазицентрализованной процедуры пострадают (недоберут абитуриентов) вузы среднего уровня. Таким образом, существующая система требует манипулирования как со стороны абитуриентов, так и со стороны вузов.

2.7. Выводы по разделу

Задачи, связанные с устойчивыми паросочетаниями, имеют широкое практическое применение. Все эти задачи – от школ и вузов до больниц и трансплатологии – могут быть решены с помощью алгоритмов поиска устойчивых паросочетаний.

В данный момент при решении далеко не всех подобных проблем используются методы теории графов и алгоритмы поиска устойчивых паросочетаний. Развитие этой темы и разработка оптимальных алгоритмов приведёт к существенному продвижению в решении указанных задач.

3. ЗАДАЧИ ОБ УСТОЙЧИВЫХ ПАРОСОЧЕТАНИЯХ НА ДВУДОЛЬНОМ И ПОЛНОМ ГРАФАХ

3.1. Нерешённые задачи

Задачи, поставленные ниже, были поставлены Д. Кнутом в конце его работы об устойчивых паросочетаниях.

Задача 1. Исследовать среднее число смены партнёров женщиной в процессе выполнения основного алгоритма (женщина x меняет жениха всякий раз, когда меняется F[x]). Будет ли это число существенно меньше числа предложений? Кажется естественным предположение о том, что это число имеет порядок n*ln(ln(n)). В самом деле, женщина со случайным списком предпочтений первое предложение принимает всегда, второе предложение принимает с вероятностью 1/2, …, k-е предложение — с вероятностью 1/k. Женщина, получившая k предложений, меняет женихов в среднем $1 + 1/2 + 1/3 + \ldots + 1/k = H_k$ раз. В среднем женщина получает H_n предложений, поэтому среднее количество смен партнёра ею должно быть примерно равно ln(ln(n)).

Задача 2. Верна ли эта гипотеза о нижней оценке E(n): среднее число предложений, сделанных в случае, когда предпочтения женщин фиксированы, а предпочтения мужчин случайны, всегда не меньше чем $(n+1)\cdot H_n - n$?

Задача 3. Пусть при фиксированной матрице предпочтений мужчин предпочтения женщин формируются случайно. Верна ли гипотеза о том, что среднее число предложений достигает своего максимума, когда все мужчины имеют одинаковые списки предпочтений?

Задача 4. Существует ли эффективный способ точного вычисления среднего числа предложений при фиксированной матрице предпочтений женщин? Иначе говоря, можно ли точно определить это значение по структуре матрицы предпочтений женщин? Возможно, оно зависит от некоторых свойств матрицы предпочтений, таких, например, как число несовпадений рейтингов мужчин в списках предпочтений женщин.

Задача 5. Найти матрицы предпочтений n мужчин и n женщин, при которых число устойчивых паросочетаний максимально. Например, при $n = 4$ максимальное из известных значений числа устойчивых паросочетаний равно 10. Эта ситуация возникает при следующих матрицах предпочтений:

A:	a	b	c	d		a:	D	C	B	A
B:	b	a	d	c		b:	C	D	A	B
C:	c	d	a	b		c:	B	A	D	C
D:	d	c	b	a		d:	A	B	C	D

Рис 3.1. Матрицы предпочтений, при которых число устойчивых паросочетаний максимально

В данном примере в устойчивых паросочетаниях партнёршами A, B, C и D являются соответственно:

a	b	c	d
b	a	c	d
a	b	d	c
b	a	d	c
b	d	a	c
c	a	d	b
c	d	a	b
c	d	b	a
d	c	a	b
d	c	b	a

Рис. 3.2. Устойчивые паросочетания

Задача 6. Найти способ описания структуры множества устойчивых паросочетаний при заданных матрицах предпочтений, который позволил бы охарактеризовать решения, не прибегая к их явному нахождению.

Задача 7. При заданных случайным образом матрицах предпочтений найти асимптотическое поведение среднего числа устойчивых паросочетаний

Задача 8. Пусть даны матрицы предпочтений мужчин и женщин. Построить ориентированный граф «разводов». В нём будет n! вершин, каждая из которых соответствует перестановке $(\pi_1, \ldots, \pi_n)$ элементов множества $\{1, 2, \ldots, n\}$ (номеров женщин в паросочетании с фиксированным порядком мужчин).

Пусть $(A_1 a_{\pi 1}, \ldots, A_n a_{\pi n})$ — неустойчивое паросочетание. Предположим, что при некоторых $i \neq j$ имеют место предпочтения $a_{\pi j}\ A_i a_{\pi i}$ и $A_i a_{\pi j}\ A_j$. Тогда граф разводов содержит дугу из вершины $(\pi_1, \ldots, \pi_n)$ в вершину $(\pi'_1, \ldots, \pi'_n)$, где $\pi'_i = \pi_j$, $\pi'_j = \pi_i$, а для всех остальных номеров k имеет место совпадение $\pi'_k = \pi_k$. Вершины, не имеющие исходящих дуг, соответствуют устойчивым паросочетаниям.

Всегда ли существует путь, начинающийся в заданной вершине и приводящий к устойчивому паросочетанию? Если да, то насколько он может быть короток?

Задача 9. Существует ли алгоритм поиска устойчивого паросочетания, где количество операций, даже в наихудшем случае, растёт медленнее, чем n^2? (Мы не учитываем времени ввода матриц предпочтений мужчин и женщин.)

Задача 10. Существует ли интересная взаимосвязь между задачей об устойчивых паросочетаниях и задачей о назначении?

Задача 11. (о трёхдольном сочетании). Допускает ли задача об устойчивых паросочетаниях обобщение на случай трёх множеств (например, мужчины, женщины, собаки)?

Задача 12. (без браков и измен: устойчивость расселения по комнатам). Каждый человек в группе, состоящей из 2n человек, составляет рейтинг оставшихся 2n−1 потенциальных соседей. Найти эффективный (полиномиальный в наихудшем случае) алгоритм поиска устойчивого разбиения 2n человек на n пар. (Устойчивость в данном случае означает, что не найдётся двух людей в разных комнатах, которые оба предпочли бы жить друг с другом.)

На текущий момент часть задач, поставленных Кнутом, была решена. Так, для задачи 5 были найдены матрицы предпочтений, а в качестве решения задачи 11 построено обобщение на трёхдольный случай.

Высокий интерес в связи с практической значимостью имеют задачи 9 и 12 – о поиске оптимальных алгоритмов для решения упомянутых выше задач.

3.2. Алгоритм поиска устойчивого паросочетания

Для решения задачи поиска устойчивого праосочетания Гейл и Шепли разработали алгоритм, имеющий время работы $O(n^2)$. В настоящее время именно этот алгоритм используется при поиске устойчивых паросочетаний.

Опишем алгоритм, позволяющий найти одно устойчивое паросочетание. В дальнейшем будем называть его основным алгоритмом. Тем самым мы получим конструктивное доказательство существования по крайней мере одного решения задачи об устойчивых паросочетаниях. Инициатива выбора принадлежит мужчинам: они по очереди, один за другим, делают предложения женщинам. Женщины либо принимают предложения, либо отвергают их в соответствии со своими предпочтениями. Как мы увидим, это всегда приводит к устойчивому решению. В алгоритме используются 3 переменные k, X, x и две константы n и Ω:

n — число мужчин (равное числу женщин);
k — число сформированных (пробных) пар;
X — мужчина, делающий предложение;
x — женщина, которой делается предложение;
Ω — (очень нежелательный) воображаемый мужчина.

k ←0; {все женщины (временно) помолвлены с Ω}
while k < n **do**
begin X ←(k +1)-й мужчина;
while X ≠ Ω **do**
begin x ← лучшая кандидатура из текущего списка предпочтений мужчины X;

```
            if x предпочитает мужчину X своему жениху then
                begin помолвка X и x;
                X ←предыдущий жених женщины x
            end;
            if X ≠ Ω then
                вычёркиваем x из текущего списка предпочтений мужчины X;
            end;
    k ←k +1
end
```

Доказательство корректности алгоритма

Математически строгое исследование алгоритмов началось не так давно. Ранее алгоритмы разрабатывались исходя из интуитивных представлений, а их работоспособность проверялась с помощью написанных на их основе компьютерных программ. Для того чтобы доказать корректность алгоритма и оценить его эффективность, необходимо выяснить смысл всех переменных и их взаимосвязь на каждом шаге алгоритма. Доказательство проводится методом математической индукции по номеру итерации. Сначала отметим следующие свойства, справедливые для любой итерации алгоритма.

Предложение 1. Если женщина a вычёркивается из списка предпочтений мужчины A, то ни одно устойчивое паросочетание не будет содержать пары Aa.

Доказательство. При выполнении операции «вычеркнуть женщину x из списка предпочтений мужчины X» в ходе реализации алгоритма при значениях переменных x = a и X = A возможны две ситуации:

- мужчина A сделал предложение женщине a, но та предпочла ему своего жениха B;

- женщина a, помолвленная с A, оставляет его, получив более привлекательное предложение от B.

В обоих случаях B более предпочтителен для a, чем A, при этом для B кандидатура a предпочтительнее всех остальных женщин в его текущем списке. Докажем предложение 1 от противного. Рассмотрим сначала случай, когда текущий список предпочтений B совпадает с начальным. Предположим, что устойчивое паросочетание содержит пару Aa. Тогда B женат на ком-то, кто для него лучше, чем a. Однако в рейтинге мужчины B кандидатура a стоит на первом месте. Противоречие. Предполагая, что предложение 1 имеет место на всех предыдущих итерациях алгоритма, по индукции обобщаем этот результат на произвольный случай.

Предложение 2. Пусть в исходном списке предпочтений мужчины A кандидатура a предшествовала его нынешней невесте. Тогда a отвергла A ради кого-то другого.

Предложение 3. Никакие две женщины не могут одновременно быть помолвлены с одним мужчиной, за исключением Ω.

Предложение 4. Никакая женщина никогда не ухудшает свой выбор в ходе алгоритма.

Предложение 5. Список предпочтений любого мужчины никогда не становится пустым.

Доказательство. Предположим противное. В силу предложения 2 это означает, что этого мужчину отвергли все женщины. Но тогда в силу предложений 3 и 4 каждая из женщин имеет своего «бойфренда», отличного от Ω. Таким образом, должно существовать n мужчин, кроме A. Противоречие.

Заметим, что только нарушение предложения 5 могло бы привести к невозможности составить паросочетание с помощью описанного выше алгоритма, а так как предложение 5 доказано, паросочетание найдётся всегда.

Описанный алгоритм конечен, поскольку после каждого очередного выбора невесты либо сокращается список предпочтений одного из мужчин, либо значение переменной k увеличивается на единицу.

Преложение 6. Полученное паросочетание устойчиво.

Доказательство. Пусть A женат не на a, но предпочитает её своей жене. Это означает, что a предпочла ему кого-то другого (предложения 2, 4). На самом деле алгоритм вырабатывает оптимальный вариант для каждого мужчины. Каждый мужчина женат на лучшей из возможных кандидатур: не существует устойчивого паросочетания, в котором его супругой была бы более желанная для него женщина. (Это следует из предложения 1.) Поскольку вырабатываемое алгоритмом решение является оптимальным для всех мужчин, их нумерация не имеет значения, хотя при работе алгоритма используется некоторая их очерёдность. В то же время полученный результат является наихудшим для всех женщин. (То есть партнёр, назначенный женщине алгоритмом, нравится ей не больше её партнёра в любом устойчивом паросочетании.)

Доказательство. Пусть в решении, найденном алгоритмом, содержится пара Aa (это означает в силу оптимальности найденного решения для всех мужчин, что a для A — наилучшая из всех реальных кандидатур). Предположим, что в другом устойчивом паросочетании содержатся пары Ba и Ab, причём a предпочитает A мужчине B. Тогда A должен предпочитать b кандидатуре a, что противоречит предположению об оптимальности a для A. С помощью того же алгоритма можно получить решение, наилучшее для всех женщин (и наихудшее для мужчин), — в этом случае предложения «руки и сердца» должны делать женщины.

Тот факт, что найденное решение является «наилучшим для мужчин и наихудшим для женщин одновременно», представляет собой частный случай более общего результата.

Теорема 1. Если некоторое устойчивое паросочетание содержит пару Aa, а некоторое другое устойчивое паросочетание — пары Ab и Ba, то возможна одна из двух ситуаций: bAa и AaB или aAb и BaA. (Иными словами, любое другое устойчивое паросочетание лучше для одного из супругов и хуже для другого.)

Доказательство. По определению устойчивости новые партнёры бывших супругов A и a не могут одновременно оказаться хуже для них обоих. Остаётся показать, что они не могут быть и лучше для A и a одновременно. Пусть $A = X_0$, $a = x_0$, $b = x_1$. Предположим, что bAa, т. е. $x_1X_0x_0$. В первом устойчивом паросочетании X_0 имеет худшую (из двух) партию, следовательно, x_1 выбрала

того, кто ей желаннее, чем X_0. Пусть в первом устойчивом паросочетании x_1 замужем за X_1. Тогда $X_1x_1X_0$. Во втором решении x_1 имеет худшую (из двух) партию, значит, X_1 сделал лучший выбор. Пусть во втором устойчивом паросочетании X_1 женат на x_2. Тогда $x_2X_1x_1$ и так далее. Получаем последовательность:

X_0x_0, X_1x_1, X_2x_2, … в первом устойчивом паросочетании,

X_0x_1, X_1x_1, X_2x_3, … во втором устойчивом паросочетании,

где $x_{k+1}X_kx_k$ и $X_{k+1}x_kX_k$ для всех $k>0$.

Поскольку число мужчин конечно, найдутся такие номера j и k, что $j < k$ и $X_j = X_k$. Пусть j и k — наименьшие из таких номеров. Имеем $x_j = x_k$. Тогда $j = 0$, так как иначе во втором устойчивом паросочетании содержались бы пары $Xk{-}1xk = X_{k-1}x_j$ и $X_{j-1}x_j$. (Но тогда $X_{j-1} = X_{k-1}$, что противоречит минимальности j.) Поэтому второе устойчивое паросочетание содержит $X_{k-1}x_0$. Но $x_0 = a$. Тогда по условию теоремы $X_{k-1} = B$. Учитывая, что $X_kx_kX_{k-1}$, получаем AaB.

Следствие. Для двух произвольных устойчивых паросочетаний I и II имеет место свойство: если для всех мужчин I не хуже, чем II, то для всех женщин I не лучше, чем II.

Анализ алгоритма

Определим число шагов алгоритма. Проанализируем алгоритм с помощью схемы алгоритма на рис. 3.3. Число, написанное рядом со стрелкой, указывает количество выполнений соответствующей операции. Число мужчин, равно как и женщин, есть заданная константа n. Обозначим через N общее число предложений, сделанных «соискателями»; оно зависит от списков предпочтений.

Рис. 3.3. Схема основного алгоритма

3.3. Поиск справедливого устойчивого паросочетания

Рассмотренные алгоритмы вырабатывают решение, наиболее благоприятное для мужчин (или для женщин после соответствующей модификации обозначений). Возникает вопрос о решении, в котором представители обоих полов были бы равноправны. Можно было бы, перечислив все устойчивые паросочетания, сделать выбор из них в соответствии с некоторым критерием «справедливости». Однако при большом числе решений такой подход может оказаться слишком трудоёмким. Вообще говоря, количество устойчивых паросочетаний заранее неизвестно. Поэтому представляется более предпочтительным использование другого подхода. Например, выберем пару Aa случайным образом. Выполним модифицированный алгоритм поиска устойчивого паросочетания, содержащего пару Aa. При существовании такого решения выполним те же действия для задачи размерности n−1 (исключив из исходных данных A и a). В противном случае назначаем невесте a другого жениха и повторяем весь процесс. Можно улучшить алгоритм так, чтобы число обращений к процедуре поиска устойчивого паросочетания не превышало бы n^2. Тогда среднее число шагов алгоритма будет иметь полиномиальный порядок $O(n^4)$, часто даже $O(n^3 \cdot ln(n))$.

С. Селков (S. Selkow) предложил алгоритм, где случайный выбор играет меньшую роль. Он вырабатывает справедливое оптимальное решение в том смысле, что оно минимизирует неудовлетворённость самого несчастного. В паросочетании M определим меру неудовлетворённости конкретного человека как номер позиции партнёра в рейтинге рассматриваемого человека. Обозначим максимальную среди 2n значений меры неудовлетворённости через $U(M)$. Наша цель — минимизировать $U(M)$. На первом шаге алгоритма ищем (с помощью основного алгоритма) как решение, оптимальное для всех мужчин, так и решение, оптимальное для всех женщин. Они дают верхнюю и нижнюю оценки меры неудовлетворённости $U(M)$. Составляем пары из тех, чьи верхние и нижние оценки совпадают. Повторяем процесс для задачи меньшей размерности, пока это возможно. Затем (в редуцированной системе) среди мужчин и женщин с максимальными верхними оценками случайным образом выбираем одного. Без ограничения общности считаем, что выбрана женщина (иначе поменяем местами роли мужчин и женщин). Обозначим её через a. Пусть A — худший вариант для a из всех оставшихся мужчин. Заметим, что (в силу оптимальности решения по Парето) a представляет собой наилучшую кандидатуру для A (среди всех оставшихся женщин). Исключаем a из списка предпочтений A и получаем новое решение, оптимальное для мужчин. При этом увеличивается значение нижней оценки меры неудовлетворённости мужчины A и ещё по крайней мере одного мужчины. Одновременно снижается верхняя оценка меры неудовлетворённости женщины a и по крайней мере ещё одной женщины. Очевидно, что, повторяя этот процесс, мы получим устойчивое паросочетание.

3.4. Поиск всех устойчивых паросочетаний

В этом разделе мы опишем рекурсивную процедуру «toutm», зависящую от параметра j, пробегающего значения от 0 до n. В начале работы процедуры (H[1], H[2], …, H[n]) — некоторое устойчивое паросочетание. По завершении работы алгоритма эти переменные H[k] примут своё первоначальное значение. Результатом будет печать всех устойчивых паросочетаний (H′[1], H′[2], …, H′[n]) таких, что

H′[k] = H[k] для $1 \le k \le j$,

H′[k] ≥ H[k] для $j < k \le n$.

(Никто из мужчин не улучшает свой выбор, причём мужчины с номерами от 1 до j не меняют своих партнёрш.)

Для получения всех устойчивых паросочетаний мы сначала выполняем основной алгоритм, но в момент вывода решения, оптимального для мужчин, запускаем рекурсивную процедуру toutm(0). Процедура toutm имеет следующий вид.

```
procedure toutm(j: integer):
x, X, y, t: integer;
SH, SF, SQ: array[1:n] of integer;
if j = n then печать решения (HC[1, H[1]], ..., HC[n, H[n]])
else begin for t ← 1 to n do
   begin SH[t] ← H[t];
   SF[t] ← F[t];
   SQ[t] ← Q[t]
   end;
continue:  toutm(j + 1);
change:    X ← j + 1;
           y ← HC[X, H[X]]; {запрещаем X жениться на этой даме}
propose:   H[x] ← H[x] + 1;
           if H[x] > n then goto finish;
           x ← HC[X, H[X]];
           if P[x, X] > Q[x] then
              begin t ← F[x];
              if t ⩽ j then goto finish;
              F[x] ← X;
              Q[x] ← P[x, X];
              if x = y then goto continue;
              X ← t
              end;
           goto propose;
finish:    for t ← 1 to n do
              begin H[t] ← SH[t];
              F[t] ← SF[t];
              Q[t] ← SQ[t]
              end
   end
```

Для этого алгоритма корректность доказать труднее, чем для основного, хотя здесь применим тот же подход. Будем считать, что по предположению индукции

вызов процедуры toutm(j +1) корректен. Сначала покажем, что к моменту выполнения операции H[X]←H[X]+1 все устойчивые паросочетания, где партнёршей X является H[X], уже получены. (Тем самым ни одно устойчивое паросочетание не будет пропущено.) Затем убедимся в том, что все полученные паросочетания устойчивы. Действительно, если некоторый мужчина A предпочитает женщину a своей текущей партнёрше, то и для неё кто-то предпочтительнее A, поскольку в процессе выполнения алгоритма женщины улучшают свой выбор.

3.5. Алгоритм поиска устойчивого паросочетания на полном графе

Проблема распределения по комнатам по существу, является версией задачи устойчивого паросочетания, включающей только один набор людей (одну долю).

Каждый человек в наборе мощности n (n - чётно), расставляет n - 1 других в порядке предпочтения. Задача состоит в том, чтобы найти устойчивое паросочетание, т.е. разбиение множества на n/2 пар соседей по комнате, так что не должно быть двух человек, которые не являются соседями по комнате, но предпочитают друг друга своим фактическим соседям.

Гейл и Шепли уже продемонстрировали, что, в отличие от устойчивого паросочетания на двудольном графе, в этом случае возможно отсутствие стабильного паросочетания. Они привели пример размера 4, описанный с помощью списка предпочтений ниже, в котором кто-либо, связанный с человеком 4, будет вызывать нестабильность.

A: B C D
B: C A D
C: A B D
D: что угодно

Рис. 3.4. Матрица предпочтений, при которой возникает нестабильность

Возможно, что проблема существования устойчивого решения NP-полна.

Представим алгоритм решения данной задачи.

Алгоритм определяет, существует ли решение задачи и, если существует, находит его. Он определяет подходящие структуры данных для манипуляции со списками предпочтения и перестановками.

Алгоритм содержит 2 этапа. На первом этапе участники делают предложение друг другу, примерно так же, как это делается в основном алгоритме. Участники предлагают создать пару последовательно каждому лицу из своего списка предпочтения, переходя к следующему лицу в списке в случае отказа создать пару. Участник отказывает создать пару, если он уже имеет предложение от лица, которое он предпочитает больше. На этом этапе один участник может быть отвергнут всеми остальными участниками, что свидетельствует об отсутствии стабильного решения. В противном случае этап 1 завершается тем, что каждый участник имеет предложение от какого-либо участника. Эта ситуация может быть представлена как набор S упорядоченных пар вида (p,q), где q имеет предложение

от p, и мы говорим, что q является текущим фаворитом p. Если этот набор является решением, то есть для любого (q,p) из S (p,q) тоже содержится в S, алгоритм завершается и это решение стабильно.

В противном случае переходим к этапу 2, на котором набор S последовательно изменяется с помощью так называемых перестановок. Пусть (p,q) принадлежит S, но (q,p) не принадлежит. Для каждого такого p определяем его текущего второго фаворита, являющегося следующим участником в списке предпочтений участника p после q, который может принять предложение p. Перестановка в S — это последовательность (p_0,q_0), (p_1,q_1),. . ., (p_{k-1},q_{k-1}), такая, что (p_i,q_i) входит в S для любого i, и q_{i+1} является вторым фаворитом участника p_i (здесь i + 1 берётся по модулю k). Если такая перестановка (p_0,q_0), . . . , (p_{2k},q_{2k}) имеет нечетную длину, мы нашли так называемый нечетный набор, который показывает, что стабильного соответствия нет. В противном случае заменяем пары (p_i,q_i) на (p_i,q_{i+1}) (где i + 1 берётся по модулю k).

Этап 2 алгоритма можно представить следующим образом:

```
S = output of Phase 1;
   while (true) {
      identify a rotation r in S;
      if (r is an odd party)
          return null; // нет устойчивых паросочетаний
      else
          apply r to S;
      if (S is a matching)
          return S; // гарантированно устойчивое
   }
```

Пример:

Задан список предпочтений для шести участников p1, p2, p3, p4, p5, p6.

p_1 : p_3 p_4 p_2 p_6 p_5
p_2 : p_6 p_5 p_4 p_1 p_3
p_3 : p_2 p_4 p_5 p_1 p_6
p_4 : p_5 p_2 p_3 p_6 p_1
p_5 : p_3 p_1 p_2 p_4 p_6
p_6 : p_5 p_1 p_3 p_4 p_2

Возможное выполнение этапа состоит из следующих предложений и отказов, где → представляет предложение и × представляет отказ.

$p_1 \rightarrow p_3$
$p_2 \rightarrow p_6$
$p_3 \rightarrow p_2$
$p_4 \rightarrow p_5$
$p_5 \rightarrow p_3$; $p_3 \times p_1$
$p_1 \rightarrow p_4$
$p_6 \rightarrow p_5$; $p_5 \times p_6$
$p_6 \rightarrow p_1$

Таким образом, этап 1 завершается набором $S = \{(p_1,p_4), (p_2,p_6), (p_3,p_2), (p_4,p_5), (p_5,p_3), (p_6,p_1)\}$.

На этапе 2 обнаруживается перестановка $r_1 = (p_1,p_4), (p_3,p_2)$. Это вытекает из того, что p_2 является вторым фаворитом участника p_1, а p_4 является вторым фаворитом участника p_3. Используем r_1 для получения нового набора $S = \{(p_1,p_2), (p_2,p_6), (p_3,p_4), (p_4,p_5), (p_5,p_3), (p_6,p_1)\}$. Теперь обнаруживается перестановка $r_2 = (p_1,p_2), (p_2,p_6), (p_4,p_5)$, и она дает набор $S = \{(p_1,p_6), (p_2,p_5), (p_3,p_4), (p_4,p_2), (p_5,p_3), (p_6,p_1)\}$. Наконец, перестановка $r_3 = (p_2,p_5), (p_3,p_4)$ дает $S = \{(p_1,p_6), (p_2,p_4), (p_3,p_5), (p_4,p_2), (p_5,p_3), (p_6,p_1)\}$. Последнее соответствие является стабильным решением.

3.6. Выводы по разделу

В данной главе рассмотрены основные нерешенные задачи, связанные с устойчивыми паросочетаниями. Проведен анализ проблем и оценка возможных решений.

В качестве решения предложенных задач разаботаны алгоритмы поиска, оптимальные по времени и по памяти. В первую очередь – основной алгоритм поиска устойчивого праосочетания, который был реализован также на языке C++ и дополнен алгоритмами поиска справедливых устойчивых паросочетаний и поиска всех паросочетаний.

Также решена задача создания полиномиального по времени алгоритма поиска устойчивого паросочетания на полном графе. Был разработан алгоритм с временной асимптотикой $O(n^2)$, что сопоставимо с временем работы основного алгоритма.

ЗАКЛЮЧЕНИЕ

В работе изучались устойчивые паросочетания и задачи, связанные с ними. Хорошо исследованной и решённой считается задача поиска максимального паросочетания. Однако, на практике возможны задачи с матрицей предпочтений, что усложняет поиск оптимального паросочетания. Среди применений данной задачи стоит выделить распределение органов доноров по больницам, поиск соответствия пар молодоженов. В этом случае поиск ответа не так тривиален и требует больших временных затрат.

Были рассмотрены нерешённые задачи об устойчивых паросочетаниях. Одна из них заключалась в поиске оптимального алгоритма нахождения устойчивых паросочетаний на полном графе. В работе над данной задачей было достигнуто уменьшение временных затрат на поиск устойчивого паросочетания по сравнению с ранее существующим алгоритмом и создание алгоритма, имеющего такую же временную асимптготику, что и основной алгоритм поиска устойчивых паросочетаний на двудольном графе $O(n^2)$.

Решение данной задачи реализовано программно на языке *C++*. Кроме того, также изучены и реализованы основные алгоритмы для работы с устойчивыми паросочетаниями:

- нахождение устойчивого паросочетания на двудольном графе
- нахождение уникального устойчивого паросочетания на двудольном графе
- нахождение всех устойчивых паросочетаний на двудольном графе
- проверка существования устойчивого паросочетания на полном графе
- нахождение устойчивого паросочетания на полном графе.

Реализованная программа не имеет графического интерфейса, что не позволяет использовать программу в качестве отдельного программного продукта. Также отсутствует графическое представление решаемой задачи и используемых графов. После доработки программы в указанных направлениях возможно использование программного комплекса для решения широкого спектра задач об устойчивых паросочетаниях.

Программа представляет собой библиотеку с функциями, позволяющими находить устойчивые паросочетания на двудольном и полном графах. Данная реализация может быть использована при разработке программного обеспечения для медицинских и образовательных учреждений. Например, для поиска удачного соответствия донорских органов, предпочтительных учебных заведений и подобных задач, встречающихся на практике.

Часть задач, касающихся устойчивых паросочетаний, до сих пор остаётся нерешённой. Таким образом, существует потенциал развития данной темы в будущем.

На основе написанной программной библиотеки можно создать программный продукт с графическим интерфейсом, готовый к использованию

рядовыми пользователями для решения задач, связанных с устойчивыми паросочетаниями.

Также данную библиотеку можно усовершенствовать, добавив стандартные алгоритмы на графах и собрав динамическую библиотеку для кроссплатформенного использования.

ЛИТЕРАТУРА

1. Abdulkadiroglu A., Pathak P.A., Roth A.E. Strategy-proofness versus Efficiency in Matching with Indifferences: Redesigning the NYC High School Match // American Economic Review. 2009. № 99 (5). P. 1954–1978.
2. Abdulkadiroglu A., Pathak P.A., Roth A.E., Sonmez T. The Boston Public School Match // American Economic Review. Papers and Proceedings. 2005. № 95(2). P. 368–371.
3. Aleskerov F., Bouyssou D., Monjardet B., Utility Maximization, Choice and Preference, Springer Verlag, Berlin, 2007
4. Aleskerov F., “Simple and Simplest Semiorders”, Dokl. Math., 387:2 (2002), P. 175–177
5. Avery C., Jolls C., Posner R.A, Roth A.E. The New Market for Federal Judicial Law Clerks // University of Chicago Law Review. 2007. № 74. P. 447–486.
6. Diestel R. Graph Theory - Springer, 2005. P. 410.
7. Erdil A., Ergin H., “What’s the Matter with Tie-Breaking? Improving Efficiency in School Choice”, Amer. Economic Review, 98:3 (2008), P. 669–689
8. Gale D., Shapley L.S. College Admissions and the Stability of Marriage // The American Mathematical Monthly. 1962. № 69 (1). P. 9–15.
9. Haruvy E., Roth A.E., Unver U.M. The Dynamics of Law Clerk Matching: An Experimental and Computational Investigation of Proposals for Reform of the Market // Journal of Economic Dynamics and Control. 2006. № 30(3). P. 457–486.
10. Luce D., “Semiorders and a Theory of Utility Discrimination”, Econometrica, 24:2 (1956), P. 178–191
11. Niederle M., Roth A.E. The Gastroenterology Fellowship Match: How it Failed, and Why it Could Succeed Once Again // Gastroenterology. 2004. № 127. P. 658–666.
12. Peranson E., Roth A.E. The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design // American Economic Review. 1999. № 89(4). P. 748–780.
13. Peterson W.W., Addressing for random-access storage, IBM J. Res. Develop. 1. 1957, P. 130–146.
14. Pittel B., On likely solutions of a stable marriage problem // The Annals of Applied Probability 2. 1992 P. 358-401.
15. Roth A.E. The College Admissions Problem is not Equivalent to the Marriage Problem // Journal of Economic Theory. 1985. № 36. P. 277–288.
16. Roth A.E. The Economics of Matching: Stability and Incentives // Mathematics of Operations Research. 1982. № 7. P. 617–628.
17. Roth A.E. The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory // Journal of Political Economy. 1984. № 92. P. 991–1016.

18. Roth A.E., Sonmez T., Unver U.M. Kidney Exchange // Quarterly Journal of Economics. 2004. № 119(2). P. 457–488.
19. Roth A.E., Sonmez T., Unver U.M. Pairwise Kidney Exchange // Journal of Economic Theory. 2005. № 125(2). P. 151–188.
20. Roth A.E., Sonmez T., Unver U.M. A Kidney Exchange Clearinghouse in New England // American Economic Review. Papers and Proceedings. 2005. № 95(2). P. 376–380.
21. Roth A.E., Sotomayor M.A.O. Two-sided Matching: A Study in Game-Theoretic Modeling and Analysis. Cambridge University Press, 1990.
22. Roth A.E., Vande Vate J.H. Random Paths to Stability in Two- sided Matching // Econometrica. 1990. № 58(6). P. 1475–1480.
23. Shapley L.S. A Value for n-person Games // Contributions to the Theory of Games. Vol. II / H.W. Kuhn, A.W. Tucker (eds.) Annals of Mathematical Studies. 1953. № 28. P. 307–317. Princeton University Press, 1953.
24. Shapley L.S., Scarf H. On Cores and Indivisibility // Journal of Mathematical Economics. 1974. № 1. P. 23–37.
25. Sedgewick R., Quicksort, Ph.D. Thesis, Stanford University, Computer Science Dept., 1975.
26. Алексеев В.В., Гаврилов Г.П., Сапоженко А.А. (ред.) Теория графов. Покрытия, укладки, турниры. Сборник переводов - М. : Мир, 1974.— 224 с.
27. Алескеров Ф.Т., Хабина Э.Л., Шварц Д.А. Бинарные отношения, графы и коллективные решения. 1-ое изд. М.: Изд. дом ГУ ВШЭ, 2007; 2-ое изд. М.: Физматлит, 2012.
28. Асанов, М.О. Дискретная математика. Графы, матроиды, алгоритмы / М.О. Асанов, В.А. Баранский, В.В. Расин. – СПб.: Изд-во “Лань”, 2010. – 368 с.
29. Асельдеров З.М., Донец Г.А. Представление и восстановление графов - К.:Наукова Думка, 1991, 96 с.
30. Ахо, А. Структуры данных и алгоритмы.– М.: Издательский дом "Вильямс", 2001. – 384 с.
31. Виленкин Н.Я. Комбинаторика. — М.: Наука, 1987.
32. Голумбик, М.Ч. Algorithms Graph Theory and Perfect Graphs. –N.Y.:North Holland, 1980. – 314 с.
33. Гэри, М. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982. – 419 с.
34. Джонсон, Д. The NP-completeness column: an ongoing guide. // Journal of Algorithms. – 1985.– №6.– С. 434–451.
35. Зыков А.А. Основы теории графов. - М.:Наука, 1987, 384 с.
36. Калмыков Г. И. Древесная классификация помеченных графов. - М.: ФИЗМАТЛИТ, 2003. - 192 с.
37. Кисельгоф С.Г. Выбор вузов абитуриентами с квадратичной функцией полезности // Проблемы управления. 2012. № 5. С. 33–40.
38. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; пер. с англ. – М.: Издательский дом “Вильямс”, 2013. – 1328 с.

39. Липский В. Комбинаторика для программистов. — М.: Мир, 1988.
40. Мельников О.И. Теория графов в занимательных задачах. Изд.3, испр. и доп. 2009. 232 с.
41. Мельников О.И. Занимательные задачи по теории графов. - Минск: ТетраСистемс, 2001. - 144 с.
42. Оре О. Теория графов. 2-е изд. М.: Наука, Главная редакция физико-математической литературы, 1980, 336 с
43. Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность. – М.: Мир, 1985. – 512 с.
44. Рейнгольд, Э. Комбинаторные алгоритмы. Теория и практика. – М.: Мир, 1980. – 478 с.
45. Робертс, Ф.С. Дискретные математические модели с приложениями к социальным, биологическим и экологическим задачам. / Пер. с англ. А.М. Раппопорта, С.И.Травкина – М.: Наука, 1986. – 496 с.
46. Сачков В.Н. Комбинаторные методы дискретной математики. — М.: Наука,1977.
47. Седжвик, Р. Фундаментальные алгоритмы на С++. Алгоритмы на графах / Р. Седжвик; пер. с англ. – СПб.: ООО “ДиаСофтЮП”, 2002. – 496 с.
48. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
49. Эвнин, А. Ю. Вокруг теоремы Холла. / 2-е изд., перераб. и доп. – М.: Книжный дом «ЛИБРОКОМ»,2011. – 88 с.
50. Эвнин, А. Ю. Элементы дискретной оптимизации. – Ч.: ЮУрГУ, 2012. – 91 с.

ПРИЛОЖЕНИЕ 1. КОД ПРОГРАММЫ ТИТУЛЬНИК

П1.1. файл stablematching.h

```
#ifndef
galeshapley_h
                #define galeshapley_h


                List cpp_wrapper_galeshapley(umat& prefM, mat&
                rankW);
                bool cpp_wrapper_galeshapley_check_stability(mat&
                uM, mat& uW, umat& proposals, umat& engage-
                ments);


                #endif
```

П1.2. ФАЙЛ STABLEMATCHING.CPP

```
#include
<queue>
                #include <matchingR.h>


                #include "utils.h"
                #include "galeshapley.h"


                //' C++ wrapper for Gale-Shapley Algorithm
                //'
                //' This function provides an R wrapper for the C++ backend. Users should not
                //' call this function directly and instead use
                //' \code{\link{galeShapley.marriageMarket}} or
                //' \code{\link{galeShapley.collegeAdmissions}}.
                //'
                //' @param proposerPref is a matrix with the preference order of the proposing
                //'   side of the market. If there are \code{n} proposers and \code{m} reviewers
                //'   in the market, then this matrix will be of dimension \code{m} by \code{n}.
                //'   The \code{i,j}th element refers to \code{j}'s \code{i}th most favorite
                //'   partner. Preference orders must be complete and specified using C++
                //'   indexing (starting at 0).
                //' @param reviewerUtils is a matrix with cardinal utilities of the courted side
                //'   of the market. If there are \code{n} proposers and \code{m} reviewers, then
                //'   this matrix will be of dimension \code{n} by \code{m}. The \code{i,j}th
                //'   element refers to the payoff that individual \code{j} receives from being
                //'   matched to individual \code{i}.
                //' @return  A list with elements that specify who is matched to whom. Suppose
                //'   there are \code{n} proposers and \code{m} reviewers. The list contains
                //'   the following items:
                //'  \itemize{
                //'    \item{\code{proposals} is a vector of length \code{n} whose \code{i}th
                //'    element contains the number of the reviewer that proposer \code{i} is
                //'    matched to using C++ indexing. Proposers that remain unmatched will be
                //'    listed as being matched to \code{m}.}
                //'    \item{\code{engagements} is a vector of length \code{m} whose \code{j}th
                //'    element contains the number of the proposer that reviewer \code{j} is
                //'    matched to using C++ indexing. Reviwers that remain unmatched will be
                //'    listed as being matched to \code{n}.}
                //'  }
                List cpp_wrapper_galeshapley(const umat& proposerPref, const mat& reviewerUtils) {
```

```
// number of proposers (men)
int M = proposerPref.n_cols;

// number of reviewers (women)
int N = proposerPref.n_rows;

// initialize engagements, proposals
vec engagements(N), proposals(M);

// create an integer queue of bachelors
// the idea of using queues for this problem is borrowed from
// http://rosettacode.org/wiki/Stable_marriage_problem#C.2B.2B
queue<int> bachelors;

// set all proposals to N (aka no proposals)
proposals.fill(N);

// set all engagements to M (aka no engagements)
engagements.fill(M);

// every proposer starts out as a bachelor
for(int iX=M-1; iX >= 0; iX--) {
   bachelors.push(iX);
}


// loop until there are no more proposals to be made
while (!bachelors.empty()) {

   // get the index of the proposer
   int proposer = bachelors.front();

   // get the proposer's preferences: we use a raw pointer to the memory
   // used by the column `proposer` for performance reasons (this is to avoid
   // making a copy of the proposers vector of preferences)
   const uword * proposerPrefcol = proposerPref.colptr(proposer);

   // find the best available match for proposer
   for(int jX=0; jX<N; jX++) {

      // get the index of the reviewer that the proposer is interested in
      // by dereferencing the pointer; increment the pointer after use (not its value)
      const uword wX = *proposerPrefcol++;

      // check if wX is available (`M` means unmatched)
      if(engagements(wX)==M) {
```

```
            // if available, then form a match
            engagements(wX) = proposer;
            proposals(proposer) = wX;

            // go to the next proposer
            break;
          }

          // wX is already matched, let's see if wX can be poached
          if(reviewerUtils(proposer, wX) > reviewerUtils(engagements(wX), wX)) {

            // wX's previous partner becomes unmatched (`N` means unmatched)
            proposals(engagements(wX)) = N;
            bachelors.push(engagements(wX));

            // proposer and wX form a match
            engagements(wX) = proposer;
            proposals(proposer) = wX;

            // go to the next proposer
            break;
          }
        }

        // remove proposer from bachelor queue: proposer will remain unmatched
        bachelors.pop();
      }


      return List::create(
        _["proposals"]   = proposals,
        _["engagements"] = engagements);
}




//' C++ Wrapper to Check Stability of Two-sided Matching
//'
//' This function checks if a given matching is stable for a particular set of
//' preferences. This function provides an R wrapper for the C++ backend. Users
//' should not call this function directly and instead use
//' \code{\link{galeShapley.checkStability}}.
//'
//' @param proposerUtils is a matrix with cardinal utilities of the proposing
```

```
//'   side of the market. If there are \code{n} proposers and \code{m} reviewers,
//'   then this matrix will be of dimension \code{m} by \code{n}. The
//'   \code{i,j}th element refers to the payoff that individual \code{j} receives
//'   from being matched to individual \code{i}.
//' @param reviewerUtils is a matrix with cardinal utilities of the courted side
//'   of the market. If there are \code{n} proposers and \code{m} reviewers, then
//'   this matrix will be of dimension \code{n} by \code{m}. The \code{i,j}th
//'   element refers to the payoff that individual \code{j} receives from being
//'   matched to individual \code{i}.
//' @param proposals is a matrix that contains the number of the reviewer that a
//'   given proposer is matched to: the first row contains the number of the
//'   reviewer that is matched with the first proposer (using C++ indexing), the
//'   second row contains the id of the reviewer that is matched with the second
//'   proposer, etc. The column dimension accommodates proposers with multiple
//'   slots.
//' @param engagements is a matrix that contains the number of the proposer that
//'   a given reviewer is matched to (using C++ indexing). The column dimension
//'   accommodates reviewers with multiple slots.
//' @return true if the matching is stable, false otherwise
bool cpp_wrapper_galeshapley_check_stability(mat proposerUtils, mat reviewerUtils, umat proposals,
umat engagements) {


    // number of workers
    const int M = proposerUtils.n_cols;

    // number of firms
    const int N = proposerUtils.n_rows;

    // number of slots per firm
    const int slotsReviewers = engagements.n_cols;

    // number of slots per worker
    const int slotsProposers = proposals.n_cols;


    // more jobs than workers (add utility from being unmatched to firms' preferences)
    if(N*slotsReviewers>M*slotsProposers) {
        reviewerUtils.insert_rows(M, 1);
        reviewerUtils.row(M).fill(-1e10);
    }
    // more workers than jobs (add utility from being unmatched to workers' preferences)
    if(M*slotsProposers>N*slotsReviewers) {
        proposerUtils.insert_rows(N, 1);
        proposerUtils.row(N).fill(-1e10);
    }
```

```
        // loop over workers
        for(int wX=0; wX<M; wX++) {

            // loop over firms
            for(int fX=0; fX<N; fX++) {

                // loop over multiple "slots" at the same worker
                for(int swX=0;swX<slotsProposers;swX++) {

                    // loop over multiple slots at the same firm
                    for(int sfX=0;sfX<slotsReviewers;sfX++) {

                        // check if wX and fX would rather be matched with each other than with their actual matches
                        if(reviewerUtils(wX, fX) > reviewerUtils(engagements(fX, sfX), fX) && proposerUtils(fX, wX) >
proposerUtils(proposals(wX, swX), wX)) {
                            ::Rf_warning("matching is not stable; worker %d would rather be matched to firm %d and
vice versa.\n", wX, fX);
                            return false;
                        }
                    }
                }
            }
        }
        return true;
    }
```

П1.2. ФАЙЛ ROOMMATE.H

```
#ifndef
roommate_
h
        #define roommate_h


        #include <deque>
        #include "matchingR.h"


        uvec cpp_wrapper_irving(const umat pref);
        bool cpp_wrapper_irving_check_stability(umat pref, umat matchings);


        #endif
```

П1.3. ФАЙЛ ROOMMATE.CPP

```
#include
"roommate.h"


//' Computes a stable roommate matching
//'
//' @param pref is a matrix with the preference order of each individual in the
//'   market. If there are \code{n} individuals, then this matrix will be of
//'   dimension \code{n-1} by \code{n}. The \code{i,j}th element refers to
//'   \code{j}'s \code{i}th most favorite partner. Preference orders must be
//'   specified using C++ indexing (starting at 0). The matrix \code{pref} must
//'   be of dimension \code{n-1} by \code{n}.
//' @return A vector of length \code{n} corresponding to the matchings that were
//'   formed (using C++ indexing). E.g. if the \code{4}th element of this vector
//'   is \code{0} then individual \code{4} was matched with individual \code{1}.
//'   If no stable matching exists, then this function returns a vector of
//'   zeros.

uvec cpp_wrapper (const umat pref) {


  // Number of participants
  uword N = pref.n_cols;


  uvec proposal_to(N);
  uvec proposal_from(N);
  uvec proposed_to(N);


  // All participants begin unmatched having proposals accepted by nobody (=N)...
  proposal_to.fill(N);
  // having accepted proposals from nobody (=N)...
  proposal_from.fill(N);
  // and having proposed to nobody.
  proposed_to.zeros();


  // Empty matchings
  uvec matchings(N);


  bool stable = false;
```

```
while (!stable) {
    // set stable to false later if anyone hasn't proposed / been proposed to
    stable = true;
    for (uword n = 0; n < N; n++) {
        // n proposes to the next best guy if he hasn't proposed to everyone already...
        if (proposed_to(n) >= N-1) { return matchings.zeros(); }


        // or if he has no proposals accepted by anyone.
        if (proposal_to(n) == N) {


            // find the player he is proposing to next
            uword proposee = pref(proposed_to(n), n);


            // proposee's preferences
            const uvec prop_call = pref.col(proposee);


            // find proposee's opinion of the proposer (lower is better)
            uword op = N;
            for (uword i = 0; i < prop_call.n_elem; i++) {
                if (prop_call(i) == n) {
                    op = i;
                    break;
                }
            }


            if (op == N) { stop("Invalid preference matrix: Incomplete preferences."); }


            // find proposee's opinion of his current match
            // lower is better
            // unmmatched is N
            uword op_curr = N;
            for (uword i = 0; i < prop_call.n_elem; i++) {
                if (prop_call(i) == proposal_from(proposee)) {
                    op_curr = i;
                    break;
                }
            }


            // if the next best guy likes him he accepts
            if (op < op_curr) {
```

```
            // make the proposal
            proposal_to(n) = proposee;
            // reject the proposee's original proposer's proposal
            // got it!?
            if (proposal_from(proposee) != N) {
               proposal_to(proposal_from(proposee)) = N;
               // someone has proposed to nobody, we're not stabler yet
               stable = false;
            }
            // record the proposal
            proposal_from(proposee) = n;
         } else {
            // offer was rejected, we're not stable yet
            stable = false;
         }


         // iterate n's proposal forward
         proposed_to(n)++;
      }
   }
}


// Generate tables, initially of length N
std::vector< std::deque<uword> > table(N, std::deque<uword>(N-1));
for (uword n = 0; n < N; ++n) {
   for (uword i=0; i<N-1; i++) {
      // fill in the table with preferences
      table[n][i] = pref(i, n);
   }
}


// Delete entries we eliminated in round 1
for (uword n = 0; n < N; n++) {
   for (int i = table[n].size()-1; i>= 0; i--) {
      if (table[n][i] == proposal_from(n)) {
         break;
      } else {
         if (table[n].size() == 0) { return matchings.zeros(); }
         // find and erase from the table
         bool erased = false;
         for (uword j = 0; j < table[table[n].back()].size(); j++) {
            if (table[table[n].back()][j] == n) {
```

```
                table[table[n].back()].erase(table[table[n].back()].begin() + j);
                erased = true;
                break;
            }
        }
        if (!erased) { return matchings.zeros(); }
        table[n].pop_back();
    }
  }
}


stable = false;
while(!stable) {
  stable = true;
  for (uword n = 0; n < N; n++) {
    if (table[n].size() > 1) {
      stable = false;
      std::vector<uword> x;
      std::vector<uword> index;


      uword new_index = n;
      // Unassigned for now, so assign to the maximum value
      uword rot_tail = static_cast<uword>(-1);


      while (rot_tail == (uword) (index.end() - index.begin() - 1)) {
        int new_x = table[new_index][1];
        new_index = table[new_x].back();


        // Check for a rotation
        rot_tail = find(index.begin(), index.end(), new_index) - index.begin();


        x.push_back(new_x);
        index.push_back(new_index);
      }


      // Delete the rotation
      for (uword i = rot_tail + 1; i < index.size(); i++) {
        while(table[x[i]].back() != index[i-1]) {
          // find and erase from the table
          // x[i] needs to be removed from  table[table[x[i]].back()], and
          // table[table[x[i]].back()][x[i]] needs to be removed from
```

```
                    // table[x[i]].
                    uword tab_size = table[table[x[i]].back()].size();


                    // Remove x[i] from table[table[x[i]].back()]
                    // If x[i] is not in table[table[x[i]].back()], then it should remove
                    // nothing.
                    // This uses an 'erase-remove' idiom from the std library.
                    table[table[x[i]].back()].erase(std::remove(table[table[x[i]].back()].begin(),
                                    table[table[x[i]].back()].end(),
                                    x[i]),
                                    table[table[x[i]].back()].end());


                    // Check to see if it removed x[i] or not (whether the table's the same size)
                    if (tab_size == table[table[x[i]].back()].size()) { return matchings.zeros(); }


                    // Check to see if there's only one element remaining (if so, no stable matching.)
                    if (table[x[i]].size() == 1) { return matchings.zeros(); }


                    // Remove table[table[x[i]].back()][x[i]] from table[x[i]] (it should be at the end).
                    table[x[i]].pop_back();
                }
            }
        }
    }
}


// Check if anything is empty
for (uword i = 0; i < table.size(); i++) {
    if (table[i].empty()) { return matchings.zeros(); }
}


// Create the matchings
matchings.resize(N);
for (uword n = 0; n < N; n++) {
    matchings[n] = table[n][0];
}


return matchings;
}
```

```
//' Check if a matching solves the stable roommate problem
//'
//' This function checks if a given matching is stable for a particular set of
//' preferences. This function checks if there's an unmatched pair that would
//' rather be matched with each other than with their assigned partners.
//'
//' @param pref is a matrix with the preference order of each individual in the
//'   market. If there are \code{n} individuals, then this matrix will be of
//'   dimension \code{n-1} by \code{n}. The \code{i,j}th element refers to
//'   \code{j}'s \code{i}th most favorite partner. Preference orders must be
//'   specified using C++ indexing (starting at 0). The matrix \code{pref} must
//'   be of dimension \code{n-1} by \code{n}.
//' @param matchings is a vector of length \code{n} corresponding to the
//'   matchings that were formed (using C++ indexing). E.g. if the \code{4}th
//'   element of this vector is \code{0} then individual \code{4} was matched
//'   with individual \code{1}. If no stable matching exists, then this function
//'   returns a vector of zeros.
//' @return true if the matching is stable, false otherwise
//'  @export
bool cpp_wrapper_check_stability(umat& pref, umat& matchings) {


  // loop through everyone and check whether there's anyone else
  // who they'd rather be with
  for (uword i=0; i<pref.n_cols; i++) {
    for (uword j=i+1; j<pref.n_cols; j++) {


      // do i, j prefer to switch?
      bool i_prefers = false;
      bool j_prefers = false;


      // i?
      for (uword k=0; k<pref.n_rows; k++) {
        if (pref(k, i) == j) i_prefers = true;
        if (pref(k, i) == matchings(i)) break;
      }


      // j?
      for (uword k=0; k<pref.n_rows; k++) {
        if (pref(k, j) == j) j_prefers = true;
        if (pref(k, j) == matchings(j)) break;
      }
```

```
            // do they both want to switch?
            if (i_prefers && j_prefers) { return false; }
          }
        }


        return true;


    }
```