Министерство науки и высшего образования 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: 01.04.02 Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, к.т.н.
_____/Е.М. Сартасов
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
профессор
_____/А.А.Замышляева
«____»_____20___ г.

Исследование устойчивых паросочетаний на основе задачи о зачислении абитуриентов в вузы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.04.02.2020.068.ПЗ ВКР

Руководитель работы, к.п.н.,
доцент
_____/А.Ю. Эвнин
«____»_____2020 г.

Автор работы
Студент группы ЕТ-222
_____/М.С. Василенко
«____»_____2020 г.
Нормоконтролер,
ст. преподаватель
_____/Н.С. Мидоночева
«____»_____2020 г.

Челябинск
2020

АННОТАЦИЯ

Василенко М.С. Исследование устойчивых паросочетаний на основе задачи о зачислении абитуриентов в вузы. – Челябинск: ЮУрГУ, ЕТ-222, 72 с., 17 ил., 12 табл., библиогр. список – 25 наим., 1 прил.

В работе исследованы алгоритмы поиска устойчивых паросочетаний.

Исследована предметная область в отношении устойчивых паросочетаний. Наиболее эффективные алгоритмы поиска устойчивых паросочетаний модифицированы для задачи о зачислении абитуриентов в вузы. Проведен сравнительный анализ полученных модификаций по времени работы на основе выполненной программной реализации методов. Осуществлена проверка получаемых алгоритмами решений на устойчивость. Из анализа сделан вывод, что данные модификации выдают стабильные устойчивые паросочетания и показывают высокую эффективность по времени работы, что подтверждает теоретические оценки и делает возможным применение данных модификаций на децентрализованных рынках для устранения связанных с этим сложностей и перехода к централизации.

Программная реализация осуществлена на языке C++ в среде разработки Microsoft Visual Studio 2017. В приложении приведен текст программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

За последние полвека было сформулировано множество интересных проблем, основной идеей которых является необходимость распределения людей или объектов в пары друг с другом. В качестве примера можно рассмотреть распределение интернов по больницам, распределение сотрудников по вакансиям, сопоставление пациента и донора, распределение учащихся по школам, распределение абитуриентов по вузам. Такая необходимость существовала и ранее, например, при составлении танцующих пар, при рассадке гостей за столом, однако механизмы автоматизации данного процесса стали появляться сравнительно недавно.

Для решения подобных задач стали использовать теорию графов. Действительно, составив по исходным данным двудольный граф, легко заметить, что задача сводится к известной задаче о поиске максимального паросочетания. Однако такой поиск не учитывает индивидуальные предпочтения каждого объекта относительно своей пары. Поэтому для решения данных задач было необходимо разработать алгоритм, который занимается распределением на основе упорядоченных предпочтений каждого объекта. Такие алгоритмы стали называть алгоритмами поиска устойчивых паросочетаний (Stable matching), которые учитывают пожелания каждого объекта и не могут быть улучшены относительно удовлетворения предпочтений.

Впоследствии возникла необходимость расширить данный подход с задач распределения вида «один-к-одному», как в случае с сотрудниками и вакансиями, так как на одну вакансию обычно претендует один сотрудник, на задачи распределения вида «один-ко-многим», как в случае с абитуриентами и вузами, так как в один вуз может поступить множество студентов, но каждый студент обучается не более чем в одном вузе. Задачи распределения такого типа встречаются сейчас очень часто в различных сферах и на различных рынках, поэтому разработка эффективных решений в

этой области очень актуальна. Последние десятилетия ведутся активные исследования. Одними из новых разработок являются алгоритмы отложенного принятия решений DA (Deferred Acceptance) и алгоритм верхнего торгового цикла TTC (Top Trading Cycle).

Целью данной работы является исследование алгоритмов построения устойчивых паросочетаний вида «один-ко-многим» и сравнение их друг с другом по времени работы и эффективности распределения.

Для достижения поставленной цели необходимо решить следующие задачи:

− выполнить анализ предметной области;

− рассмотреть и модифицировать для задачи «один-ко-многим» эффективные алгоритмы поиска устойчивых паросочетаний, таких как алгоритм Гейла-Шепли, алгоритм DA, алгоритм TTC, многокритериальный серийный диктаторский механизм;

− реализовать алгоритмы на языке C++;

− протестировать алгоритмы на различных экспериментальных данных;

− проанализировать полученные результаты.

1 УСТОЙЧИВЫЕ ПАРОСОЧЕТАНИЯ

1.1 История развития теории устойчивых паросочетаний

В современном мире теория графов находит широкое применение в различных областях, например, в геоинформационных системах. Различные здания, сооружения, кварталы, населенные пункты можно рассматривать как вершины графа, а соединяющие их дороги, авиалинии, линии электропередач, инженерные сети, – как ребра графа. Применение различных алгоритмов на таких графах позволяет вычислить кратчайший путь и спланировать оптимальный маршрут.

В настоящее время теория графов содержит большое количество нерешенных проблем и задач. Теория построения паросочетаний является особым разделом теории графов с различными областями применения.

Родоначальником теории графов считают Леонарда Эйлера. В 1736 году в одном их своих писем ученый формулирует задачу о семи кенигсбергских мостах и предлагает ее решение. Впоследствии эта задача становится одной из классических задач теории графов [1].

Теория паросочетаний относится к нескольким слабо связанным областям исследований, касающихся распределения, сопоставления и обмена неделимых ресурсов и лежит на пересечении теории графов, теории игр, теории социального выбора и разработки механизмов. Паросочетание может включать обмен или распределение неделимых объектов, таких как органы для пересадки, общие комнаты, курсы, дачи и т. д. Также паросочетание может включать двустороннее распределение на рынках с двумя сторонами, таких как учащиеся и школы, фирмы и работники, абитуриенты и вузы, которые должны быть сопоставлены друг с другом.

Первый фундаментальный научный доклад по этому вопросу был написан Дэвидом Гейлом из университета Брауна и Ллойдом Шепли из Принстонского университета в 1962 году. Математики заинтересовались за-

дачей о стабильных браках, или задачей о марьяже – математической задачей из области кооперативных игр и теории графов. В изначальной формулировке в этой задаче было необходимо составить брачные пары из женихов и невест таким образом, чтобы муж из одной семьи и жена из другой не предпочитали друг друга сильнее, чем законных супругов. Гейл и Шепли представили двухстороннюю модель паросочетания и подходящую концепцию решения под названием устойчивое паросочетание и описали ее в своей статье «Поступление в колледж и стабильность браков» в журнале American Mathematical Monthly. Они также показали, что устойчивое паросочетание всегда существует и доказали этот результат с помощью простого итеративного алгоритма. Гейл и Шепли, скорее всего, не знали, что этот доклад приведет к появлению новой литературы по теории графов, которая в настоящее время обычно называется теорией паросочетаний. В 1984 году профессор Элвин Рот из Гарвардского университета показал, что тот же алгоритм, предложенный Гейлом и Шепли, был независимо открыт Национальной программой распределения резидентов в Соединенных Штатах, и с 1950-х годов использовался для подбора медицинских интернов для работы в больницах. В 1980-х годах было написано несколько статей о двухсторонней модели паросочетания и вариантах изучения стратегических и структурных вопросов, касающихся стабильности. В настоящее время теория продолжает активно разрабатываться и развиваться, и находятся все новые применения. В 2012 году Ллойд Шепли и Элвин Рот получили нобелевскую премию по экономическим наукам за фундаментальный труд под названием «Теория стабильного распределения и практики устройства рынков» [2].

Теория распределения и обмена неделимыми товарами была также активно исследована в 1974 году Шепли и Гейлом вместе с профессором и экономистом Гербертом Скарфом из Йельского университета. Ученые предложили базовую модель, которая представляла собой рынок жилья. Каждый из агентов рынка владеет домом и имеет определенные предпочтения относительно домов, включая собственный. Агентам разрешено обмениваться до-

мами, находясь в условиях биржевой экономики. Ученые выяснили, что алгоритм, предложенный Гейлом, можно применить и к данной задаче с той лишь разницей, что в данной модели рынка только одна сторона состоит из активных агентов, в то время как другая состоит из объектов. Связи между моделями двухсторонних паросочетаний и рынком жилья были также позже открыты и исследованы Балинским и Тайфуном Сонмезом из Бостонского колледжа в 1999 году, Эргином в 2002 году, Атилой Абдулкадироглу из университета Дьюка и Сонмезом в 2003 году, Клаусом и Элерсом в 2006 году, Кодзимой и Манеей в 2007 году.

Модель распределения, как правило, состоит из объектов и агентов, каждый из которых имеет предпочтения над объектами. Эти объекты распределяются между агентами. Структура прав контроля в отношении объектов обычно приводится в определении задачи. Например, каждый агент может уже иметь объекты для начала, как в задаче обмена почки (Рот, Сонмез и Унвер в 2004 году). В случае с рынком жилья некоторые агенты могут как иметь, так и не иметь предметы (Абдулкадироглу и Сонмез в 1999 году). Кроме того, могут существовать более сложные структуры прав контроля, как в задаче выбора школы, где каждая школа приоритизирует учащихся (как определено Абдулкадироглу и Сонмезом в 2003 году). Почти все эти модели имеют практическое применение. Во всех этих моделях имеется центральный планирующий орган, осуществляющий прямой механизм сбора информации о предпочтениях от агентов. Например, центральное медицинское учреждение, определяющее, какие пациенты получат почки, либо жилищное управление колледжа, выделяющее места в студенческих общежитиях. Такой центральный орган использует строго определенную и согласованную заранее процедуру, которую принято называть механизмом.

В процессе дальнейшего развития теории было установлено, что в моделях с первоначально существующими имущественными правами различная справедливая и индивидуальная защита собственности должна уважаться любым приемлемым механизмом по нормативным, институциональным или

экономическим причинам. В нормативном плане следует ожидать, что существует равная вероятность назначения объекта агентам, которые имеют одинаковые права на объекты. Рассмотрим пример, в котором при выборе школы ученики являются агентами. В процессе распределения может возникнуть ситуация, когда двое или несколько учащихся будут иметь одинаковый приоритет для школы. В таком случае, этим ученикам необходимо предоставить равные возможности для поступления. Также может возникнуть ситуация, когда ученик зачислен не в самую предпочтительную для него школу, но при этом данная школа приняла учащегося с меньшим приоритетом. В таком случае происходит ожидаемый конфликт интересов, при котором первый ученик будет завидовать второму. Потенциально семья данного ученика может даже возбудить судебный иск против школьного округа. Поэтому необходимо не только следовать определенным нормативным критериям, но и при помощи механизмов выбора устранять оправданную зависть.

Также было выяснено, что, если это возможно, механизмы должны быть не манипулируемыми. Агенты рынка, умело искажая сообщения о своих предпочтениях, не должны иметь возможности манипулировать этими системами. Это важно не только для достижения устойчивого распределения, но и для сохранения справедливости. Параг Патак из Массачусетского технологического института и Сонмез в 2008 году провели исследование, в результате которого было выяснено, что не все студенты достаточно осведомлены, чтобы успешно манипулировать механизмом. Кроме того, внедрение стратегически устойчивого механизма позволяет свести к минимуму информационную нагрузку на агентов. Им требуется лишь правильно оформить свои предпочтения и не нужно перед этим угадывать предпочтения других агентов [3].

1.2 Применение устойчивых паросочетаний

Во многих ситуациях для решения задач распределения используется централизованный механизм, в котором существует некий единый орган, со-

бирающий предпочтения сторон и информацию о квотах с помощью информационной системы или вручную. В ряде других случаев используются псевдо-централизованные механизмы, когда устанавливаются единые строгие правила взаимодействия между сторонами, например, абитуриентами и вузами, но итоговое распределение происходит в результате индивидуальных ограниченных регламентом взаимодействий, как, например, в приемных кампаниях в вузах России и Украины. Еще один подход использует полностью децентрализованную схему. При таком подходе взаимодействия между сторонами никак не регламентированы, и примером такого подхода является система поступления в вузы США. Далее будет проведен обзор применяемых на практике централизованных схем распределения [4].

1.2.1 Программа распределения студентов в терапевтической интернатуре США

В течение долгого времени в Соединенных Штатах Америки конкуренция среди госпиталей за потенциальных интернов была так высока, что зачастую больницы делали предложения интересующим их студентам как можно раньше. В результате студенты были вынуждены выбирать место прохождения интернатуры за несколько лет до окончания курса обучения и принимать одно из первых предложений, боясь в случае отказа остаться без места. Позднее, в 1945 г. впервые были установлены единые правила распределения интернов по больницам. В то же время была создана Национальная программа распределения резидентов NRMP.

Для борьбы с проблемой ранних предложений были введены ограничения по времени, когда больницы могут предлагать места студентам. В итоге студенты стали вынуждены давать свой ответ на поступившее предложение в очень сжатые сроки, вплоть до нескольких часов. Впоследствии в 1990-х годах при использовании аналогичных децентрализованных систем распределения возникали ситуации, когда студент в течении 35-минутного перелета на самолете получал одновременно предложение о приеме на работу, затем

сообщение о необходимости дать срочный ответ, и затем отзыв предложения из-за отсутствия ответа [5].

Элвин Рот в одном из своих исследований показал, что использовавшиеся до 1952 г. механизмы распределения порождали неустойчивые паросочетания [6]. Если полученное в результате процесса распределения паросочетание не является устойчивым, то после объявления результатов существует риск, что найдутся недовольные агенты, которые хотят изменить назначения на более выгодные для них. Поэтому для устранения недовольств и конфликтов путем получения устойчивых распределений в 1952 г. был внедрен централизованный алгоритм, схожий с механизмом Гейла-Шепли. Студенты подавали заявления и проходили собеседования осенью, после чего в феврале студенты и больницы отправляли свои предпочтения в центр сбора и обработки информации. Все распределения проводились в один день при помощи компьютерных вычислений, и этот день был назван «днем распределения» [5]. Данный механизм успешно применялся до 1995 г.

Механизм, применяющийся с 1952 года, выбирал из всех возможных устойчивых паросочетаний те, которые являлись наилучшими по предпочениям для больниц. Более того, в данном механизме наилучшим поведением для интернов было всегда честно сообщать свои предпочтения, как показал позже Элвин Рот. При этом больницам, квота свободных мест в которых равнялась единице, искажать свои предпочтения также было невыгодно.

В процессе модификации механизма распределения Элвином Ротом и его коллегой Эллиотом Перансоном в 1995–1999 гг. [7] были приняты во внимание многочисленные обращения интернов по поводу ориентированности существующего механизма на интересы больниц. В результате было принято решение выбрать версию алгоритма с предлагающими студентами, что сделало процесс распределения ориентированным на интересы будущих докторов.

В середине 90-х годов прошлого века начал меняться гендерный состав выпускников медицинских вузов. Высшее образование в медицинской сфере

получало все больше женщин, что естественно стало предпосылкой для образования все большего количества семейных пар в среде молодых докторов. Несмотря на постепенный характер изменений, ситуация привела к неожиданным эффектам, так как молодые супруги по объективным причинам не были готовы разлучаться и хотели получить два хороших места в госпиталях одного и того же города. При этом существующий механизм распределения не мог учитывать такие пожелания и часто распределял молодоженов в разные города. Когда пара оказывалась перед необходимостью принятия назначения мужа в Чикаго, а жены в Нью-Йорк, единственным решением становилась попытка договориться с больницами в обход централизованной системы и поискать назначение в одном городе вручную. Это ставило всю систему под угрозу.

В процессе решения данной проблемы Рот и Перансон разработали модификацию алгоритма [7], которая позволила учесть просьбы и предпочтения семейных пар, однако наложив несущественные ограничения. Данная разработка была основана на последовательной процедуре случайного устранения нестабильностей Рота и Ванде Вата [8].

1.2.2 Процесс распределения учащихся по школам

Процесс распределения учеников по школам отличается от классической задачи распределения. Особенность процесса состоит в том, что обычно школы, в отличие от вузов или компаний, не проводят каких-либо специальных вступительных испытаний. Критерии упорядочения детей часто не являются строгими. Кроме того, проживание в непосредственной близости от учебного заведения, наличие социальных льгот или факт обучения в заведении братьев и сестер может повышать приоритет некоторых ребят перед школой. Поэтому часто школы не могут сообщить строго упорядоченный по предпочтительности список учеников.

Рассмотрим существовавший до 2001 г. механизм распределения и приема учеников в Бостоне и школьных округах Нью-Йорка [9, 10]. Перед

началом процесса распределения каждый учащийся, желающий в этом году поступить в учебное заведение, вместе с родителями составлял список школ по убыванию предпочтительности. Далее процесс проходил в несколько этапов. На первом рассматривались школы, которые учащиеся поставили в своих списках предпочтений на первое место. В случае если мест в какой-либо школе оказывалось недостаточно для зачисления всех указавших ее на первой позиции, принимались в рассмотрение социальные и какие-либо еще льготы учеников. Если принять однозначное решение все еще было невозможно, применялся механизм случайного выбора.

Все учащиеся, не получившие место в каком-либо учебном заведении на первом этапе, переходили в следующий этап. В рассмотрение на втором этапе шли школы, которые занимали второе место по привлекательности в списках ребят, оставшихся не зачисленными, и так далее. Однако порождаемое данным механизмом распределение являлось неустойчивым, так как если школа, занимающая второе место в списках учеников, была заполнена еще на первом этапе, то учащиеся снова не получали мест. В процессе реформирования данной системы Элвин Рот вместе с Атилой Абдулкадироглы, Парагом Патаком и Тайфуном Сонмезом выяснили, что при использовании такой схемы распределения учащимся и их родителям было невыгодно честно сообщать свои настоящие предпочтения. Руководители образовательных округов откровенно рекомендовали родителям думать стратегически при составлении своего списка предпочтений, в результате чего некоторые семьи в результате хитрых манипуляций предпочтениями получали дополнительное преимущество. Семьи, честно сообщавшие свои предпочтения несмотря на предупреждения руководства округа, значительно проигрывали и получали крайне нежелательные для себя школы.

Учеными было предложено два варианта нового механизма распределения. Оба варианта гарантировали невозможность манипулируемости со стороны учеников. В случае распределения в государственные учебные заведения устойчивость как эквивалент справедливости и установленных зако-

ном приоритетов является необходимым фактором, поэтому руководителями был выбран именно алгоритм Гейла-Шепли, так как порождаемое им паросочетание всегда является устойчивым, поэтому никакой ученик или его семья не могут заявить, что в той школе, куда он хотел бы попасть, учится ребенок с меньшим приоритетом.

1.2.3 Программа пересадки почек

Ежегодно во всем мире огромное количество людей нуждается в пересадке органов. Наиболее частой необходимостью является пересадка почки. Пересаживать органы можно только по добровольному желанию живого донора на безвозмездной основе, так как человек может жить полноценной жизнью лишь с одной почкой, или по завещанию, разрешающую пересадку от трупного донора. Количество органов от трупных доноров значительно меньше существующей необходимости, поэтому активно привлекается возможность пересадки почки от живого донора. В Соединенных Штатах Америки любой живой донор может безвозмездно пожертвовать свою почку нуждающемуся пациенту, например, родственнику или другу. Однако часто случается так, что донор и пациент несовместимы друг с другом по группе крови или каким-либо иным медицинским факторам, поэтому помощь становится невозможной.

Вследствие этого был основан научный фундамент системы обмена почками между такими несовместимыми парами. Донор из первой пары отдает почку реципиенту из второй, в то время как донор из второй пары отдает ее реципиенту из первой. Такая идея была предложена Элвином Ротом, Тайфуном Сонмезом и их коллегой Утку Унвером из Бостонского колледжа [11–13]. Использование этой идеи позволило оказывать помощь гораздо большему количеству нуждающихся и попутно уменьшить время ожидания жизненно важных операций.

1.2.4 Проблемы и трудности децентрализованных рынков

Несмотря на все преимущества централизованных схем распределения, довольно часто агенты рынков предпочитают договариваться в обход системы, не доверяя системе и друг другу. Элвин Рот вместе с Муриэлем Нидерле из Стэнфордского университета проанализировали различные рынки [14] и на примере распределения молодых гастроэнтерологов по больницам определили три основные проблемы децентрализованных рынков: перегруженность, широту охвата и недостаток безопасности.

Первая проблема перегруженности заключается в том, что в отсутствие централизованного механизма распределения любое предложение позиции больницей производится вручную. При такой схеме ответ будущего доктора занимает достаточно большое время. Поэтому госпиталь просто за счет дефицита времени может не иметь возможности предложить работу всем интересующим его кандидатам, что вынуждает делать предложения как можно раньше и требовать почти немедленного ответа. В итоге некоторые студенты должны были выбирать место прохождения интернатуры как минимум за год до выпускного вечера и принимать какое-либо из первых предложений, так как у них не оставалось физической возможности рассмотреть все возможные интересующие их варианты и существовал страх в случае отказа остаться без места.

Вторая выделенная проблема децентрализованных рынков – сужение охвата. При анализе данных о различных распределениях докторов было выяснено, что при использовании централизованной процедуры распределения студенты имели статистически более высокие шансы получить позицию в больнице, городе и даже штате, отличном от тех, где они получили образование. Фактически при использовании децентрализованной схемы распределения рынок постепенно распадался на отдельные раздробленные по территориальному признаку сегменты. Данная проблема является следствием из проблемы перегруженности, потому что в условиях конкуренции и дефицита времени на собеседование госпитали имели больше возможностей получить

информацию о молодых будущих докторах из своей местности, чем о кандидатах из других регионов страны.

Третья выделенная проблема недостатка безопасности связана с первыми двумя. Существующее недоверие к централизованной схеме распределения и опасения, что при условии участия в централизованной программе на рынке будут оставаться игроки, по-прежнему делающие предложения в обход системы, приводили к нежеланию госпиталей участвовать в программах централизации. Некоторые выпускники медицинских вузов США даже подавали иски к организаторам централизованных программ по поводу неконкурентного, по их мнению, назначения заработных плат. Рот и Нидерле использовали данные по статистике заработных плат и в итоге показали, что децентрализация системы распределения интернов не приводит к росту их доходов, поэтому такие иски не имеют под собой достаточного основания. В результате исследований было также обнаружено, что отказ некоторых агентов рынка от участия в централизованном механизме распределения был вызван сочетанием двух факторов. Из-за значительных изменений в системе подготовки молодых гастроэнтерологов все меньше выпускников хотели становиться интернами по данной специальности, но при этом они, в отличие от больниц, не имели понятия о значительном сокращении кандидатов на эти места. Поэтому в 1997 г., получая предложение до начала официальной процедуры распределения, молодые врачи в основном не отказывались от него, ожидая высокой конкуренции за места во время централизованного распределения. В исследованиях и экспериментах было показано, что каждый из таких факторов по отдельности не приводит к произвольному отказу от централизованной процедуры распределения, поскольку большинство агентов не заинтересованы в этом.

Значительный интерес представляют проведенные исследователями эксперименты, показавшие, что агенты не способны прийти к равновесным стратегиям, даже если проходят обучение, т. е. участвуют в соответствующей игре большое количество раз. В конечном итоге то, что было предложено

разработчиками механизма для предоставления дополнительных возможностей лучшим агентам, оборачивается ухудшением положения этой группы агентов рынка [4].

1.2.5 Программа распределения помощников федеральных судей

После окончания университета до начала самостоятельно практики молодые юристы с целью приобретения возможности набраться опыта и завести контакты становятся, как правило, помощниками федеральных судей, которые тоже заинтересованы получить помощника в своей деятельности. При этом процесс распределения происходит абсолютно децентрализованно. Всякий раз при попытке установления единых сроков начала процесса распределения на данном рынке эти сроки быстро начинают нарушаться. Привлекая к процессу профессиональных ученых-юристов, Элвин Рот совместно с исследователями в сфере экспериментальной экономики заинтересовался этой проблемой [15–16].

Основные проблемы при распределении помощников судей, как было выявлено в результате проведения масштабных исследований и опросов, схожи с проблемами, возникающими на других децентрализованных рынках. Из-за недоверия к централизованным схемам и другим игрокам рынка, а также из-за отсутствия четкого понимания процессов централизованных механизмов как судьи, так и молодые выпускники верят в то, будущему клерку выгоднее всегда соглашаться на первое поступившее предложение. Это и стало основной преградой на пути к избавлению от порождаемой такими распределениями неэффективности. Переходу к более эффективному распределению мешает уверенность каждого из агентов рынка в том, что другие не будут согласны на такой переход.

Вот некоторые примеры высказываний судей по этому вопросу. Один из опрошенных утверждал, что, с точки зрения судьи, заблаговременное предложение позволяет ему привлечь кандидатов, которые в противном случае не стали бы серьезно рассматривать его на должность клерка. Другой же

говорил, что так как его дом и его офис расположены в небольшом городке, то далеко не каждый юноша или девушка захотят жить здесь. Он придерживался позиции не нанимать на работу тех, кто еще не закончил первый семестр своего первого года обучения в юридической школе. При этом судья обнаружил, что в каждом классе среди лучших студентов есть те, кто скорее предпочтут получить работу раньше даже в таком маленьком городке, чем ждать и соревноваться за получение лучших мест в большом городе [5].

1.3 Основные определения теории устойчивых паросочетаний

Исследования в данной работе опираются на теорию графов, поэтому необходимо напомнить некоторые определения из этой теории.

Простым графом называется упорядоченная пара $G = \langle V, E \rangle$, где V – непустое конечное множество (элементы V – вершины графа), а E – конечное множество неупорядоченных пар различных элементов V (элементы E – ребра графа).

Петля – ребро, соединяющее вершину саму с собой, или, по-другому, инцидентное данной вершине.

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

В данной работе будут рассмотрены только простые графы, без петель и кратных ребер.

Двудольный граф – это граф $G = \langle V, E \rangle$, вершины которого можно разбить на два множества $V_1 \subset V$ и $V_2 \subset V$ таких, что ребра могут соединять только вершины из разных множеств, так как в каждом множестве никакие две вершины не соединены друг с другом ребром [17–19].

1.3.1 Понятие устойчивого паросочетания

Рассмотрим неориентированный двудольный граф $G = (M \cup W, E)$, где M и W – конечные непересекающиеся между собой подмножества вершин

графа, а E – множество ребер, соединяющее эти вершины. У каждого ребра $e \in E$ есть две концевые вершины $m \in M$ и $w \in W$, поэтому $e = \{m, w\}$.

Определение 1.1. Паросочетанием в двудольном графе называется подмножество ребер $F \in E$ такое, что любая вершина $v \in (M \cup W)$ инцидентна не более чем одному ребру [17–19].

Классической задачей теории графов является задача поиска паросочетания максимальной мощности в заданном графе.

Расширим классическую модель. Пусть теперь необходимо найти такое паросочетание максимальной мощности, которое учитывает предпочтения вершин или, точнее, соответствующих вершинам рациональных агентов. Такие предпочтения описываются бинарными отношениями.

На множестве X задано бинарное отношение P, если $P \subseteq X \times X$. Если пара $(x, y) \in P$, обозначим это как xPy. Если $(x, y) \notin P$, обозначим это как $xP^c y$.

Определение 1.2. Линейным порядком [20] называется бинарное отношение P, заданное на множестве X и обладающее свойствами:

- связности, т. е. $\forall x, y \in X$, если $x \neq y$, то или xPy, или yPx;
- транзитивности, т. е. $\forall x, y, z \in X$, если xPy, yPz, то xPz;
- асимметричности, т. е. $\forall x, y \in X$, если xPy, то $yP^c x$.

Задание на множестве отношения предпочтения, являющегося линейным порядком, фактически означает, что все элементы данного множества упорядочены от наиболее предпочтительного до наименее предпочтительного. Предпочтения каждой вершины $m \in M$ описываются бинарным отношением P_m, заданным на множестве $W \cup m$ и являющимся линейным порядком. Агент m упорядочивает по предпочтительности всех агентов множества W и опцию «не быть сопоставленным». Аналогично, отношение предпочтения P_w каждой вершины $w \in W$ задано на множестве $M \cup w$ и является линейным порядком. Совокупность всех отношений предпочтений называется профилем предпочтений и обозначается P. Будем говорить, что $m(w)$ допустим(а)

для $w(m)$, если $mP_w w$ и, соответственно, $wP_m m$. Ребра в графе G могут связывать вершины m и w, в случае если только они являются допустимыми друг для друга.

Определение 1.3. Обобщенное паросочетание [20] μ – это взаимнооднозначное отображение $\mu: M \cup W \to M \cup W$ такое, что:

• $\forall w \in W$ если $\mu(w) = m$, то $\mu(m) = w$, и, аналогично, $\forall m \in M$ если $\mu(m) = w$, то $\mu(w) = m$;

• $\forall m \in M\ \mu(m) = w$ или $\mu(m) = m$, и, аналогично, $\forall w \in W\ \mu(w) = m$ или $\mu(w) = w$.

Обобщенное паросочетание μ соответствует паросочетанию F в графе G в следующем смысле: если $\mu(m) = w$, то ребро $e = \{m, w\} \in F$. В дальнейшем эти два понятия будут считаться эквивалентными и слово «обобщенное» будет опускаться.

Для того, чтобы учесть предпочтения сторон, был предложен подход, основанный на поиске устойчивого паросочетания. Основная идея данного подхода состоит в том, что никакая вершина, а именно соответствующий ей агент или пара вершин из различных множеств не должны иметь стимула отказаться от предложенного варианта распределения с выгодой для себя при условии, что все остальные вершины остаются в паросочетании.

Определение 1.4. Паросочетание μ устойчиво [20], если оно удовлетворяет требованиям:

• индивидуальной рациональности: $\forall x \in M \cup W$ таких, что $\mu(x) \neq x$ верно, что $\mu(x) P_x x$, то есть для каждого агента, который получил пару, партнер $\mu(x)$ является допустимым;

• попарной устойчивости: не существует $m \in M$ и $w \in W$ таких, что одновременно $mP_w \mu(w)$ и $wP_m \mu(m)$, т.е. не существует такой пары, в которой каждый предпочитает другого партнера по сравнению с полученными.

Определение 1.5. Пара (m, w) называется блокирующей паросочетание μ [20], если ее присутствие нарушает устойчивость паросочетания μ в соответствии со второй частью определения 1.4.

1.3.2 Пример профилей предпочтения и устойчивого паросочетания

Рассмотрим пример, иллюстрирующий профили предпочтения группы агентов, различные паросочетания на данном графе, наличие блокирующих пар в таких паросочетаниях и устойчивое паросочетание.

Две группы агентов состоят из трех человек каждая, первая группа – мужчины: Albert, Bradley, Charles. В таблице 1.1 представлена информация о профилях предпочтений мужчин, отсортированных по убыванию предпочтительности слева направо.

Таблица 1.1 – Профили предпочтений мужчин

Albert	Diane	Emily	Fergie
Bradley	Emily	Diane	Fergie
Charles	Diane	Emily	Fergie

Вторая группа агентов – женщины: Diane, Emily, Fergie. Их профили также представлены в таблице 1.2 и отсортированы по убыванию.

Таблица 1.2 – Профили предпочтений женщин

Diane	Bradley	Albert	Charles
Emily	Albert	Bradley	Charles
Fergie	Albert	Bradley	Charles

Необходимо найти в данном полном двудольном графе устойчивое паросочетание. Рассмотрим первое возможное паросочетание, представленное в таблице 1.3. Зеленым в профилях выделены партнеры в данном паросочетании.

Данное паросочетание не является устойчивым, так как в нем присутствует блокирующая пара, помеченная красным в таблице 1.4.

Таблица 1.3 – Возможное паросочетание

Albert	Diane	Emily	Fergie
Bradley	Emily	Diane	Fergie
Charles	Diane	Emily	Fergie
Diane	Bradley	Albert	Charles
Emily	Albert	Bradley	Charles
Fergie	Albert	Bradley	Charles

Таблица 1.4 – Блокирующие пары

Albert	Diane	Emily	Fergie
Bradley	Emily	Diane	Fergie
Charles	Diane	Emily	Fergie
Diane	Bradley	Albert	Charles
Emily	Albert	Bradley	Charles
Fergie	Albert	Bradley	Charles

Перестроим паросочетание, как показано в таблице 1.5.

У некоторых агентов предпочтения учтены не лучшим образом, однако блокирующих пар в паросочетании нет, и оно является устойчивым.

Таблица 1.5 – Устойчивое паросочетание

Albert	Diane	Emily	Fergie
Bradley	Emily	Diane	Fergie
Charles	Diane	Emily	Fergie
Diane	Bradley	Albert	Charles
Emily	Albert	Bradley	Charles
Fergie	Albert	Bradley	Charles

1.4 Выводы по разделу

Теория устойчивых паросочетаний позволяет эффективно распределять объекты и ставить в соответствие друг другу агентов на различных рынках.

В этом разделе были рассмотрены различные практические применения устойчивых паросочетаний, каждое из которых выделяет некоторый класс задач. Также были объяснены основные понятия теории графов и теории устойчивых паросочетаний, которые будут использоваться далее в работе.

В следующем разделе будут рассмотрены алгоритмы поиска устойчивых паросочетаний, предложены их улучшения и расширено их применение для задачи вида «один-ко-многим».

2 АЛГОРИТМЫ ПОСТРОЕНИЯ УСТОЙЧИВЫХ ПАРОСОЧЕТАНИЙ

2.1 Анализ существующих механизмов поступления абитуриентов в вузы

В предыдущем разделе была рассмотрена задача о построении устойчивого паросочетания в том случае, когда каждый из агентов составляет пару максимум с одним агентом из другой группы, а стороны рынка симметричны. Расширим эту модель. Предположим, что некоторые агенты одной из сторон могут составлять пары сразу с несколькими игроками другой стороны. Такая модель позволяет описать реальную структуру отношений в таких ситуациях, как сопоставление фирм, предлагающих несколько вакансий, и работников, желающих получить место работы. Также в качестве примера можно рассмотреть школы, предлагающие места на новый учебный год, и учеников, желающих поступить в эти учебные заведения. Этот пример можно расширить до более сложного, добавив выпускные экзамены и вступительные испытания, и получить взаимоотношения между вузами и абитуриентами, как по сложившейся традиции и принято называть данную задачу. Следует заметить, что рассматриваемая модель является по своей сути абстрактной, а результаты применимы к любым подобным системам.

Процесс зачисления абитуриентов в вузы является очень сложным и часто зависит от исторических особенностей развития образования, в том числе высшего. Централизованные схемы зачисления абитуриентов в вузы существуют в Турции, Венгрии, Ирландии, Германии, Китае и многих других странах. Централизованная процедура может охватывать как все вузы, как в Турции или Ирландии, так и отдельные специальности, например, как в Германии.

2.1.1 Поступление абитуриентов в вузы Турции

В Турции, как и во множестве стран мира, прием в вузы проводится на основании заранее определенного конкурса, основанием отбора на котором являются баллы, полученные выпускником на едином экзамене. Этот экзамен отдаленно напоминает российский Единый государственный экзамен. Все программы бакалавриата разделены на определенное количество предметных категорий, для каждой из которых специалистами предлагается формула для вычисления среднего балла выпускника. Абитуриенту необходимо сдать экзамен по разным предметам, необходимым для поступления на выбранную категорию. Для каждой категории наибольшую важность будут иметь предметы, более всего соответствующие профилю категории. В формуле для естественно-научного направления наибольший вес имеют результаты экзамена по естественным наукам, а для гуманитарного направления наибольший вес у экзамена по языку. Для вузов всей страны устанавливается единый критерий и набор весов по каждому направлению, чтобы обеспечить одинаковое упорядочивание абитуриентов в зависимости от направления обучения.

Для распределения абитуриентов в Турции [21] используется многокритериальный серийный диктаторский механизм (multi-categoriy serial dictatorship, MCSD). Процедура серийного диктатора выглядит следующим образом. Процедуры проводятся независимо между категориями. На каждом этапе сначала право выбора получает абитуриент, лучший из оставшихся в этой предметной категории. Затем право выбора переходит к следующему, и так далее, до тех пор, пока в вузах есть места. В результате работы этого алгоритма некоторые абитуриенты могут получить места сразу в нескольких вузах или факультетах, которые относятся к разным предметным направлениям. В таком случае абитуриенту предоставляется право выбора лучшего места из тех, что ему предлагаются распределением.

MCSD эквивалентен механизму отложенного принятия с предлагающими вузами, хотя и можно сделать предположение, что механизм изначаль-

но ориентирован на интересы и предпочтения студентов. Это означает, что механизм обладает всеми свойствами механизма отложенного принятия: порождает устойчивое паросочетание, наихудшее из всех устойчивых для абитуриентов и создает абитуриентам стимулы для искажения предпочтений. Когда предпочтения всех агентов одной из сторон рынка устроены одинаково (например, все вузы упорядочивают абитуриентов по предпочтительности в соответствии со средним баллом аттестата), то существует единственное устойчивое паросочетание. Соответственно, любой устойчивый механизм строит это паросочетание.

2.1.2 Поступление абитуриентов в вузы Германии

Система поступления абитуриентов в вузы в Германии также представляет собой большой интерес. Особое внимание стоит обратить на набор студентов в медицинские вузы и университеты.

Централизованная процедура распределения предусмотрена только для будущих медицинских работников. Все места, выделенные для зачисления, делятся на три группы: места, зарезервированные для лучших абитуриентов (20% мест), места, зарезервированные для давно пытающихся поступить абитуриентов (20%) и свободные места (60%). Значительный исследовательский интерес представляет сама централизованная процедура распределения абитуриентов. Первыми получают возможность поступления лучшие абитуриенты. Для них специально выделены для них 20% мест. Каждый абитуриент из числа выдающихся сообщает свои предпочтения. После этого абитуриенты проходят процедуру зачисления в вузы по Бостонскому механизму, или многокритериальному механизму серийного диктатора. Это означает, что сначала вузы рассматривают только абитуриентов, поставивших их на первое место в своих списках предпочтений. Затем, в случае если остались свободные места из квоты для лучших, в рассмотрение берутся абитуриенты, поставившие вуз на второе место, и так далее. В данном механизме существует определенная несправедливость распределения. Если абитуриент поставил на

первые два места сильные вузы, переоценил свои силы и не прошел в первый вуз, то он рискует не пройти так же и во второй, потому что все места будут заполнены на первом же шаге. Похожая по сути процедура проводится и для группы абитуриентов, давно пытающихся поступить в университет. После проведения этих сложных процедур зачисления формируется общий список абитуриентов, не получивших места, и незаполненных мест. Далее для заполнения всех оставшихся мест используется механизм отложенного принятия (Deferred Acceptance) с предлагающими вузами. В общем случае такая система создана для того, чтобы дать самым сильным абитуриентам и давно пытающимся поступить взрослым дополнительные возможности. Показано, абитуриенты из приоритетных групп могут легко манипулировать своими предпочтениями для получения более выгодного для них распределения. Более того, показано, что, только манипулируя, т.е. изменяя порядок вузов по предпочтительности или исключая вузы из списка допустимых, абитуриент может использовать предоставленный ему приоритет в своих интересах. В любом равновесии соответствующей игры, где стратегия каждого абитуриента – сообщаемый упорядоченный список вузов, а выигрыш – получаемое место учебы, окончательное паросочетание будет устойчивым.

2.1.3 Поступление абитуриентов в вузы России

Система поступления в вузы Российской Федерации не может быть однозначно отнесена к централизованной или, наоборот, децентрализованной. В Российской Федерации абитуриенты оцениваются вузом в соответствии с суммой баллов, набранных ими в сумме по 3 либо 4 предметам на Едином государственном экзамене (ЕГЭ). Также при поступлении учитываются различные достижения в олимпиадном движении, участии в конференциях и конкурсах всероссийского и международного уровня. Перечень предметов, необходимых к сдаче для желающих поступить абитуриентов, утверждается Министерством образования Российской Федерации для каждой специальности и является единым для всех вузов и университетов, осуществляющих

обучение по данной специальности. Приемная кампания включает этап подачи заявлений и этап зачисления.

На этапе подачи заявлений абитуриенты подают заявления в интересующие их вузы. С 2010 г. по настоящее время в России было введено ограничение на число вузов, в которые мог подавать заявление абитуриент. Количество заявлений не может превышать пяти. При этом в каждом вузе абитуриент может подавать заявление не более чем на три специальности. Такие ограничения введены для того, чтобы сильные абитуриенты не подавали заявления в большое количество вузов одновременно, искажая видимый процесс поступления и приводя затем более слабые вузы к ситуации недобора абитуриентов. Заявления принимаются приемными комиссиями вузов до середины либо до конца июля и только при наличии копии аттестата о полном среднем образовании и результатов ЕГЭ по предметам, сдаваемым при приеме на эту специальность. Достижения в олимпиадном движении и участие в конкурсах и конференциях также необходимо подтверждать соответствующими документами для получения дополнительных баллов. Необходимо отметить, что, если абитуриент подаёт заявление на одну и ту же специальность в разные вузы, он имеет одинаковую сумму баллов по результатам ЕГЭ.

На этапе зачисления абитуриентов, которые следует сразу после этапа приема документов, проходит две волны. В первой волне вузы объявляют списки абитуриентов, которых они готовы принять, сортируя всех подавших заявления выпускников по каждому направлению по сумме баллов ЕГЭ и достижениям в олимпиадном движении, а также принимая во внимание наличие других льгот. Если в вузе достаточно свободных мест, то все подавшие заявления включаются в список принятых на первой волне абитуриентов. До 4 августа абитуриенты должны определиться с выбором учебного заведения и принести подлинник аттестата в один из вузов, включивших их в список. Если до установленного срока включенный в список принимаемых абитури-

ент не предоставил подлинник, то он выбывает из дальнейшего конкурса на эту специальность в этом вузе.

Все принятые на первом этапе, но не предоставившие подлинник абитуриенты исключаются из списков на второй волне зачисления. Если абитуриент принес подлинник аттестата на первом этапе зачисления, то он имеет право забрать его и принести в другой вуз, готовый принять его на втором этапе и более предпочтительный. Далее снова устанавливается срок, до которого абитуриент может принести в вуз подлинник аттестата или другого документа об образовании. После этого централизованная приемная кампания заканчивается, и вузы издают приказы о зачислении абитуриентов.

Действительно, постановка проблемы крайне близка к оригинальной модели Гейла и Шепли. Агенты обеих сторон рынка создают списки предпочтений относительно другой стороны. Абитуриенты формулируют предпочтения на множестве факультетов, а вузы ранжируют абитуриентов по результатам сдачи ЕГЭ. 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 недоберут абитуриентов вузы среднего уровня. Таким образом, существующая система требует манипулирования агентов обеих сторон рынка [4].

2.2 Постановка задачи о зачислении абитуриентов в вузы

Для того чтобы перейти к задаче о зачислении абитуриентов в вузы, вместо множества мужчин введем множество университетов, а вместо мно-

жества женщин – множество кандидатов, подающих заявления на поступление. В каждом университете есть квота на количество студентов, которое университет может принять. Также существует фиктивный университет, который означает, что кандидату придется попробовать поступить через год.

Пусть $S = \{s_1, s_2, \dots, s_n\}$ – множество абитуриентов, $C = \{c_1, c_2, \dots, c_m\}$ – множество университетов. Каждый университет состоит из множества факультетов – $c_i = \{f_1, f_2, \dots, f_p\}$. Абитуриент подает документы на конкретный факультет некоторого вуза, поэтому для абитуриента различные факультеты фактически являются различными вузами. У каждого вуза есть общая квота на количество абитуриентов, которое он может принять, и эта квота делится между всеми факультетами. Абитуриент сдает единый экзамен по различным предметам, а также может иметь достижения в области олимпиадного движения, конференций и конкурсов, что является дополнительным критерием ранжирования абитуриента. У каждого факультета в университете свое направление обучения и, соответственно, свои профили предпочтений в отношении студентов, поэтому будет справедливо рассматривать в качестве агентов не вузы, а факультеты вузов [24–25].

Итак, $S = \{s_1, s_2, \dots, s_n\}$ – множество абитуриентов, $F = \{f_1, f_2, \dots, f_k\}$ – множество факультетов различных университетов, $Q = \{q_1, q_2, \dots, q_k\}$ – множество соответственных квот на прием абитуриентов различных факультетов. Предполагается, что $k < n$. Каждый абитуриент s_i имеет профиль предпочтений P_{s_i} на множестве факультетов. Для всех абитуриентов действует общее ограничение на количество факультетов, на которые можно подать заявления, и отображается небольшой константой K. Каждый факультет f_j задает критерии в отношении ранжирования абитуриентов, и эти критерии позволяют факультету составить на множестве студентов транзитивные, линейные и строгие предпочтения, которые отображаются профилем P_{f_j}.

Каждый абитуриент может быть зачислен не более чем на один факультет. Каждый факультет может принять абитуриентов не более, чем поз-

воляет размер квоты. Распределение является неустойчивым, если хотя бы для одного абитуриента s_i существует возможность перейти на более предпочтительный факультет f_j. Возможность перехода означает, что f_j заинтересован в приеме s_i и имеет незаполненную квоту, либо s_i занимает более высокую позицию в профиле предпочтений P_{f_j}, чем один или несколько абитуриентов, зачисленных на f_j. В этом случае будем говорить, что пара (s_i, f_j) блокирует паросочетание μ.

Паросочетание μ является устойчивым, если отсутствуют блокирующие его пары. Факультет f_j называется достижимым для абитуриента s_i, если существует устойчивое паросочетание μ, для которого $(s_i, f_j) \in \mu$. С учетом того, что для одной и той же системы предпочтений в общем случае может существовать несколько устойчивых распределений, назовем распределение оптимальным, если каждый абитуриент получает наилучший для него факультет из достижимых. Такое решение основано на определенных особенностях механизмов распределения, рассматриваемых далее, и доказанных фактах о том, что наилучшее распределение для абитуриентов является наилучшим для обеих сторон агентов, так как сводит к минимуму манипулирование квотами и предпочтениями. Для снижения риска манипулирования списки предпочтений подаются в один день, и зачисление проходит также в один этап.

2.3 Многокритериальный серийный диктаторский механизм (MCSD)

Механизм серийного диктатора, или механизм приоритетов, определяет порядок, в котором позволяет первому агенту получить его наиболее желаемый объект, следующему агенту – его наиболее желаемый объект среди оставшихся, и так далее. Данный механизм берет свое начало из процесса распределения детей по школам в Бостоне, поэтому иногда его называют бостонским механизмом.

Более формально, механизм сперва определяет функцию упорядочения f, где $f(i)$ – это агент с i-м приоритетом. Механизм серийного диктатора очень прост в исполнении: необходимо определить порядок случайным образом или на основе привилегий, и позволить агентам делать выбор.

Далее представлена модель алгоритма серийного диктатора.

1. Каждый абитуриент отправляет свой рейтинг предпочтений.

2. На данном этапе берутся в рассмотрение только первые факультеты в списке каждого абитуриента. Для каждого факультета рассматриваются абитуриенты, которые указали его в качестве своего лучшего выбора, и места на этих факультетах назначаются этим абитуриентам по одному в порядке приоритетов до тех пор, пока либо закончатся свободные места, либо не останется ни одного абитуриента, который перечислил его как лучший выбор.

3. На шаге с номером k берутся в рассмотрение оставшиеся абитуриенты и только факультеты, находящиеся на позиции k в их профилях предпочтений. Для каждого такого факультета, на котором еще остались места, рассматриваются абитуриенты, которые указали эти факультеты в качестве k-го выбора, и места на этих факультетах назначаются этим абитуриентам по одному в порядке приоритетов до тех пор, пока либо закончатся свободные места, либо не останется ни одного абитуриента, который перечислил его как k-й выбор [23].

Алгоритм завершит выполнение за конечное число шагов, а именно за размер наибольшего из списков предпочтений абитуриентов. Точная оценка времени работы алгоритма зависит от деталей реализации, но она будет пропорциональна размеру списков предпочтений и сумме количества абитуриентов и количества факультетов.

Данный механизм не является стратегически оптимальным. Если абитуриенты ставят на вершину списка предпочтений два сильных факультета и не проходят на первом шаге, то с большой вероятностью они не поступят и на второй в очереди, потому что сильные факультеты будут заняты в первую

очередь. Слабые факультеты при этом могут недобрать студентов. Поэтому механизмом легко, и даже необходимо, манипулировать. Однако, он используется и сейчас на многих рынках распределений из-за своей простоты в использовании и легкости в реализации.

2.4 Механизм отложенного принятия (Deferred Acceptance Algorithm)

Для решения задачи поиска устойчивого паросочетания Гейл и Шепли разработали алгоритм, впоследствии доработанный и названный механизмом отложенного принятия.

В зависимости от того, какое из подмножеств вершин M или W имеет преимущество, существуют две симметричные версии алгоритма. Вернемся к классической модели задачи «один-к-одному», представленной задачей о супружеских парах. Рассмотрим версию механизма, где преимущество имеет подмножество M, и назовем ее механизмом отложенного принятия с предлагающими мужчинами. Алгоритм состоит из повторяющихся шагов. В результате выполнения каждого шага строится временное паросочетание. Данный алгоритм по сути является конструктивным доказательством существования устойчивого паросочетания для любого двудольного графа с профилем предпочтений вершин.

2.4.1 Математическая модель механизма отложенного принятия

Модель будет строится на основе рассмотренной ранее теории устойчивых паросочетаний.

1. Каждому мужчине $m \in M$ поставим в пару в соответствии с профилем P_m наиболее предпочтительную женщину $w_m^1 \in W$. Будем говорить, что $\mu_1(m) = w_m^1$, или что m сделал предложение w_m^1.

2. Перестроим μ_1 таким образом, чтобы данное отображение стало индивидуально-рациональным паросочетанием. Для этого, если по итогам первого шага, $w = \mu_1(m)$, но $wP_w m$, т.е. m недопустим для w, то установим $\mu_1(m) = m$, что фактически означает опцию «остаться в одиночестве».

3. На данном шаге μ_1 не является взаимно-однозначным отображением, так как одна и та же женщина w может быть поставлена в соответствие нескольким мужчинам m. Пусть $M_1(w) = \{m : \mu_1(m) = w\}$. Для каждой w, такой что $|M_1(w)| > 0$, выберем наиболее предпочтительного мужчину m_w^1 из подмножества $M_1(w)$ в соответствии с профилем предпочтения P_w. Создадим пару $\mu_1(w) = m_w^1$. Для остальных мужчин из подмножества $M_1(w)$ установим $\mu_1(m) = m$. Если изначально $M_1(w) = \emptyset$, то выставим женщине w $\mu_1(w) = w$, то есть опцию «остаться в одиночестве». Теперь по итогам данного шага построено временное паросочетание, являющееся взаимно-однозначным отображением и индивидуально-рациональным паросочетанием.

4. Будем строить новое временное паросочетание μ_2. Поставим вначале $\mu_2 = \mu_1$. Для каждого $m \in M$, такого что $\mu_1(m) = m$, найдем w_m^2, наиболее предпочтительную из множества $W \setminus \{w_m^1\}$ в соответствии с профилем предпочтений P_m, и установим $\mu_2(m) = w_m^2$. Если же такой возможности нет, то $\mu_2(m) = m$ и мужчина попадает в множество M_{alone}, и в дальнейшем мужчины из этого множества не участвуют в итерациях алгоритма.

5. Вновь перестроим паросочетание μ_2 в соответствии с шагом 3 алгоритма. Будем повторять шаги до тех пор, пока на некотором шаге n не окажется, что все $m \in M$, такие что $\mu_{n-1}(m) = m$, принадлежат множеству M_{alone}. Работа алгоритма останавливается, когда каждый мужчина либо поставлен в пару с некоторой женщиной, либо получил отказ от всех допустимых женщин [4].

Полученное паросочетание μ_n становится окончательным. Множества M и W конечны, поэтому алгоритм завершает свое выполнение за конечное число шагов, а именно итоговая асимптотика работы алгоритма составляет $O(N^2)$ в случае если размеры сторон равны N, так как на каждой итерации мужчина делает предложение очередной женщине, но всего может быть не более N^2 предложений.

2.4.2 Пример работы алгоритма отложенного принятия

Пусть предпочтения мужчин заданы следующим образом, как на рисунке 2.1.

Антон:	Маша	Лена	Нина	Катя	Оля
Борис:	Оля	Маша	Нина	Катя	Лена
Валерий:	Лена	Маша	Катя	Нина	Оля
Геннадий:	Оля	Лена	Маша	Катя	Нина
Денис:	Лена	Нина	Маша	Оля	Катя

Рисунок 2.1 – Профили предпочтений мужчин

Пусть предпочтения женщин заданы следующим образом, как на рисунке 2.2.

Катя:	Антон	Валерий	Геннадий	Денис	Борис
Лена:	Антон	Геннадий	Борис	Денис	Валерий
Маша:	Денис	Борис	Валерий	Антон	Геннадий
Нина:	Антон	Валерий	Борис	Геннадий	Денис
Оля:	Геннадий	Борис	Денис	Валерий	Антон

Рисунок 2.2 – Профили предпочтений женщин

На первом этапе каждый мужчина делает предложение женщине, находящейся на первом месте в списке предпочтений, а женщины соответственно принимают предложение самого предпочитаемого мужчины, как отображено на рисунке 2.3.

			Катя	Лена	Маша	Нина	Оля
Антон	→	Маша		~~Валерий~~	Антон		~~Борис~~
Борис	→	Оля		Денис			Геннадий
Валерий	→	Лена					
Геннадий	→	Оля					
Денис	→	Лена					

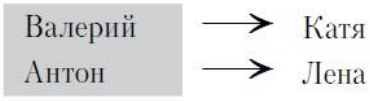
Рисунок 2.3 – Первый этап работы алгоритма

На следующем этапе получившие отказ мужчины, а именно Валерий и Борис, снова делают предложения следующим в списках женщинам, как показано на рисунке 2.4.

Катя	Лена	Маша	Нина	Оля
	Денис	~~Антон~~ ~~Валерий~~ Борис		Геннадий

Рисунок 2.4 – Предложения от получивших отказ на первом этапе мужчин

Далее снова мужчины без пары на данный момент делают предложения следующим по списку женщинам, как показано на рисунке 2.5.



Катя	Лена	Маша	Нина	Оля
Валерий	~~Денис~~ Антон	Борис		Геннадий

Рисунок 2.5 – Предложения мужчин без пар следующим в списках женщинам

И на последнем этапе, как показано на рисунке 2.6, заканчивается распределение.

Катя	Лена	Маша	Нина	Оля
Валерий	Антон	Борис	Денис	Геннадий

Рисунок 2.6 – Завершающий этап распределения

2.4.3 Теоремы теории устойчивых паросочетаний

Алгоритм отложенного принятия позволил вывести группу теорем теории устойчивых паросочетаний.

Теорема 2.1 Алгоритм отложенного принятия всегда позволяет найти устойчивое паросочетание [3, 5, 22], которое, однако, не является единственным.

В процессе поиска устойчивого паросочетания алгоритмом отложенного принятия одна сторона агентов является «предлагающей», а вторая – «принимающей». Устойчивое паросочетание является оптимальным для стороны, если каждый агент получает при распределении наилучшего возможного партнера.

Теорема 2.2 Алгоритм отложенного принятия находит устойчивое паросочетание, оптимальное для предлагающей стороны, при этом агенты принимающей стороны получают худшего из возможных партнеров [3, 4, 22].

Возникает вопрос, может ли принимающая сторона так исказить сообщенную информацию, чтобы улучшить свое положение. Такое искажение называется манипулированием. При использовании данного алгоритма манипулирование приводит к улучшению, например, при помощи сообщения неполного списка предпочтений принимающей стороны.

Рассмотрим пример. Пусть агенты сообщили свои истинные предпочтения, тогда при применении алгоритма с предлагающими мужчинами получаем следующее распределение, как показано в таблице 2.1.

Таблица 2.1 – Распределение с истинными предпочтениями

Albert	Diane	Emily	**Diane**	Bradley	Albert
Bradley	Emily	Diane	**Emily**	Albert	Bradley
Albert	Diane	Emily	**Diane**	Bradley	Albert
Bradley	Emily	Diane	**Emily**	Albert	Bradley

В случае если женщины сообщают неполный профиль предпочтений, они улучшают свое положение, как показано в таблице 2.2.

Таблица 2.2 – Манипулирование предпочтениями со стороны женщин

Albert	Diane	Emily	**Diane**	Bradley	⦸
Bradley	Emily	Diane	**Emily**	Albert	Bradley
Albert	Diane	Emily	**Diane**	Bradley	⦸
Bradley	Emily	Diane	**Emily**	Albert	Bradley

Механизм называется стратегически оптимальным, если сообщение истинных предпочтений является доминирующей стратегией для всех агентов. Позднее была доказана следующая теорема:

Теорема 2.3. Не существует механизма поиска устойчивого паросочетания, который являлся бы стратегически оптимальным для обеих сторон и при этом всегда приводил к стабильному результату [3–5, 22].

Далее возник вопрос, может ли получиться так, что в одном из устойчивых паросочетаний агент не нашел себе пару, в то время как в другом пара будет найдена. Была доказана следующая теорема:

Теорема 2.4. Множество агентов каждой из сторон, не определенных ни с кем в пару, остается одинаковым для всех устойчивых паросочетаний [3, 4, 22].

2.5 Модификация механизма отложенного принятия для задачи «один-ко-многим»

Модифицируем механизм отложенного принятия для модели распределения абитуриентов по факультетам.

Итак, $S = \{s_1, s_2, \dots, s_n\}$ – множество абитуриентов, $F = \{f_1, f_2, \dots, f_k\}$ – множество факультетов различных университетов, $Q = \{q_1, q_2, \dots, q_k\}$ – множество соответственных квот на прием абитуриентов различных факультетов.

У каждого абитуриента и у каждого факультета есть свой профиль предпочтений в соответствии с ограничениями, рассмотренными в модели. Рассматривается механизм с предлагающими абитуриентами.

Далее представлена модель модификации алгоритма.

1. Каждый абитуриент подает заявление на факультет, стоящий первым в списке предпочтений для абитуриента. Каждый факультет временно удерживает абитуриентов с самым высоким приоритетом в профиле предпочтений, ограничивая их количество размером квоты, и отказывает остальным.

2. Шаг с номером $t \geq 2$: Каждый абитуриент, которому было отказано на шаге с номером $(t-1)$, подает заявление на следующий по списку предпочтений факультет. Каждый факультет рассматривает как уже удерживаемых, так и новых абитуриентов. Каждый факультет временно удерживает абитуриентов с самым высоким приоритетом в профиле предпочтений, ограничивая их количество размером квоты, и отказывает остальным.

Алгоритм заканчивает работу, когда исчерпаны списки предпочтений абитуриентов. Так как списки имеют конечный размер, то алгоритм за конечное число итераций завершает свое выполнение. Точная оценка времени работы алгоритма зависит от деталей реализации, но она будет пропорциональна размеру списков предпочтений и сумме количества абитуриентов и количества факультетов. В оптимальной реализации хранение списка абитуриентов, зачисленных на текущий момент на факультет, осуществляется с помощью структуры данных, позволяющей извлекать элемент и добавлять новые элементы за логарифм от числа содержащихся в структуры элементов или быстрее, например, при помощи красного-черного дерева или хеш-таблицы, поэтому к оценке времени работы необходимо добавить поддержку данной структуры.

2.6 Алгоритм верхнего торгового цикла (Top Trading Cycle)

Алгоритм верхнего торгового цикла (Top Trading Cycle, или TTC) был разработан изначально для задачи о рынке недвижимости Дэвидом Гейлом и доработан Гербертом Скарфом и Ллойдом Шепли.

2.6.1 Математическая модель алгоритма верхнего торгового цикла

Основной алгоритм TTC иллюстрируется следующей ситуацией. В студенческих общежитиях живет n студентов. В общежитии n комнат. Каждый студент i живет в отдельной комнате h_i. У каждого студента есть строгие предпочтения относительно комнат, назначенных другим студентам, и своей собственной. Это может привести к взаимовыгодным обменам. Необходимо найти основное стабильное распределение так, чтобы все взаимовыгодные обмены были реализованы.

Далее представлена модель алгоритма верхнего торгового цикла.

1. Каждый студент i указывает на наиболее предпочтительную комнату, возможно, на свою собственную. Каждая комната указывает на своего владельца. Это создает ориентированный граф.

2. В данном графе необходимо найти циклы. Такие циклы существуют, и каждый студент находится не более чем в одном цикле.

3. Необходимо назначить каждому студенту в цикле комнату, на которую он указывает, и удалить его из графа вместе с назначенной комнатой.

4. Если остались нераспределенные студенты и комнаты, продолжить итерации алгоритма.

Алгоритм завершает выполнение за конечное число итераций, так как в графе конечное число вершин, а на каждой итерации вершины удаляются [23].

2.6.2 Пример работы алгоритма верхнего торгового цикла

Рассмотрим двудольный граф. В левой доле находятся студенты, в правой – комнаты. Студенты имеют профили предпочтений – комнаты отсортированы от лучшей к наименее предпочтительной слева направо. Проведем ребра в этом графе в соответствии с алгоритмом, как на рисунке 2.7.

$1: \quad (h_3, h_2, h_4, h_1)$
$2: \quad (h_4, h_1, h_2, h_3)$
$3: \quad (h_1, h_4, h_3, h_2)$
$4: \quad (h_3, h_2, h_1, h_4)$

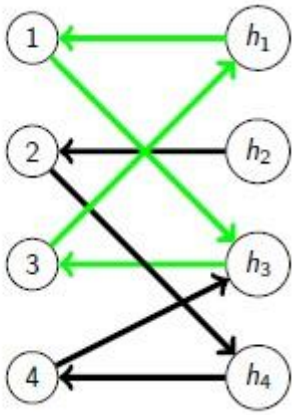


Рисунок 2.7 – Двудольный граф студентов и комнат

Выделим цикл $(1, h_3, 3, h_1, 1)$. Студентам 1 и 3 назначаются соответственно комнаты h_3 и h_1. Удалим эти вершины из графа и продолжим итерации, как показано на рисунке 2.8.

Студенты 2 и 4 остались со своими обновленными предпочтениями. Выделим цикл $(2, h_4, 4, h_2, 2)$. Студентам 2 и 4 назначаются соответственно комнаты h_4 и h_2. Удаляем эти вершины из графа, и граф становится пустым.

$2: \quad (h_4, h_2)$
$4: \quad (h_2, h_4)$

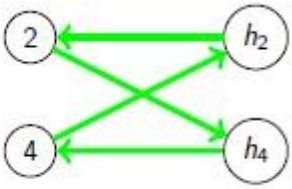


Рисунок 2.8 – Граф после удаления получивших назначение агентов

Алгоритм закончил свою работу, итоговое паросочетание: $(1, h_3), (2, h_4), (3, h_1), (4, h_2)$.

2.7 Модификация механизма TTC для задачи «один-ко-многим»

Модифицируем механизм отложенного принятия для модели распределения абитуриентов по факультетам.

Итак, $S = \{s_1, s_2, \dots, s_n\}$ – множество абитуриентов, $F = \{f_1, f_2, \dots, f_k\}$ – множество факультетов различных университетов, $Q = \{q_1, q_2, \dots, q_k\}$ – множество соответственных квот на прием абитуриентов различных факультетов.

У каждого абитуриента и у каждого факультета есть свой профиль предпочтений в соответствии с ограничениями, рассмотренными в модели. Рассматривается механизм с предлагающими абитуриентами.

Далее представлена модель модификации алгоритма.

1. Назначим счетчик c_j каждому факультету f_j, который будет показывать, сколько мест все еще свободны на этом факультете. Изначально эти счетчики равны соответственно квотам факультетов, т.е. $c_j = q_j$.

2. Каждый абитуриент s_i подает заявление на самый предпочтительный для него факультет f_j из профиля предпочтений P_{s_i}. Каждый факультет f_j указывает на абитуриента с наивысшем приоритетом в профиле предпочтений данного факультета P_{f_j}.

3. В данном графе необходимо найти циклы. Такие циклы существуют, и каждый студент находится не более чем в одном цикле.

4. Необходимо назначить каждому абитуриенту s_i в цикле факультет f_j, на который он указывает, и удалить его из графа. Счетчик c_j факультета необходимо уменьшить на 1. Если $c_j = 0$, то вершина, соответствующая факультету f_j, также удаляется из графа.

5. Если остались нераспределенные абитуриенты и места на факультетах, продолжить итерации алгоритма.

Алгоритм завершит свое выполнение за конечное число итераций, так как в графе конечное число вершин, а на каждой итерации вершины удаляются. Точная оценка времени работы алгоритма зависит от деталей реализации, но она будет пропорциональна размеру списков предпочтений и сумме количества абитуриентов и количества факультетов. В оптимальных реализациях необходимо эффективно хранить состояние двудольного графа и вершины, входящие в доли, что можно достигнуть при помощи красного-черного дерева или хеш-таблицы. Также в оценку времени необходимо добавить затраты на поиск циклов, а именно на каждой итерации необходимо выяснять, принадлежит ли абитуриент циклу, что является дорогостоящей операцией. Это существенно увеличивает время работы алгоритма.

2.8 Выводы по разделу

В данном разделе были проанализированы существующие на практике алгоритмы зачисления абитуриентов в вузы. Была поставлена задача о зачислении абитуриентов в вузы, являющаяся задачей поиска устойчивого паросочетания вида «один-ко-многим».

Были подробно рассмотрены существующие подходы поиска устойчивых паросочетаний для задачи вида «один-к-одному», такие как многокритериальный серийный диктаторский механизм (бостонский механизм серийного диктатора), алгоритм отложенного принятия и алгоритм верхнего торгового цикла. Данные алгоритмы были модифицированы для решения задачи вида «один-ко-многим», а именно задачи о зачислении абитуриентов в вузы, и

была проведена теоретическая оценка времени работы данных алгоритмов и стабильности получаемых решений.

В следующей главе будет проведен анализ эффективности алгоритмов на основе их программных реализаций.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ

3.1 Сравнительный анализ механизмов по времени работы

На основе программных реализаций алгоритмов поиска устойчивых паросочетаний проведем их сравнительный анализ по времени работы. Протестируем методы на различных сгенерированных тестовых наборах данных. Наборы данных зависят от количества поступающих абитуриентов, количества принимающих вузов, и константы K, означающей наибольшее количество учебных заведений, в которые может подать заявления абитуриент. В РФ на данный момент эта константа равна 5. Для всех групп зафиксированных параметров измерения проводились по три раза и в качестве результирующего значения было взято их среднее арифметическое. Результаты измерений и сравнительного анализа представлены в таблицах 3.1–3.5.

Таблица 3.1 – Сравнение алгоритмов поиска на тестах с 3 вузами и $K = 3$

Количество абитуриентов	Бостонский механизм серийного диктатора, мс	Алгоритм отложенного принятия (DA), мс	Алгоритм верхнего торгового цикла (TTC), мс
1000	1	1	258
5000	2	3	5060
10000	2	6	19570
15000	4	7	45434
20000	6	14	127165
25000	8	17	169433
30000	9	18	236412
50000	11	29	678761

Таблица 3.2 – Сравнение алгоритмов поиска на тестах с 10 вузами и $K = 3$

Количество абитуриентов	Бостонский механизм серийного диктатора, мс	Алгоритм отложенного принятия (DA), мс	Алгоритм верхнего торгового цикла (TTC), мс
1000	1	1	172
5000	1	3	3968
10000	2	5	15077
15000	4	10	36069
20000	5	11	76536
25000	9	16	124529
30000	10	17	175702
50000	11	28	522894

Таблица 3.3 – Сравнение алгоритмов поиска на тестах с 10 вузами и $K = 4$

Количество абитуриентов	Бостонский механизм серийного диктатора, мс	Алгоритм отложенного принятия (DA), мс	Алгоритм верхнего торгового цикла (TTC), мс
1000	1	1	174
5000	2	2	3870
10000	4	7	16454
15000	4	8	37135
20000	4	11	77368
25000	5	14	123492
30000	11	25	140692
50000	14	44	511839

На основе полученных измерений для различного количества вузов и константы K построим диаграммы зависимости времени работы методов от количества поступающих абитуриентов. Полученные графики и диаграммы представлены на рисунках 3.1 – 3.8.

Таблица 3.4 – Сравнение алгоритмов поиска на тестах с 100 вузами и $K = 4$

Количество абитуриентов	Бостонский механизм серийного диктатора, мс	Алгоритм отложенного принятия (DA), мс	Алгоритм верхнего торгового цикла (TTC), мс
1000	1	1	210
5000	2	4	4276
10000	3	5	16209
15000	6	16	40591
20000	7	16	73012
25000	8	17	115886
30000	8	18	164074
50000	11	29	482483

Таблица 3.5 – Сравнение алгоритмов поиска на тестах с 1000 вузами и $K = 5$

Количество абитуриентов	Бостонский механизм серийного диктатора, мс	Алгоритм отложенного принятия (DA), мс	Алгоритм верхнего торгового цикла (TTC), мс
1000	1	1	604
5000	2	2	14260
10000	4	6	44775
15000	6	9	98667
20000	9	13	144097
25000	9	15	283608
30000	12	21	296392
50000	20	43	795183

По статистике на 2019 год в РФ было примерно 780000 выпускников и 965 университетов, из выпускников далеко не все становятся абитуриентами вузов. Размер тестовых данных приближен к реальным данным с учетом разумности времени работы алгоритмов.

Рисунок 3.1 – Сравнение методов по времени работы при количестве вузов, равному 3, и K=3

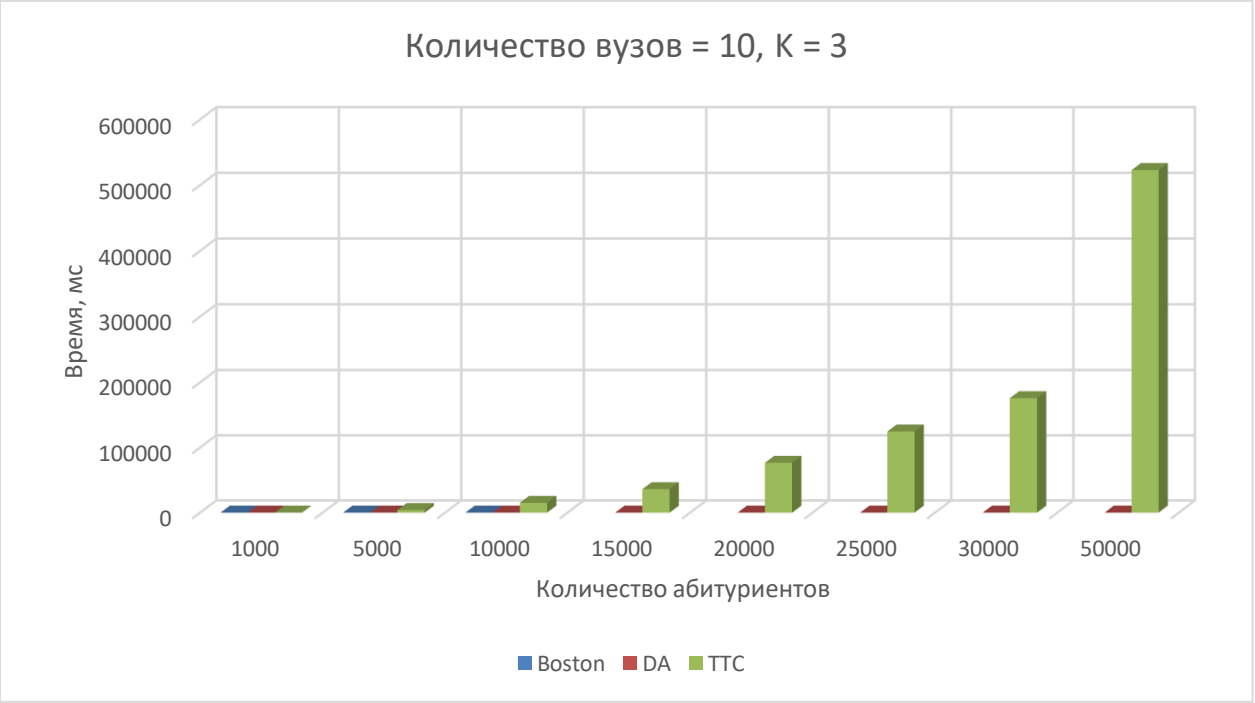


Рисунок 3.2 – Сравнение методов по времени работы при количестве вузов, равному 10, и K=3

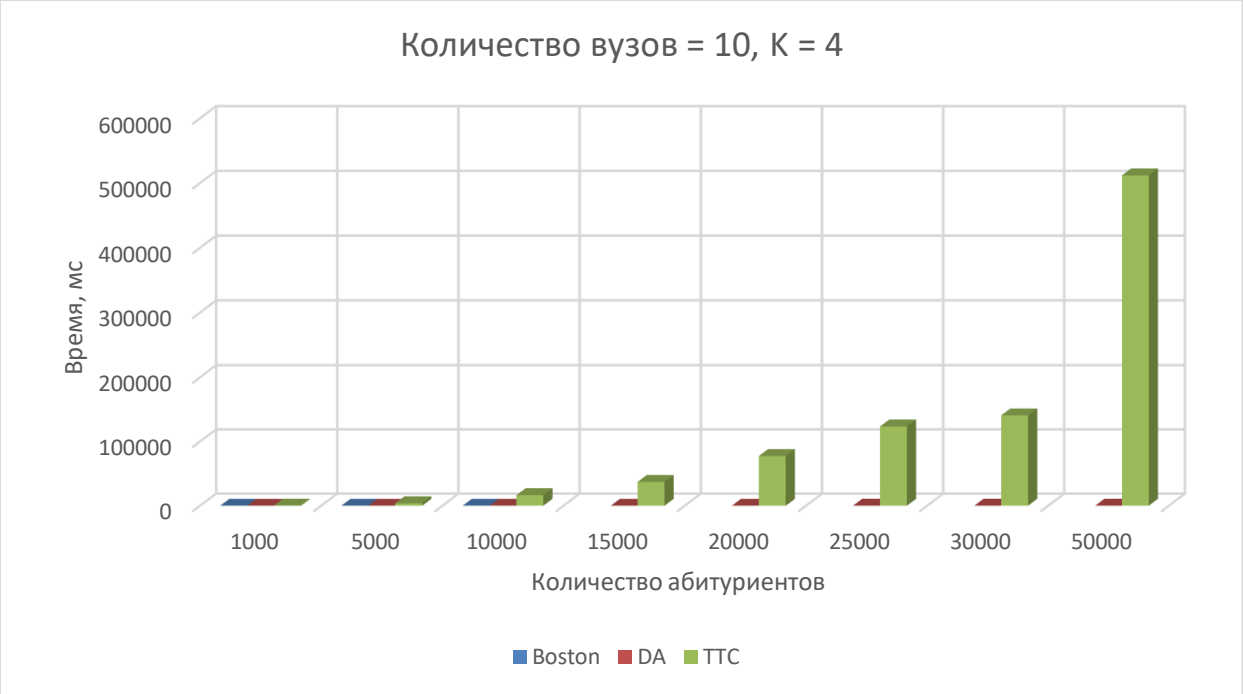


Рисунок 3.3 – Сравнение методов по времени работы при количестве вузов, равному 10, и K=4

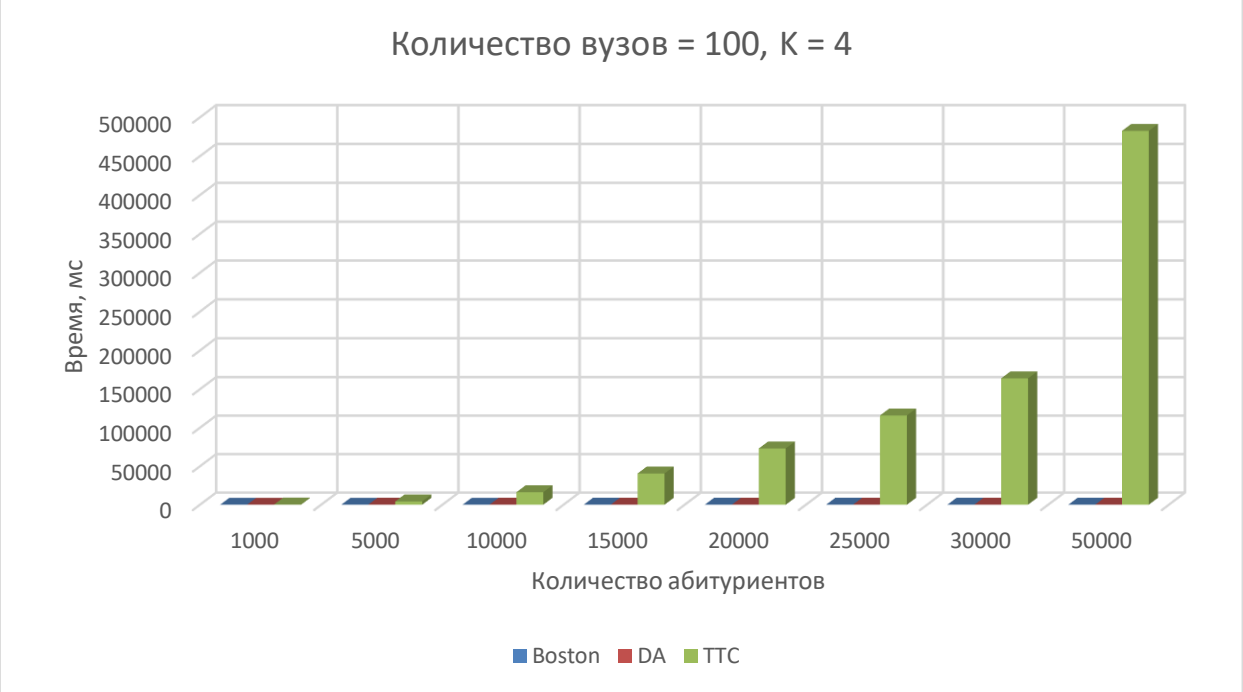


Рисунок 3.4 – Сравнение методов по времени работы при количестве вузов, равному 100, и K=4

Рисунок 3.5 – Сравнение методов по времени работы при количестве вузов, равному 1000, и K=5

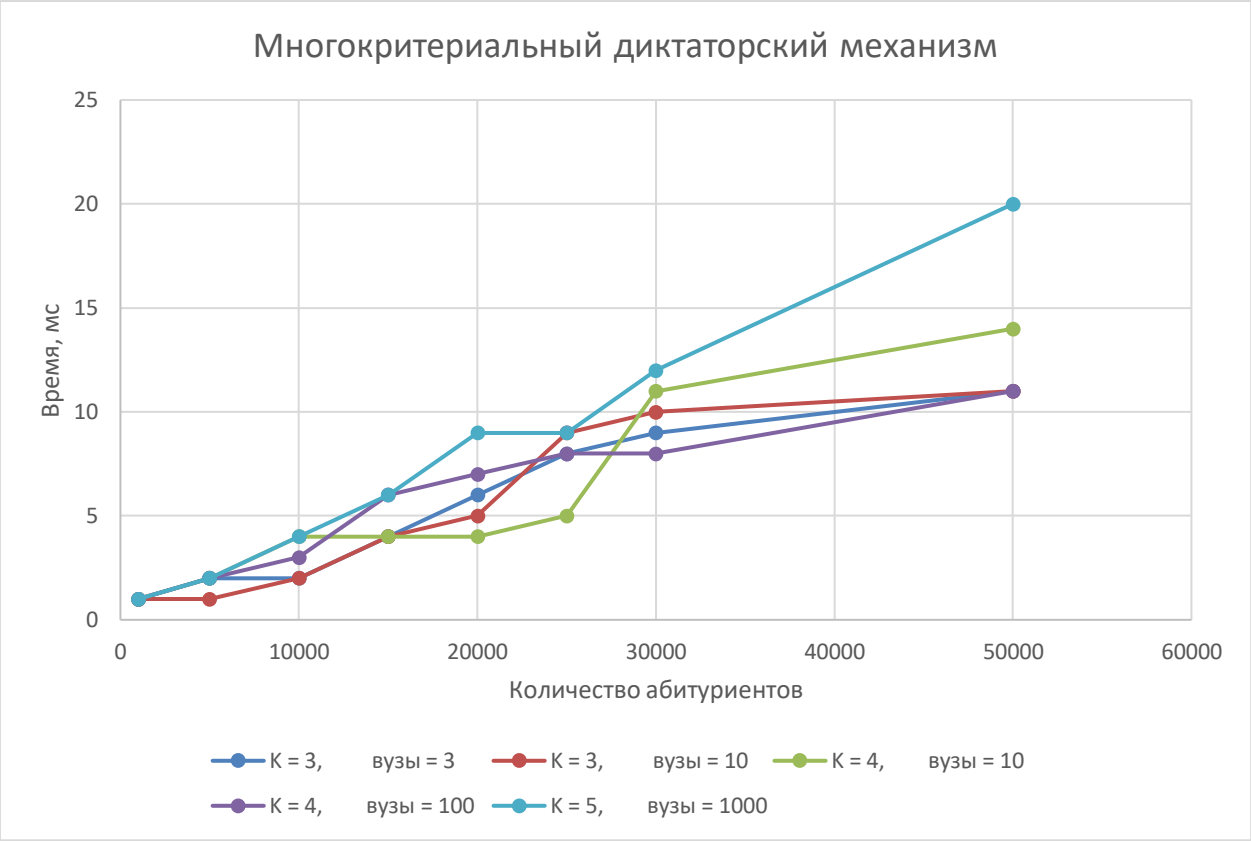


Рисунок 3.6 – Время работы многокритериального диктаторского механизма для различного количества вузов и значений константы K

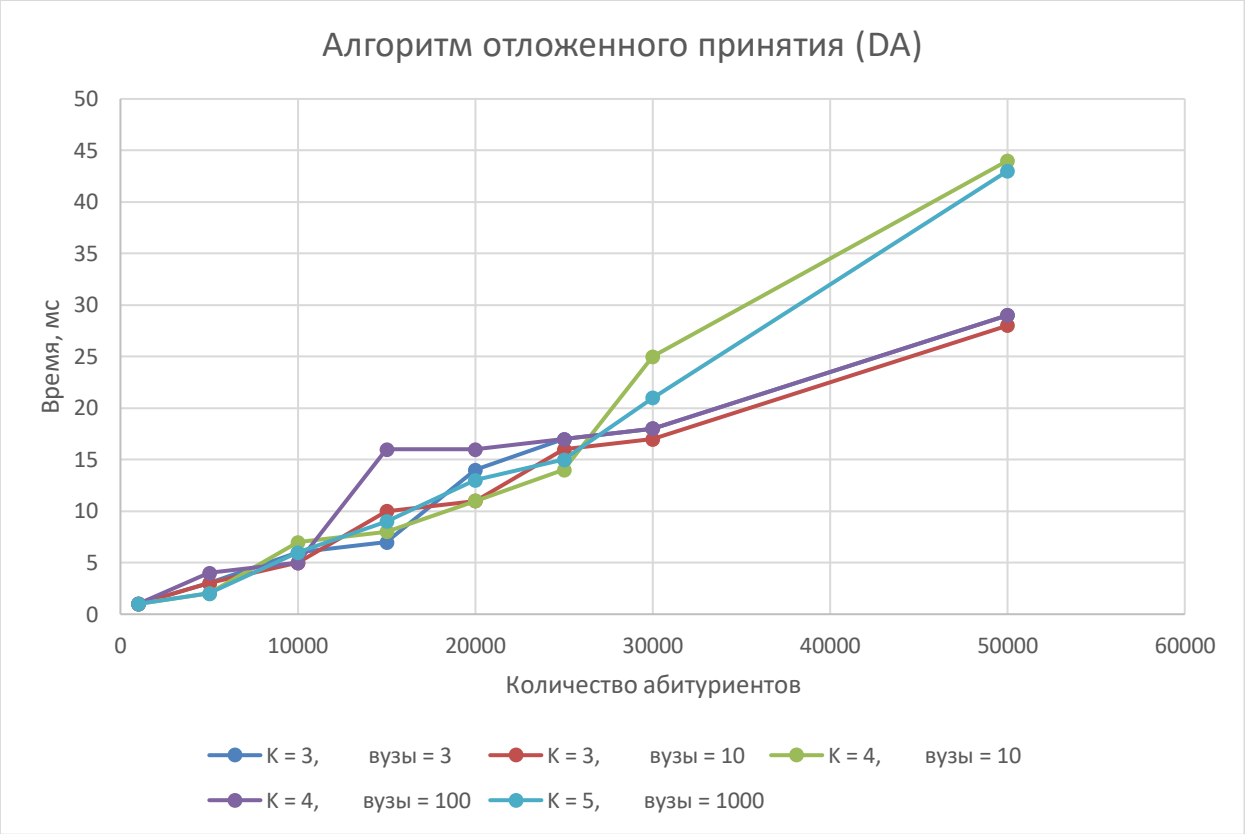


Рисунок 3.7 – Время работы алгоритма отложенного принятия для различного количества вузов и значений константы K

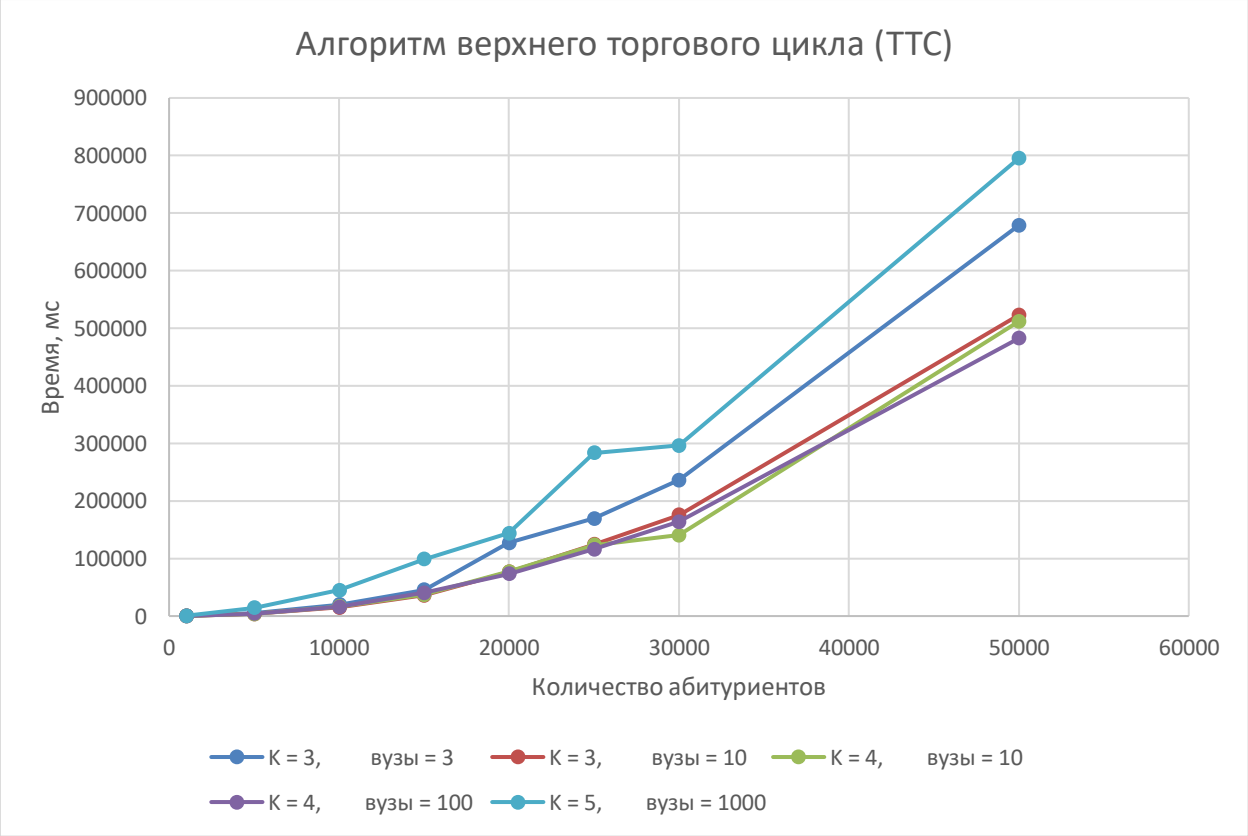


Рисунок 3.8 – Время работы алгоритма верхнего торгового цикла для различного количества вузов и значений константы K

По результатам измерений можно сделать следующие выводы. Время работы алгоритмов увеличивается в зависимости от увеличения количества абитуриентов, вузов и количества вузов K, в которые абитуриенты могут подавать заявления. Это подтверждает теоретические оценки времени работы алгоритмов.

Проведенное тестирование по времени работы показывает, что исследуемые алгоритмы серийного диктатора и отложенного принятия выдают результаты для данных реального размера за время, меньшее секунды, а алгоритм верхнего торгового цикла ввиду необходимости поиска циклов справляется за время, не превосходящее полчаса. В рамках проведения приемной кампании, длящейся длительный период, такое время более чем приемлемо.

Алгоритмы серийного диктатора и отложенного принятия демонстрируют высокую эффективность по времени работы, но алгоритм верхнего торгового цикла основан на инновационном подходе к задаче и выдает решения, удовлетворяющие большее число агентов рынка. Оценка коэффициента удовлетворенности агентов может послужить темой дальнейших исследований.

3.2 Сравнительный анализ механизмов по стабильности найденного решения

Для проведения сравнительного анализа механизмов по стабильности найденного решения была реализована функция проверки полученного паросочетания, подсчитывающая количество потенциальных блокирующих пар. Данная функция перебирает все возможные пары <студент, университет> и проверяет, может ли данная пара являться блокирующей, на основании определения блокирующей пары для паросочетаний вида «один-ко-многим». Решения, которые выдавали механизмы во время их тестирования для определения их времени работы, были проверены данной функций на наличие блокирующих пар. Алгоритмы выдают в качестве решения стабильные устойчивые паросочетания.

3.3 Демонстрация работы программы

В качестве демонстрации работы программы представим выведенные в консоль сгенерированные данные по абитуриентам и вузам в формате табличного вывода, как на рисунке 3.9. Данные по найденному устойчивому паросочетанию сохраняются в виде списка пар номеров назначенных друг другу абитуриентов и вузов.

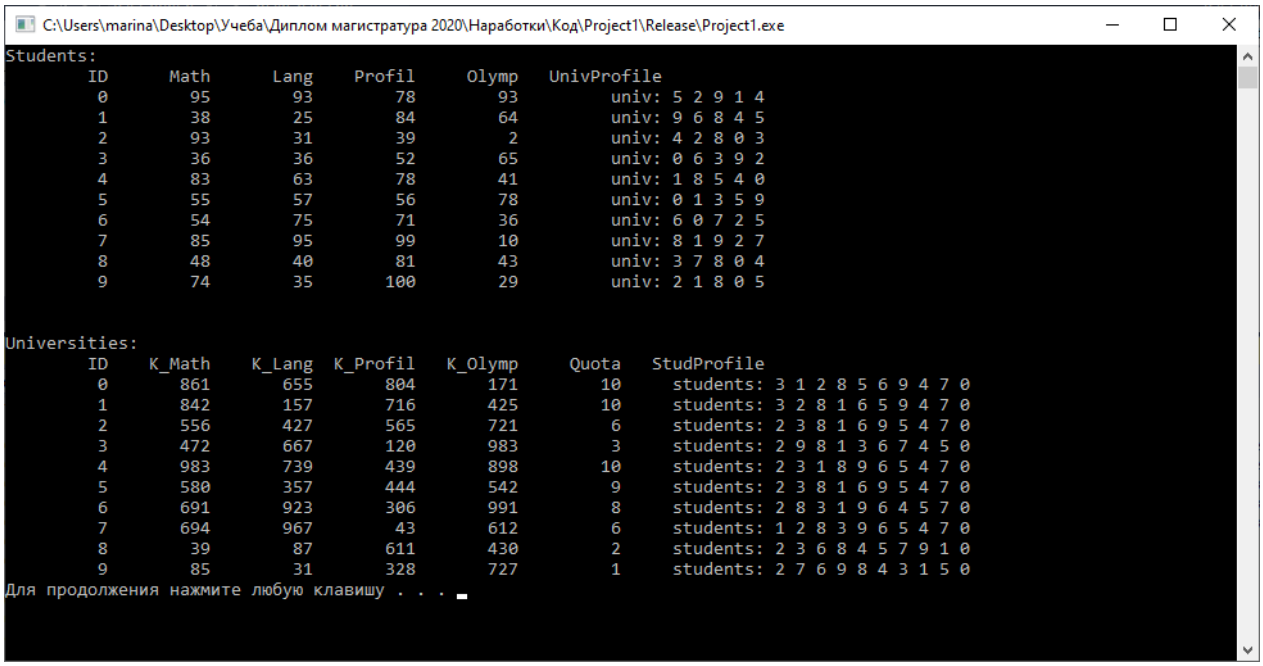


Рисунок 3.9 – Пример сгенерированных данных

3.4 Выводы по разделу

В разделе были проведены измерения времени работы методов поиска устойчивых паросочетаний на основе различных экспериментальных данных и построены графики зависимости времени работы от количества абитуриентов, вузов и количества вузов K, в которые абитуриенты могут подавать заявления. Была проведена проверка найденных алгоритмами паросочетаний на устойчивость.

ЗАКЛЮЧЕНИЕ

Данная работа посвящена исследованию устойчивых паросочетаний как способу эффективного распределения и сопоставления друг с другом агентов рынков с заданными предпочтениями.

В результате выполнения работы была осуществлена модификация существующих алгоритмов поиска устойчивых паросочетаний, таких как многокритериальный серийный диктаторский механизм, алгоритм отложенного принятия и алгоритм верхнего торгового цикла для задачи вида «один-ко-многим», проведен сравнительный анализ полученных модификаций по времени работы и осуществлена проверка найденных паросочетаний на устойчивость.

Для проведения сравнительного анализа модификации эффективных методов поиска устойчивых паросочетаний были реализованы на языке C++ в среде разработки Microsoft Visual Studio 2017. Для того чтобы измерить время работы методов, были зафиксированы такие параметры, как количество вузов и константа K, задающая количество заявлений, которые могут подать абитуриенты. Наибольшее количество вузов равно 1000, наибольшее значение константы K равно 5. Такие значения были взяты, чтобы максимально приблизить размеры сгенерированных данных к реальным данным приемной кампании в РФ. Для пяти комбинаций зафиксированных параметров было выбрано по 8 значений количества поступающих абитуриентов, для каждой зафиксированной тройки параметров количества абитуриентов, вузов и константы K было проведено по три теста времени и в качестве результата взято среднее арифметическое трех измерений.

Для проведения анализа полученных решений на устойчивость была реализована функция, принимающая построенное паросочетание и перебирающая все пары, состоящие из абитуриента и вуза, чтобы проверить, сколько пар являются блокирующими.

В ходе сравнительного анализа был сделан вывод, что предложенные модификации выдают стабильные устойчивые решения, и время нахождения этих решений приемлемо для применения в процессах централизованного распределения агентов рынка. Алгоритмы серийного диктатора и отложенного принятия демонстрируют высокую эффективность по времени работы, в то же время алгоритм верхнего торгового цикла ощутимо уступает им по времени, но предлагает более удовлетворительные для агентов рынка решения из-за инновационного подхода. Данные выводы подтверждают теоретические оценки. Эти свойства модификаций алгоритмов поиска устойчивых паросочетаний делают возможным их применение на различных рынках, таких как приемные кампании в вузы, для устранения проблемы децентрализованных рынков и перехода к централизации.

Использование исследованных механизмов позволяет облегчить процесс распределения абитуриентов по вузам и повысить эффективность конечного распределения. В частности, это позволяет повысить проходной балл и средний уровень абитуриентов на некоторых факультетах.

В дальнейшем планируется исследовать данные механизмы на степень удовлетворенности агентов рынка полученным распределением и предложить наиболее эффективный механизм для использования в приемных кампаниях российских вузов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Гиндикин, С.Г. Леонард Эйлер / Рассказы о физиках и математиках / С.Г. Гиндикин. – М.: МЦНМО, 2013. – С. 215–267.

2 Худокормов, А.Г. Нобелевские лауреаты по экономике в XXI веке: Сборник статей / А.Г. Худокормов, Ю.Я. Ольсевич, Я.А. Исламова. – ИНФРА-М, 2017. – 393 с.

3 Sonmez, T. Matching, Allocation and Exchange of Discrete Resources / T. Sonmez, U. Unver // Handbook of Social Economics, North-Holland, 2011. – Vol. 1A. – P. 781–852.

4 Кисельгоф, С.Г. Обобщенные паросочетания при предпочтениях, не являющихся линейными порядками / С.Г. Кисельгоф // Диссертация на соискание ученой степени под руководством Алексерова Ф.Т. – Москва, 2014. – 186 с.

5 Kesten, O. An Introduction to Matching, Assignment and Applications / O. Kesten // Higher School of Economics. – St. Petersburg, 2017. – 57 p.

6 Roth, A.E. The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory / A.E. Roth // Journal of Political Economy, 1984. – № 92. – P. 991–1016.

7 Peranson, E. The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design / E. Peranson, A.E. Roth // American Economic Review, 1999. – № 89(4). – P. 748–780.

8 Roth, A.E. Random Paths to Stability in Two-sided Matching / A.E. Roth, J.H. Vande Vate // Econometrica, 1990. – № 58(6). – P. 1475–1480.

9 Abdulkadiroglu, A. Strategy-proofness versus Efficiency in Matching with Indifferences: Redesigning the NYC High School Match / A. Abdulkadiroglu, P.A. Pathak, A.E. Roth // American Economic Review, 2009. – № 99 (5). – P. 1954–1978.

10 Abdulkadiroglu, A. The Boston Public School Match / A. Abdulkadiroglu, P.A. Pathak, A.E. Roth //American Economic Review: Papers and Proceedings, 2005. – № 95(2). – P. 368–371.

11 Roth, A.E. Kidney Exchange / A.E. Roth, T. Sonmez, U.M. Unver // Quarterly Journal of Economics, 2004. – № 119(2). – P. 457–488.

12 Roth, A.E. Pairwise Kidney Exchange / A.E. Roth, T. Sonmez, U.M. Unver // Journal of Economic Theory, 2005. – № 125(2). – P. 151–188.

13 Roth, A.E. A Kidney Exchange Clearinghouse in New England / A.E. Roth, T. Sonmez, U.M. Unver // American Economic Review: Papers and Proceedings, 2005. – № 95(2). – P. 376–380.

14 Niederle, M. The Gastroenterology Fellowship Match: How it Failed, and Why it Could Succeed Once Again / M. Niederle, A.E. Roth // Gastroenterology, 2004. – № 127. – P. 658–666.

15 Avery, C. The New Market for Federal Judicial Law Clerks / C. Avery, C. Jolls, R.A. Posner // University of Chicago Law Review, 2007. – № 74. – P. 447–486.

16 Haruvy, E. The Dynamics of Law Clerk Matching: An Experimental and Computational Investigation of Proposals for Reform of the Market / E. Haruvy, A.E. Roth, U.M. Unver // Journal of Economic Dynamic sand Control, 2006. – № 30(3). – P. 457–486.

17 Асанов, М.О. Дискретная математика. Графы, матроиды, алгоритмы / М.О. Асанов, В.А. Баранский, В.В. Расин. – СПб.: Изд-во «Лань», 2010. – 368 с.

18 Эвнин, А.Ю. Вокруг теоремы Холла / А.Ю. Эвнин. – 2-е изд., перераб. и доп. – М.: Книжный дом «ЛИБРОКОМ»,2011. – 88 с.

19 Эвнин, А.Ю. Элементы дискретной оптимизации / А.Ю. Эвнин. – Ч.: ЮУрГУ, 2012. – 91 с.

20 Алескеров, Ф.Т. Бинарные отношения, графы и коллективные решения / Ф.Т. Алексеров, Э.Л. Хабина, Д.А. Шварц. – 1-ое изд. М.: Изд. дом ГУ ВШЭ, 2007; 2-ое изд. М.: Физматлит, 2012.

21 Balinski, M.A. Tale of Two Mechanisms: Student Placement / M. Balinski, T. Sonmez // Journal of Economics Theory, 1999. – Vol. 84. – P. 73–94.

22 Dickerson, J.P. Stable Matching / J.P. Dickerson // Carnegie Mellon University, 2017. – 43 p.

23 Kojima, F. Matching and Market Design / F. Kojima // Yale University, 2009. – URL: http://sites.google.com/site/fuhitokojimaeconomics/

24 Железова, Е.Б. Теория и практика двусторонних рынков / Е.Б. Железова, С.Б. Измалков, С.К. Исаакович. – Вопросы экономики. – 2013. – №1. – 23 с.

25 Подвесовский, А.Г. Автоматизация распределения студентов по руководителям выпускных квалификационных работ с применением модели двустороннего матчинга / А.Г. Подвесовский, Д.Г. Лагерев, И.Г. Егорова. – Брянский государственный университет, г. Брянск, Россия: Современные информационные технологии и IT-образование. – 2017. – Том 13. – №4. – 11 с.

ПРИЛОЖЕНИЕ 1
Код программы

Файл DataGeneration.h

```
#pragma once
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

// Константа, отвечающая за количество вузов, в которые может подать документы
// абитуриент
const int K = 5;

// Структура, хранящая техническую информацию о студенте - баллы по ЕГЭ и за
// олимпиады, а так же профиль предпочтений студента относительно университетов.
// id - номер студента
// math_score - балл за ЕГЭ по математике
// lang_score - балл за ЕГЭ по русскому языку
// profil_score - балл за ЕГЭ по профильному предмету
// olymp_score - дополнительный балл за участие в олимпиадах по профилю
// univ_profile - вектор предпочтений студента
struct Student {
	int id;
	int math_score, lang_score, profil_score, olymp_score;
	vector<int> univ_profile;
};

// Структура, хранящая техническую информацию об университете - квота на
// поступление, коэффициенты важности каждого предмета для ранжировки
// абитуриентов, профиль предпочтений относительно абитуриентов,
// выстраиваемый на основе коэффициентов важности предметов, и рейтинг
// каждого абитуриента в соответствии с профилем предпочтений.
// quota - квота на поступление (максимальное количество зачисленных)
// koef_math, koef_lang, koef_profil, koef_olymp - коэффициенты
//            важности для рейтинга соответственно баллов по математике,
//            русскому языку, профильному предмету и участия в олимпиадах.
// profile_sudents - вектор предпочтений университета
// rating_for_students - хеш-таблица, сопоставляющая студенту его рейтинг.
struct University {
	int quota;
	int koef_math, koef_lang, koef_profil, koef_olymp;
	vector<int> profile_students;
	unordered_map<int, int> rating_for_student;
};

vector<Student> students;
vector<University> universities;

// Функция, возвращающая псевдослучайное число в отрезке [min, max],
// не превышающее константы RAND_MAX
int GetRandomNumber(int min, int max) {
	double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
	return static_cast<int>(rand() * fraction * (max - min + 1) + min);
}

// Функция, генерирующая список студентов заданного размера amount_stud.
```

```
// Баллы за ЕГЭ гененируются в разумных пределах поступающего абитуриента,
// профиль предпочтений по вузам генерируется случайным образом.
vector<Student> StudentsGenerate(int amount_stud, int amount_univ) {
	vector<Student> ans;
	ans.resize(amount_stud);
	vector<int> univs;
	univs.resize(amount_univ);
	for (int i = 0; i < amount_univ; i++)
		univs[i] = i;
	for (int i = 0; i < amount_stud; i++) {
		ans[i].id = i;
		ans[i].math_score = GetRandomNumber(30, 100);
		ans[i].lang_score = GetRandomNumber(25, 100);
		ans[i].profil_score = GetRandomNumber(35, 100);
		ans[i].olymp_score = GetRandomNumber(0, 100);
		random_shuffle(univs.begin(), univs.end());
		for (int j = 0; j < K; j++) {
			ans[i].univ_profile.push_back(univs[j]);
		}
	}
	return ans;
}

// Функция, генерирующая список университетов заданного размера amount_univ.
// Коэффициенты важности различных предметов для ранжирования абитуриентов
// генерируются в отрезке [1, 1000]; абитуриенты сортируются в соответствии с
// линейной комбинацией их баллов по ЕГЭ и соответствующих коэффициентов важности
// данного университета.
vector<University> UniversitiesGenerate(int amount_stud, int amount_univ, vector<Student>
stud) {
	vector<University> ans;
	ans.resize(amount_univ);
	for (int i = 0; i < amount_univ; i++) {
		int k_m = GetRandomNumber(1, 1000);
		int k_l = GetRandomNumber(1, 1000);
		int k_p = GetRandomNumber(1, 1000);
		int k_o = GetRandomNumber(1, 1000);
		ans[i].koef_math = k_m;
		ans[i].koef_lang = k_l;
		ans[i].koef_profil = k_p;
		ans[i].koef_olymp = k_o;
		ans[i].quota = GetRandomNumber(1, amount_stud);
		stable_sort(stud.begin(), stud.end(), [k_m, k_l, k_p, k_o] (const Student&
a, const Student& b) {
			int x = a.math_score * k_m + a.lang_score * k_l + a.profil_score *
k_p + a.olymp_score * k_o;
			int y = b.math_score * k_m + b.lang_score * k_l + b.profil_score *
k_p + b.olymp_score * k_o;
			return (x < y || (x == y && a.id < b.id));
		});
		for (int j = 0; j < amount_stud; j++) {
			ans[i].profile_students.push_back(stud[j].id);
			ans[i].rating_for_student[stud[j].id] = j;
		}
	}
	return ans;
}
```

Файл BostonDictatorship.h

```
#pragma once
#include <algorithm>
#include <deque>
```

```cpp
#include <vector>
#include <set>
#include <unordered_map>
#include "data_generation.h"
using namespace std;

// Функция, реализующая поиск устойчивого паросочетания многокритериальным серийным
// диктаторским механизмом. Функция принимает на вход списки данных о студентах и
// университетах и возвращает вектор пар <студент, университет>, входящих в
// итоговое устойчивое паросочетание.
vector<pair<int, int> > BostonDictatorship(vector<Student>& students, vector<University>&
universities) {
	vector<pair<int, int> > stable_matching;
	int amount_stud = static_cast<int>(students.size());
	int amount_univ = static_cast<int>(universities.size());
	vector<int> stud_match;
	stud_match.resize(amount_stud);
	stud_match.assign(amount_stud, -1);
	deque<int> q;
	for (int i = 0; i < amount_stud; i++)
		q.push_back(i);
	vector<int> univ_match_size;
	univ_match_size.resize(amount_univ);
	int pos_list = 0;
	while (!q.empty()) {
		deque<int> new_q;
		vector<vector<pair<int, int> > > temp_univ;
		temp_univ.resize(amount_univ);
		for (const auto& elem : q) {
			stud_match[elem] = students[elem].univ_profile[pos_list];
			int num_univ = stud_match[elem];
			temp_univ[num_univ].push_back({
universities[num_univ].rating_for_student[elem], elem });
		}
		for (int i = 0; i < amount_univ; i++) {
			sort(temp_univ[i].begin(), temp_univ[i].end());
			int t_temp = temp_univ[i].size();
			int t_rest = universities[i].quota - univ_match_size[i];
			univ_match_size[i] += min(t_temp, t_rest);
			if (t_temp > t_rest) {
				for (int j = t_rest; j < t_temp; j++) {
					stud_match[temp_univ[i][j].second] = -1;
				}
			}
		}
		pos_list++;
		for (const auto& elem : q) {
			if (stud_match[elem] == -1 && students[elem].univ_profile.size() >
pos_list) {
				new_q.push_back(elem);
			}
		}
		q.clear();
		for (const auto& elem : new_q) {
			q.push_back(elem);
		}
	}
	for (int i = 0; i < amount_stud; i++) {
		if (stud_match[i] != -1) {
			stable_matching.push_back({ i, stud_match[i] });
		}
	}
	return stable_matching;
}
```

Файл DeferredAcceptance.h

```
#pragma once
#include <deque>
#include <vector>
#include <set>
#include <unordered_map>
#include "data_generation.h"
using namespace std;

// Функция, реализующая поиск устойчивого паросочетания алгоритмом отложенного
// принятия. Функция принимает на вход списки данных о студентах и университетах
// и возвращает вектор пар <студент, университет>, входящих в итоговое устойчивое
// паросочетание.
vector<pair<int, int> > DeferredAcceptance(vector<Student>& students, vector<University>&
universities) {
	vector<pair<int, int> > stable_matching;
	int amount_stud = static_cast<int>(students.size());
	int amount_univ = static_cast<int>(universities.size());
	vector<int> stud_match;
	stud_match.resize(amount_stud);
	stud_match.assign(amount_stud, -1);
	deque<int> q;
	for (int i = 0; i < amount_stud; i++)
		q.push_back(i);
	vector<set<pair<int, int> > > univ_match;
	univ_match.resize(amount_univ);
	int pos_list = 0;
	while (!q.empty()) {
		deque<int> new_q;
		for (const auto& elem : q) {
			stud_match[elem] = students[elem].univ_profile[pos_list];
			int num_univ = stud_match[elem];
		}
		for (const auto& elem : q) {
			int num_univ = stud_match[elem];
			if (static_cast<int>(univ_match[num_univ].size()) <
universities[num_univ].quota) {
				univ_match[num_univ].insert({
universities[num_univ].rating_for_student[elem], elem });
			} else {
				auto it = univ_match[num_univ].end();
				it--;
				if (it->first >
universities[num_univ].rating_for_student[elem]) {
					stud_match[it->second] = -1;
					univ_match[num_univ].erase(it);
					univ_match[num_univ].insert({
universities[num_univ].rating_for_student[elem], elem });
				} else {
					stud_match[elem] = -1;
				}
			}
		}
		pos_list++;
		for (const auto& elem : q) {
			if (stud_match[elem] == -1 && students[elem].univ_profile.size() >
pos_list) {
				new_q.push_back(elem);
			}
		}
		q.clear();
```

```
                for (const auto& elem : new_q) {
                        q.push_back(elem);
                }
        }

        for (int i = 0; i < amount_stud; i++) {
                if (stud_match[i] != -1) {
                        stable_matching.push_back({ i, stud_match[i] });
                }
        }
        return stable_matching;
}
```

Файл TopTradingCycle.h

```
#pragma once
#pragma once
#include <algorithm>
#include <deque>
#include <vector>
#include <set>
#include <unordered_map>
#include <unordered_set>
#include "data_generation.h"
using namespace std;

// Функция, реализующая поиск устойчивого паросочетания алгоритмом верхнего торгового
// цикла. Функция принимает на вход списки данных о студентах и университетах и
// возвращает вектор пар <студент, университет>, входящих в итоговое устойчивое
// паросочетание.
vector<pair<int, int> > TopTradingCycle(vector<Student>& students, vector<University>&
universities) {
        vector<pair<int, int> > stable_matching;
        int amount_stud = static_cast<int>(students.size());
        int amount_univ = static_cast<int>(universities.size());
        vector<int> stud_match;
        stud_match.resize(amount_stud);
        stud_match.assign(amount_stud, -1);
        unordered_set<int> q_stud, q_univ;
        for (int i = 0; i < amount_stud; i++)
                q_stud.insert(i);
        for (int i = 0; i < amount_univ; i++)
                q_univ.insert(i);
        vector<int> univ_match_size;
        univ_match_size.resize(amount_univ);
        while (!(q_stud.empty() || q_univ.empty())) {
                unordered_map<int, int> temp_stud, temp_univ;

                for (const auto& elem : q_stud) {
                        for (int j = 0; j < K; j++) {
                                int num_univ = students[elem].univ_profile[j];
                                if (q_univ.find(num_univ) != q_univ.end()) {
                                        temp_stud[elem] = num_univ;
                                        break;
                                }
                        }
                }

                for (const auto& elem : q_univ) {
                        for (int j = 0; j < universities[elem].profile_students.size(); j++)
{
                                int num_stud = universities[elem].profile_students[j];
```

```
                        if (q_stud.find(num_stud) != q_stud.end()) {
                                temp_univ[elem] = num_stud;
                                break;
                        }
                }
        }

        for (const auto& elem : temp_stud) {
                if (q_stud.find(elem.first) != q_stud.end()) {
                        set<int> vertexes;
                        vertexes.insert(elem.first);
                        bool is_cycle = false;
                        int next = elem.second;
                        while (true) {
                                if (temp_univ.find(next) == temp_univ.end()) {
                                        break;
                                }
                                next = temp_univ[next];
                                if (next == elem.first) {
                                        is_cycle = true;
                                        break;
                                }
                                if (vertexes.find(next) != vertexes.end()) {
                                        break;
                                }
                                if (temp_stud.find(next) == temp_stud.end()) {
                                        break;
                                }
                                vertexes.insert(next);
                                next = temp_stud[next];
                        }
                        if (is_cycle) {
                                next = elem.first;
                                while (true) {
                                        stud_match[next] = temp_stud[next];
                                        univ_match_size[stud_match[next]]++;
                                        q_stud.erase(next);
                                        if (univ_match_size[stud_match[next]] ==
universities[stud_match[next]].quota) {
                                                q_univ.erase(stud_match[next]);
                                        }
                                        next = temp_univ[stud_match[next]];
                                        if (next == elem.first) {
                                                break;
                                        }
                                }
                        }
                }
        }

        for (const auto& elem : temp_univ) {
                if (univ_match_size[elem.first] == universities[elem.first].quota) {
                        q_univ.erase(elem.first);
                }
        }
        for (const auto& elem : temp_stud) {
                bool is_relevant = false;
                for (int j = 0; j < K; j++) {
                        int num_univ = students[elem.first].univ_profile[j];
                        if (q_univ.find(num_univ) != q_univ.end()) {
                                is_relevant = true;
                                break;
                        }
                }
                if (!is_relevant) {
```

```
                        q_stud.erase(elem.first);
                    }
                }
            }
        }
        for (int i = 0; i < amount_stud; i++) {
            if (stud_match[i] != -1) {
                stable_matching.push_back({ i, stud_match[i] });
            }
        }
        return stable_matching;
}
```

Файл StabilityAnalysis.h

```
#pragma once
#include <map>
#include <set>
#include <vector>
#include "data_generation.h"

// Функция, реализующая проверку полученного парочетания на устойчивость. Данная
// функция подсчитывает количество потенциальных блокирующих пар на основании
// определения блокирующей пары, профилей предпочтений студентов и вузов и полученного
// механизмом решения.
int AmountBlockingPairs(vector<Student>& students, vector<University>& universities,
                                    const vector<pair<int, int> >& matching) {
        int amount_stud = static_cast<int>(students.size());
        int amount_univ = static_cast<int>(universities.size());

        map<int, int> stud_match;
        map<int, set<int> > univ_match;
        for (const auto& elem : matching) {
            stud_match[elem.first] = elem.second;
            univ_match[elem.second].insert(elem.first);
        }

        int ans = 0;
        for (int stud = 0; stud < amount_stud; stud++) {
            for (int univ = 0; univ < amount_univ; univ++)
                if (stud_match.find(stud) != stud_match.end() && stud_match[stud] !=
univ) {
                    int pos_univ_in_stud = -1, pos_stud_in_univ = -1;
                    int matched_univ_for_stud = -1;
                    for (int i = 0; i < students[stud].univ_profile.size(); i++) {
                        if (students[stud].univ_profile[i] == univ) {
                            pos_univ_in_stud = i;
                        }
                        if (students[stud].univ_profile[i] == stud_match[stud])
{
                            matched_univ_for_stud = i;
                        }
                    }
                    for (int i = 0; i <
universities[univ].profile_students.size(); i++) {
                        if (universities[univ].profile_students[i] == stud) {
                            pos_stud_in_univ = i;
                            break;
                        }
                    }
                    if (pos_stud_in_univ > -1 && pos_univ_in_stud > -1) {
                        if (pos_univ_in_stud < matched_univ_for_stud) {
                            bool flag = false;
                            for (const auto& elem : univ_match[univ]) {
```

```
                                                if
(universities[univ].rating_for_student[elem] > pos_stud_in_univ) {
                                                        flag = true;
                                                        break;
                                                }
                                        }
                                        if (flag || univ_match[univ].size() <
universities[univ].quota) {
                                                ans++;
                                        }
                                }
                        }
                }
        }
        return ans;
}
```

Файл DataPrint.h

```
#pragma once
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <vector>
#include "data_generation.h"
using namespace std;

// Функция, выводящая в консоль итоговое паросочетание
void PrintAns(const vector<pair<int, int> >& ans) {
        for (const auto& elem : ans) {
                cout << elem.first << " " << elem.second << "\n";
        }
        return;
}

// Функция, выводящая в консоль сгенерированные данные абитуриентов и вузов в
отформатированном виде.
void PrintData(int amount_stud, int amount_univ) {
        cout << "Students:\n";
        cout.width(10);
        cout.fill(' ');
        cout << "ID";
        cout.width(10);
        cout.fill(' ');
        cout << "Math";
        cout.width(10);
        cout.fill(' ');
        cout << "Lang";
        cout.width(10);
        cout.fill(' ');
        cout << "Profil";
        cout.width(10);
        cout.fill(' ');
        cout << "Olymp";
        cout.width(15);
        cout.fill(' ');
        cout << "UnivProfile\n";
        for (int i = 0; i < amount_stud; i++) {
                cout.width(10);
                cout.fill(' ');
                cout << students[i].id;
                cout.width(10);
```

```
        cout.fill(' ');
        cout << students[i].math_score;
        cout.width(10);
        cout.fill(' ');
        cout << students[i].lang_score;
        cout.width(10);
        cout.fill(' ');
        cout << students[i].profil_score;
        cout.width(10);
        cout.fill(' ');
        cout << students[i].olymp_score;
        cout.width(15);
        cout.fill(' ');
        cout << "univ: ";
        for (int j = 0; j < K; j++) {
               cout << students[i].univ_profile[j] << " ";
        }
        cout << "\n";
}
cout << "\n\n";
cout << "Universities:\n";
cout.width(10);
cout.fill(' ');
cout << "ID";
cout.width(10);
cout.fill(' ');
cout << "K_Math";
cout.width(10);
cout.fill(' ');
cout << "K_Lang";
cout.width(10);
cout.fill(' ');
cout << "K_Profil";
cout.width(10);
cout.fill(' ');
cout << "K_Olymp";
cout.width(10);
cout.fill(' ');
cout << "Quota";
cout.width(15);
cout.fill(' ');
cout << "StudProfile\n";
for (int i = 0; i < amount_univ; i++) {
        cout.width(10);
        cout.fill(' ');
        cout << i;
        cout.width(10);
        cout.fill(' ');
        cout << universities[i].koef_math;
        cout.width(10);
        cout.fill(' ');
        cout << universities[i].koef_lang;
        cout.width(10);
        cout.fill(' ');
        cout << universities[i].koef_profil;
        cout.width(10);
        cout.fill(' ');
        cout << universities[i].koef_olymp;
        cout.width(10);
        cout.fill(' ');
        cout << universities[i].quota;
        cout.width(15);
        cout.fill(' ');
        cout << "students: ";
        for (int j = 0; j < amount_stud; j++) {
```

```
                        cout << universities[i].profile_students[j] << " ";
                }
                cout << "\n";
        }
}

// Функция, выводящая в консоль сгенерированные данные абитуриентов и вузов в тестовом
виде.
void PrintData(int amount_stud, int amount_univ) {
        cout << "Students:\n";
        for (int i = 0; i < amount_stud; i++) {
                cout << students[i].id << " " << students[i].math_score << " " <<
students[i].lang_score << " ";
                cout << students[i].profil_score << " " << students[i].olymp_score << "
univ: ";
                for (int j = 0; j < K; j++) {
                        cout << students[i].univ_profile[j] << " ";
                }
                cout << "\n";
        }
        cout << "\n";
        cout << "Univs:\n";
        for (int i = 0; i < amount_univ; i++) {
                cout << universities[i].koef_math << " " << universities[i].koef_lang << "
";
                cout << universities[i].koef_profil << " " << universities[i].koef_olymp <<
" ";
                cout << universities[i].quota << "\n";
                for (int j = 0; j < amount_stud; j++) {
                        cout << universities[i].profile_students[j] << " ";
                }
                cout << "\n";
        }
}
```

Файл MainPart.cpp

```
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <vector>
#include "boston_dictatorship.h"
#include "data_generation.h"
#include "data_print.h"
#include "deferred_acceptance.h"
#include "stability_analysis.h"
#include "top_trading_cycle.h"
using namespace std;

int main() {
        srand(time(0));

        students.resize(8);
        students[0].univ_profile = { 0, 1, 2 };
        students[1].univ_profile = { 0, 1, 2 };
        students[2].univ_profile = { 2, 1, 0 };
        students[3].univ_profile = { 2, 1, 0 };
        students[4].univ_profile = { 1, 2, 0 };
        students[5].univ_profile = { 1, 0, 2 };
        students[6].univ_profile = { 2, 0, 1 };
        students[7].univ_profile = { 1, 0, 2 };
```

```
	universities.resize(3);
	universities[0].profile_students = { 1, 2, 4, 7, 0, 3, 6, 5 };
	for (int i = 0; i < 8; i++)
		universities[0].rating_for_student[universities[0].profile_students[i]] =
i;
	universities[1].profile_students = { 1, 7, 4, 3, 0, 2, 6, 5 };
	for (int i = 0; i < 8; i++)
		universities[1].rating_for_student[universities[1].profile_students[i]] =
i;
	universities[2].profile_students = { 6, 5, 4, 2, 1, 3, 0, 7 };
	for (int i = 0; i < 8; i++)
		universities[2].rating_for_student[universities[2].profile_students[i]] =
i;
	universities[0].quota = 2;
	universities[1].quota = 2;
	universities[2].quota = 2;

	vector<pair<int, int> > stable_matching1 = BostonDictatorship(students,
universities);
	PrintAns(stable_matching1);
	cout << "\n\n";

	vector<pair<int, int> > stable_matching2 = DeferredAcceptance(students,
universities);
	PrintAns(stable_matching2);
	cout << "\n\n";

	vector<pair<int, int> > stable_matching3 = TopTradingCycle(students,
universities);
	PrintAns(stable_matching3);
	cout << "\n\n";

	int amount_stud = 10000;
	int amount_univ = 100;
	students = StudentsGenerate(amount_stud, amount_univ);
	universities = UniversitiesGenerate(amount_stud, amount_univ, students);
	PrintData(amount_stud, amount_univ);

	vector<pair<int, int> > ans;

	clock_t t_bg = clock();
	ans = BostonDictatorship(students, universities);
	double time_ans = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
	cout << fixed << setprecision(10) << time_ans << "\n";
	PrintAns(ans);
	cout << "\n\n";

	t_bg = clock();
	ans = DeferredAcceptance(students, universities);
	time_ans = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
	cout << fixed << setprecision(10) << time_ans << "\n";
	PrintAns(ans);
	cout << "\n\n";

	t_bg = clock();
	ans = TopTradingCycle(students, universities);
	time_ans = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
	cout << fixed << setprecision(10) << time_ans << "\n";
	PrintAns(ans);
	cout << "\n\n";

	ans = BostonDictatorship(students, universities);
	cout << AmountBlockingPairs(students, universities, ans) << "\n";
	PrintAns(ans);
	cout << "\n\n";
```

```
        ans = DeferredAcceptance(students, universities);
        cout << AmountBlockingPairs(students, universities, ans) << "\n";
        PrintAns(ans);
        cout << "\n\n";

        ans = TopTradingCycle(students, universities);
        cout << AmountBlockingPairs(students, universities, ans) << "\n";
        PrintAns(ans);
        cout << "\n\n";

        system("pause");
        return 0;
}
```