ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ
ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Отчет о прохождении стационарной производственной практики (научно-исследовательской работы на тему: «Разработка системы составления предварительного расписания сессии»)

Суховой Дарьи Викторовны, гр. 3530903/70302

Направление подготовки: 09.03.03 Прикладная информатика

Место прохождения практики: СПбПУ, ИКНТ, ВШИСиСТ

(указывается наименование профильной организации или наименование структурного подразделения

ФГАОУ ВО «СПбПУ», фактический адрес)

Сроки практики: с 25.01.21 по 13.04.21

Руководитель практики от ФГАОУ ВО «СПбПУ»:

Щукин А.В., к.т.н., доцент

(Ф.И.О., уч. степень, должность)

Консультант практики от профильной организации:

Пархоменко В.А., ассистент ВШИСиСТ

(Ф.И.О., должность)

Оценка:

Руководитель практики
от ФГАОУ ВО «СПбПУ» Щукин А.В.

Консультант практики
от ФГАОУ ВО «СПбПУ» Пархоменко В.А.

Обучающийся Сухова Д.В.

Дата:

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Составление расписания экзаменационных сессий сейчас является сложным и кропотливым процессом, который требует автоматизации. Создание программы, которая поможет сотрудникам университета с распределением экзаменов по датам, времени и аудиториям с учётом пожеланий преподавателей и университетского расписания, сможет значительно ускорить процесс составления расписания, а также устранит противоречия связанные с неустановленными формами сбора сведений.

Сложность составления распсиания сессии заключается в ручном переборе вариантов расстановки аттестаций и сверкой их с семестровым расписанием во избежание накладок с занятостью кабинетов и преподавателей.

Целью данной работы является исследование алгоритмов для составления расписания сессии и проектирование сервиса составления предварительного расписания сессии СПбПУ. Для достижения этой цели, в необходимо будет решить следующие **задачи**:

- Провести анализ существующих систем составления расписания.
- Рассмотреть задачау составления расписания сессии с алгоритмической точки зрения, исследовать алгоритмы для составления расписания сессии и оптимизации этого процесса.
- Написать сервис для получения данных о расписании с использованием API ruz.spbstu.ru
- Реализовать алгоритм составления предвариательного расписания сессии программно.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЗОР СИСТЕМ СОСТАВЛЕНИЯ РАСПИСАНИЯ

Проблема составления расписания не нова, и существует множество средств, призванных упростить её решение. В интернете можно найти онлайн-календари с возможностью совместного редактирования, системы управления бизнес-процессами и программы генерации расписания для разного рода предприятий. Но далеко не все из них могут быть использованы в качестве полноценной системы составления распиания сессии для университета, поэтому в параграфе 1.1 рассматриваются требования к системе составления расписания сессии СПбПу.

На рынке существует отдельная ниша систем составления расписания для ВУЗов, и в параграфе 1.2 рассматриваются её русскоязычные представители, а в параграфе 1.3 - зарубежные. Далее приведено сравнение таких систем и анализ их преимуществ и недостатков для решения конкретной задачи - составления предварительного расписания сессии в университете.

1.1. Требования к системе составления расписания сессии

1.1.1. Учёт семестрового расписания СПбПу

Важным аспектом системы составления расписания сессии СПбПу является учёт занятости аудиторий и преподавателей. Бывают ситуации, когда сессия для некоторых учёбных групп начинается в тот момент, когда у других групп всё ещё проводятся семестровые занятия. Чтобы иметь возможность составлять расписание сессии, которое не ставит проведение аттестаций в занятые по семестровому расписанию аудитории, необходимо обращаться с помощью API к официальному расписанию занятий [13]. Реализация модуля обращения к API ruz.spbstu.ru для доступа к расписанию представлена в приложении 3.

1.1.2. Роли пользователей системы

Одним из требований к системе сбора сведений является выделение двух ролей пользователей:

– Администратор - пользователь, который сообщает системе сведения о сессии, аудиториях и о плане экзаменов.

– Преподаватель - пользователь, который сообщает системе о собственных пожеланиях к проведению своих экзаменов.

Рассмотрим возможности этих видов пользователей подробнее. На диаграмме на рисунке 1.1 показаны возможности преподавателя.

Преподаватель может поучаствовать в составлении расписания, указав даты, дни недели и время, когда он доступен для проведения экзаменов или зачётов, так же указав какого типа аудитории нужны.

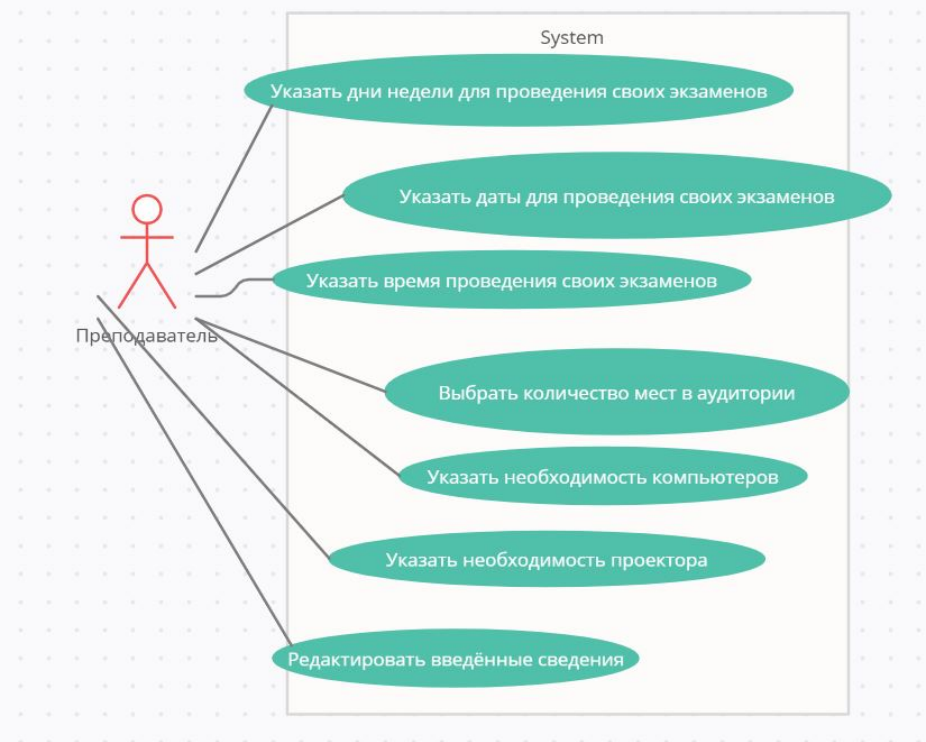

Рис.1.1. Use case диаграмма ППС

Возможности администратора показаны на диаграмме на рисунке 1.2.

Администратор заполняет общие сведения о сессии, такие как даты, время, доступные аудитории с указанием, сколько в них мест и есть ли там проектор и компьютеры и предстоящие экзамены по группам и преподавателям.

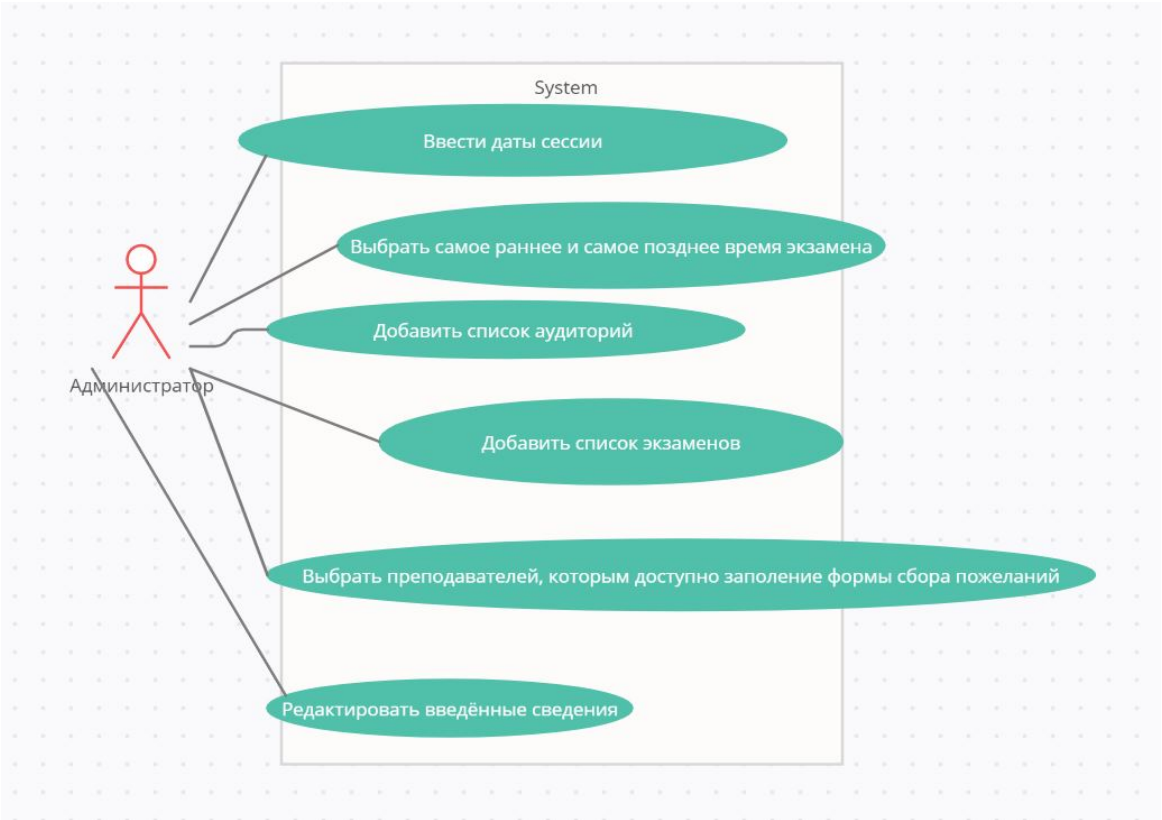

Рис.1.2. Use case диаграмма администратора

Таким образом, когда администратор заполнит все вышеперечисленные поля формы, в системе уже будет минимальный набор данных, необходимый для составления расписания. Далее, преподаватели могут заполнять свои формы с пожеланиями к своему расписанию. Незаполненная преподавателем форма не будет являться проблемой для системы. Пустая форма по умолчанию приравнивается к готовности преподавателя проводить экзамены и зачёты в любой день, в любое время.

1.1.3. Модель входных данных

Исходя из данных, котрые необходимо учитывать при составлении расписания сессии была спроектирована модель данных, соответсвующая схеме базы данных для их хранения. Она представлена на схеме 1.3.

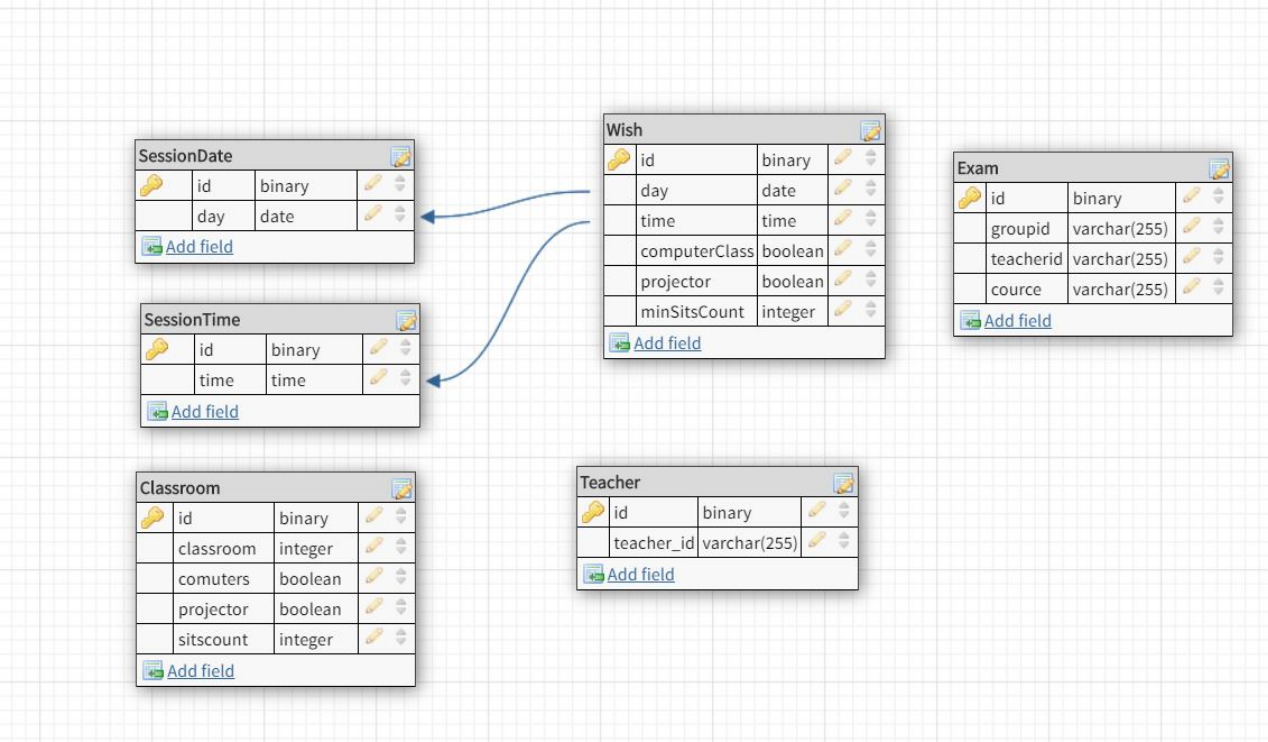

Рис.1.3. Схема базы данных

Сущности схемы:

- Classroom - таблица аудиторий с указанием есть ли там компьютеры или проектор и количеством мест.
    - id - уникальный идетификатор
    - classroom - номер кабинета
    - computers - наличие компьютеров
    - projector - наличие проектора
    - sitscount - кол-во мест в аудитории
- Exam - таблица запланированных на сессию экзаменов.
    - id - уникальный идетификатор
    - groupid - номер учебной группы
    - teacher - логин преподавателя
    - course - код дисциплины
- SessionDates - таблица со всеми доступными для экзамена датами. Такой способ хранения данных о датах сессии был выбран вместо хранения даты начала и окончания, потому что зимняя сессия может состоять из нескольких интервалов, прерванных новогодними праздниками.
    - id - уникальный идетификатор
    - day - дата

- SessionTime - таблица доступного для экзаменов времени.
    - id - уникальный идетификатор
    - time - время начала экзамена
- Wish - таблица пожеланий преподавателей.
    - id - уникальный идетификатор
    - teacherid - логин преподавателя
    - time - время
    - day - дата
    - computerClass - наличие компьютеров
    - projector - наличие проектора
    - minSitsCount - кол-во мест в аудитории
- Teacher - таблица преподавателей, которым доступно редактирование формы пожеланий.
    - id - уникальный идетификатор
    - teacherid - логин преподавателя

1.2. Обзор российских систем составления расписания

1.2.1. 1С: ХроноГраф Расписание

Фирма «1С», занимающаяся разработкой ПО для бизнеса и образования в качестве системы для автоматизации учебного планирования и составления расписания в разного рода организациях предлагает свою программу «1С: ХроноГраф Расписание» [1]. «1С: ХроноГраф Расписание» позволяет:

- составлять понедельное расписание организации или отдельных её подразделений;
- задавать периоды обучения с учётом нерабочих дней, каникул и разбиением на четные и нечётные недели;
- создавать черновое расписание, используя функцию «Предварительный расчёт».

Основной проблемой данной программы является несовместимость с другими платформами. «1С: ХроноГраф Расписание» - однопользовательская программа, и нельзя интегрировать её с web-приложением для возможности сбора данных напрямую от пользователей. Сложность составления расписания сессии в этой

системе обуславливается также её ориентированностью на составление расписания по неделям без учёта специфики проведения аттестаций.

1.2.2. Avtor (АВТОРасписание)

Программа «АВТОРасписание» [11] имеет несколько версий для различных учебных заведений: общеобразовательных школ, колледжей, техникумов, профессиональных училищ и ВУЗов. Это позволяет в подстроиться под специфику расписания конкретного типа образовательного учреждения, что является одним из её конкрурентых преимуществ.

«АВТОРасписание» имеет достаточно широкий спектр применений. Этот программный продукт позволяет

- составлять понедельное расписание для учебных групп с минимальным количеством окон;
- составлять расписание преподавателей с минимальным количеством окон;
- оптимально размещать занятия по аудиториям, учитывая их вместимость и оснащённость необходимым оборудованием;
- учитывать пожелания сотрудников к своему расписанию;
- разделять учебные группы на подгруппы;
- вносить ручные корректировки в расписание.

Преимуществом этой программы помимо прочего является возможность публиковать расписание обучающихся и преподавателей из самой системы «Автор» на сайте, внутреннем портале или на мультимедийных стендах образовательной организации. Но при этом импорт данных всё ещё производится вручную диспетчером, что не очень удобно для учебного заведения с большим штабом сотрудников, которые сами могли бы вносить свои пожелания в систему.

1.2.3. Галактика Расписание учебных занятий

«Галактика Расписание учебных занятий» - часть системы управления ВУЗом той же корпорации Галактика [12]. Этот программный продукт позволяет составлять расписание в ВУЗе, а также:

- вычислять несколько десятков показателей эффективности расписаний;
- оптимально размещать занятия по аудиториям, учитывая их вместимость и оснащённость необходимым оборудованием;
- учитывать приоритет преподавателей, учебных групп и дисциплин;

- контролировать пересечение расписаний для преподавателей, учебных групп и подгрупп во избежание «накладок»;
- контролировать длительность занятий;
- вручную бронировать аудиторный фонд;
- учитывать план изучения дисциплин для выстраивания их в правильном порядке.

«Галактика Расписание учебных занятий» - серьёзный инструмент для формирования расписания в высших учебных заведениях, учитывающий множество факторов при его составлении и имеющий удобную систему отчётности. На данный момент эта программа наиболее полно решает проблему автоматической генерации расписания российских ВУЗов, но и она не имеет интерфейса для прямого импорта пожеланий преподавателей прямо в систему. Компания «Галактика» помимо прочего предлагает техническое сопровождение своего ПО, но это учитывается при расчёте стоиости лецензии на использование программы.

1.3. Обзор зарубежных систем составления расписания

1.3.1. Apereo UniTime

UniTime от компании Apereo [2] - система автоматического создания расписания западных высших учебных заведений. Она учитывает, что студенты могут выбирвать себе индивидуальный набор курсов, чего не происходит в российских ВУЗах, где обучение происходит по плану образовательных программ направлений.

UniTime даёт возможность:

- автоматически генерировать расписание курсов и экзаменов;
- минимизировать конфликты студенческих курсов;
- вносить ручные корректировки в расписание.

Эта программа имеет понятный web-интерфейс и может быть интегрирована в другую систему, но она не позволяет преподавателям вносить данные о своей занятости, чтобы учесть их при составлении расписания. Неприспособленность программы под составление расписания для групп, а не для конкретных студентов делает её менее удобной, чем российские аналоги.

1.3.2. Lantiv Scheduling Studio

Программа «Scheduling Studio» [7] от компании Lantiv представляет собой систему совемстной работы над расписанием и реализует следующие задачи:

- совместный доступ к редактированию расписания ВУЗа;
- оффлайн редактирование с возможностью синхронизации после появления в сети;
- цветовое выделение накладок расписания;
- составление расписания на различные временные периоды: неделя, семестр, четверть, год;
- копирование составленных элементов расписания на другие периоды.

Данный программный продукт имеет приятный и понятный интерфейс, но не имеет модуля автоматической генерации расписания, из-за чего основная часть работы всё ещё ложится на плечи диспетчеров. «Scheduling Studio» удобно использовать для составления нетривиального расписания, которое меняется от недели к неделе и плохо вписывается в шаблон школьного расписания или расписания учебных занятий ВУЗа, например. Но для составления расписани сессии требуется большая степень автоматизации, чем предлагается этим ПО.

1.4. Выводы

Сведения о возможностях каждого из описанного в параграфах 1.2 и 1.3 сведём в таблицу 1.1.

Видно, что среди систем составления расписания для составления предварительного расписания сессии лучше всего могла бы подойти программа «Галактика Расписание учебных занятий», так как она удовлетворяет большинству требований, но при этом она, как и почти всё представленное в таблице ПО, требует оплату за использование. Также среди представленных программных продуктов все, кроме одного, не предоставляют открытый доступ к исходному коду. Таким образом, в рамках данной работы необходимо было разработать программу, удовлетворяющую всем перечисленным в таблице критериям.

Таблица 1.1

Сравнение систем составления расписания

	Учитывает пожелания ППС	Интегрируется с сайтами ВУЗов	Имеет возможность задавать нетривиальное расписание	Плата за использование	Генерация предварительного расписания	Открытый исходный код
1С: ХроноГраф Расписание	+	-	-	+	+	-
Avtor	+	-	+	+	+	-
Галактика Расписание учебных занятий	+	+	+	+	+	-
Apereo UniTime	-	+	+	-	+	+
Lantiv Scheduling Studio	-	-	+	+	-	-

ГЛАВА 2. АЛГОРИТМЫ СОСТАВЛЕНИЯ РАСПИСАНИЯ СЕССИИ

Глава посвещена обзору алгоритмов, которые можно применять для составления расписания сессии, а также для оптимизации этого процесса.

В параграфе 2.1 приведена постановка задачи составления расписания сессии, далее в параграфе 2.2 рассморены два возможных режима составления расписания, а в параграфах] 2.3 и 2.4 - алгоритмы, используемые для решения данной задачи.

2.1. Постановка задачи составления расписания сессии

Для составления расписания учебной сессии в университете необходимо каждой запланированной аттестации подобрать день, время и аудиторию проведения. Известны доступные аудитории и множество аттестаций, которые необходимо провести в рамках сессии. Для каждой аудитории известно время ее доступности. Для каждого типа аттестации определена длительность, максимальное количество в день и количество дней отдыха до и после её проведения. Нужно составить расписание сессии так, чтобы общая эффективность расписания была наибольшей. Далее введём метематические обозначения для названных выше

параметров, формализуем ограничения и определим, какое расписание будет являться оптимальным.

2.1.1. Обозначения

Составим расписание сесии для g учебных групп, каждая из которых характеризуется номером и количеством учащихся в ней студентов.

Множество номеров учебных групп:

$$Gr = \{gr_1,...,gr_g\} \tag{2.1}$$

Количество студентов в соответсвующей учебной группе:

$$S = \{s_1,...,s_g\} \tag{2.2}$$

В данной задаче необходимо учитывать пожелания преподавателей, поэтому каждый преподаватель помимо ФИО должен характеризоваться множеством удобных для него дней и часов проведения аттестаций. Также преподавателям необходимо прописывать приоритет, чтобы в случаях, когда нельзя удовлетворить пожеланиям всех преподавателей, в первую очередь будут учитываться пожеланиям именно преподавателей с большим приоритетом.

Множество, состоящее из p имён преподавателей ВУЗа:

$$T = \{t_1,...,t_p\} \tag{2.3}$$

Приоритеты соответсвующих преподавателей:

$$Pr = \{pr_1,..,pr_p\} \tag{2.4}$$

Множество из q дней, которые преподаватель t выбрал возможными для проведения им аттестаций:

$$Td_t = \{td_{t1},...,td_{tq}\} \tag{2.5}$$

Множество из c часов, которые преподаватель t выбрал возможными для проведения им аттестаций:

$$Tt_t = \{tt_{t1},...,tt_{tc}\} \tag{2.6}$$

Важно также учитывать правила проведения сессий, ограничивающие количество аттестаций в день и определяющие, сколько дней отдыха нужно оставить группе до и после аттестации.

Определим множество из y типов аттестаций (экзамен, зачёт и т.п.):

$$A = \{a_1,...,a_y\} \tag{2.7}$$

Количество дней отдыха перед (Pb) и после (Pa) проведением аттестации каждого типа:

$$Pb = \{pb_1,...,pb_y\} \tag{2.8}$$

$$Pa = \{pa_1,...,pa_y\} \tag{2.9}$$

Длительность каждого типа аттестации в часах:

$$Ad = \{ad_1,...,ad_y\} \tag{2.10}$$

Максимальное количество аттестаций каждого типа в день:

$$Ac = \{ac_1,...,ac_y\} \tag{2.11}$$

При выборе аудиторий для проведения аттестаций необходимо полагаться на их размер и техническую оснащённость. Опрелелим множество из u номеров аудиторий:

$$R = \{r_1,...,r_u\} \tag{2.12}$$

Типы соответсвующих аудиторий (компьютерный класс, имеет проектор и т.п.):

$$Rt = \{rt_1,...,rt_y\} \tag{2.13}$$

Количество мест в соответсвующих аудиториях:

$$Rs = \{rs_1,..,rs_y\} \tag{2.14}$$

Аттестации, которые необходимо провести в течение сессии описываются учбной группой, дисциплиной и преподавателями, которые должны провести данную аттестацию. Так же для каждой аттестации необходимо указать её тип и тип аудитории, в которой она должна проводиться.

Множество аттестаций:

$$E = \{e_1,...,e_k\} \tag{2.15}$$

Множества групп (Eg) и дисциплин (Ec) для каждой из k аттестаций:

$$Eg = \{eg_i \in Gr, \forall i \in \{1,..,k\}\} \tag{2.16}$$

$$Ec = \{ec_1,...,ec_k\} \tag{2.17}$$

Множества типов аттестаций и типов аудиторий для каждой из k аттестаций:

$$Ea = \{ea_i \in A, \forall i \in \{1,..,k\}\} \tag{2.18}$$

$$Er = \{er_1,...,er_k\} \tag{2.19}$$

Множество, состоящее из преподавателей, проводящих аттестацию e:

$$Et_e = \{et_{ei} \in T, \forall i\} \tag{2.20}$$

Каждой аттестации необходимо сопоставить дату, часы и аудиторию для проведения. Декартово произведение всех возможных дат, часов и свободных кабинетов представляет собой множество потенциальных окон для проведения аттестаций.

Множество окон:

$$W = \{w_1,..,w_n\} \tag{2.21}$$

$wd_i \in D$ - календарный день, соответствующий окну i;

$wh_i \in H$ - время, соответствующее окну i;

$wc_i \in R, \forall i \in \{1,...,n\}$ - аудитория, соответствующая окну i.

Введём множество Solutions, состоящее из функций булевых переменных, которые принимают значение 1, если за i-ей аттестацией бронируется j-е окно:

$$S(i,j) = \begin{cases} 1 & \text{the attestation } e_i \text{ is held in window } w_j \\ 0 & \text{the attestation } e_i \text{ is not held in window } w_j \end{cases} \tag{2.22}$$

Таким образом, необходимо найти значения функции $S \in Solutions$ для $\forall i \in \{1,..,k\}$ и $\forall j \in \{1,..,n\}$. Множество пар <i,j>, для которых S(i,j) = 1, представляют собой расписание сессии - соотвествие аттестации дню, врмени и аудитории.

Программно описанные в этом параграфе множестве реализованы в качестве классов, представленных] в приложении 2.

2.1.2. Ограничения

Задача составления расписания сессии имеет ряд физических ограничений, а также ограничений, обусловленных правилами проведения сессии. Соблюдение этих ограничений позволит составить корректное расписание сессии:

Во-первых, в одной аудитории в одно время может проводиться только одна аттестация:

$$\forall j \in \{1,..,n\} \sum_{i=1}^{k} S(i,j) = 1 \tag{2.23}$$

Каждая аттестация в период сессии должна проводиться единожды, так как в данном случае не рассматривается расписание дополнительной сессии:

$$\forall i \in \{1,..,k\} \sum_{j=1}^{n} S(i,j) = Ad_e a_i \tag{2.24}$$

Преподаватель не должен отрабатывать в определённый день больше некоторого количества часов x:

$$\forall b \in \{1,..,p\} : \sum_{\forall i \in \{1,..,k\}, et_{ia}=t_b} \sum_{\forall j \in \{1,..,n\}} \sum_{\forall u \in d_u \in D, wd_j=d_u} S(i,j) <= x \tag{2.25}$$

Любая группа не может сдавать аттестации любого типа в день больше, чем макисмальное число, обусловленное типом аттестации:

$$\forall b \in \{1,..,g\}, \forall j \in \{1,..,n\}, \forall m \in \{1,..,y\} : \sum_{\forall i \in \{1,..,k\}, et_i=a_m, eg_i=gr_b} \sum_{\forall d \in D, wd_j=d} S(i,j) <= ac_m * ad_m \tag{2.26}$$

Любая группа перед любой своей аттестацией не должна иметь аттестаций в окно отдыха перед аттестацией этого типа:

$$\forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \max_{\forall j \in \{1,..,n\}, ed_j < ed_i} (ed_j) < ed_i - pb_{et_i} \tag{2.27}$$

Любая группа после любой своей аттестацией не должна иметь аттестаций в окно отдыха после аттестацией этого типа:

$$\forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \min \forall j \in \{1,..,n\}, ed_j > ed_i (ed_j) > ed_i - pa_{et_i} \tag{2.28}$$

Каждая аттестация должна проводиться в аудитории равной и или превосходящей по вместимости размеру группы:

$$\forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : s_{eg_i} > rs_{wc_j} \rightarrow S(i,j) = 0 \tag{2.29}$$

Каждая аттестация не может проводиться в аудитории с оснащённостью меньшей ожидаемой:

$$\forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : er_i < rt_{wc_j} \rightarrow S(i,j) = 0 \tag{2.30}$$

Чтобы составить расписание, комфортное для преподавательского состава, учтём некоторые ограничения связанные с предпочтениями преподавателей. Эти огнаричения уже не столь критичны, как перечисленные выше, так как их нарушение не влечёт за собой нарушение правил или законов физики, но их наличие делает расписание более удобным для сотрудников.

Каждый преподаватель волен указать список календарных дней, в которые он готов принимать аттестации. Идеальное расписание предполагает, что любой преподаватель проводит все свои аттестации только в дни из этого списка:

$$\forall t \in T, \forall d \in D : d \notin Td_t \rightarrow S(i,j) = 0 \tag{2.31}$$

Также, преподаватель может указать удобные для него часы провеления аттестаций. Идеальное расписнаие учитывает это и располагает аттестации так, чтобы они затрагивали только выбранные часы каждого преподавателя:

$$\forall t \in T, \forall h \in H : h \notin Tt_h \rightarrow S(i,j) = 0 \tag{2.32}$$

Данная задача относится к классу NP-полных задач, а значит в худшем исходе придётся перебрать все возможные k аттестаций и n окон в качестве аргументов функции S(i,j), где выполняются все описанные выше ограничения. Обобщим их для функции S:

$$S(i,j) = \begin{cases} \forall j \in \{1,..,n\} \sum_{i=1}^{k} S(i,j) = 1; \\ \forall i \in \{1,..,k\} \sum_{j=1}^{n} S(i,j) = Ad_e a_i; \\ \forall b \in \{1,..,p\} : \\ \sum_{\forall i \in \{1,..,k\}, et_{ia}=t_b} \sum_{\forall j \in \{1,..,n\}} \sum_{\forall u \in d_u \in D, wd_j=d_u} S(i,j) <= x; \\ \forall b \in \{1,..,g\}, \forall j \in \{1,..,n\}, \forall m \in \{1,..,y\} : \\ \sum_{\forall i \in \{1,..,k\}, et_i=a_m, eg_i=gr_b} \sum_{\forall d \in D, wd_j=d} S(i,j) <= ac_m * ad_m; \\ \forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \\ \max\limits_{\forall j \in \{1,..,n\}, ed_j < ed_i} (ed_j) < ed_i - pb_{et_i}; \\ \forall b \in \{1,..,g\}, \forall i \in \{1,..,k\} eg_i = gr_b : \\ \min\limits_{\forall j \in \{1,..,n\}, ed_j > ed_i} (ed_j) > ed_i - pa_{et_i}; \\ \forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : s_{eg_i} > rs_{wc_j} \rightarrow S(i,j) = 0; \\ \forall i \in \{1,...,k\}, \forall j \in \{1,..,n\} : er_i < rt_{wc_j} \rightarrow S(i,j) = 0 \end{cases} \tag{2.33}$$

Оптимальным расписанием будет такое, где минимальное количество пожеланий преподавателей игнорируется. Таким образом, сведём задачу к поиску такого расписания S, на котором выполняется слудеющий минимум:

$$\min_{\forall S \in Solutions} (\sum_{\forall i \in \{1,...,k\}, \forall j \in \{1,..,n\}, wd_j \notin Td_{et_i}, wh_j \notin Th_{et_i}} (S(i,j) * pr_{et_i})) \tag{2.34}$$

Видно, что идеальными расписаниями будут те, где учитываются пожелания всех преподавателей, так как в этом случае точно достигается наименьшее значение минимума формулы (2.34) = 0.

Для решения данной задачи рассмотрим алгоритмы динамического и линейного программирования, алгоритм на графах, эволюционный алгоритм и метод численной оптимизации.

2.2. Режимы алгоритмов составления расписния

Алгоритмы составления расписания можно использовать в двух режимах: инкрементальном и пакетном (batch-режим). В данном разделе рассмотрим, что

из себя представляет каждый из этих режимов и почему пакетный режим больше подходит для решения задачи составления расписания сессии.

2.2.1. Инкрементальный режим

Инкрементальным режимом назовём выполнение алгоритма многократно при появлении новых данных. В контексте данной задачи это означает, что каждый раз, когда новый преподаватель будет добавлять информацию о своих предпочтениях, алгоритм будет просчитывать возможные варианты расписания с учётом:

– уже имеющихся данных, которые имеют приоритет, перед новыми;
– внесённых преподавателем изменений.

Так, при каждом новом пересчёте уже составленные расписания преподавателей считаются утверждёнными, что даёт возможность не считать их заново, тем самым ускорив работу алгоритма.

Из преимуществ данного подхода можно выделить возможность преподавателя, не дожидаясь других, увидеть своё расписание. Также, с точки зрения организации, плюсом можно считать стимул преподавателей как можно раньше заполнить форму сбора информации, чтобы иметь приоритет перед другими. Таким образом появляется возможность раньше закончить процесс составление расписания.

Недостатки данного режима более существенны на практике, чем его преимущества, так как основным недостатком является закрепление наиболее удобного расписания за теми, кто заполнил форму сбора раньше остальных, что может повлечь коллизии. Можно рассмотреть случай, когда первый преподаватель, заполнивший форму сбора, выбирает большое окно для проведения экзаменов и алгоритм бронирует за ним несколько определённых случайных дат, а остальные оставляет свободными. Тогда второй преподаватель, уезжающий на конференцию в свободные даты и готовый провести экзамены в даты, занятые первым, уже не сможет это сделать, потому что у первого преподавателя приоритет, и его уже нельзя подвинуть на сводные даты.

2.2.2. Пакетный режим

Пакетным режимом в данном случае будет единоразовая обработка всех полученных сведений. Для этого необходимо определить дедлайн для сбора

пожеланий преподаватель, при наступлении которого составить предварительное расписание для всех за один вызов алгоритма.

Преимуществом такого подхода является возможность установить приоритеты преподавателей и групп, на основе которых можно решать коллизии в случае невозможности нахождения идеального решения, учитывающего пожелания каждого.

Недостатком такого подхода можно назвать невозможность до дедлайна получить расписание, но в реальности это не будет глобальным минусом, потому что составленное таким образом предварительное расписание, всё равно, в последствии может быть скорректировано вручную.

Таким образом, из двух рассмотренных режимов для решения задачи составления расписания больше подходит именно пакетный, потому что его преимущества более значимы, а единственный недостаток не играет роли на практике, в отличие от описанных недостатков инкремениального режима.

2.3. Обзор алгоритмов составления расписания сессии

2.3.1. Обход графа в глубину для поиска идеальных решений

Задача составления расписания сессии является NP-полной, как, например, задача обхода графа, для которой существует множество классических алгоритмов, решающих её. Алгоритм обхода графа в глубину [6] является одним из них. Но чтобы применить этот алгоритм для составления расписания, необходимо свести эту задачу к задаче построения графа, у которого в качестве вершин выступают элементы расписания - аттестации и временные окна для их проведния. Наличие ребра обуславливается выполнением двух следующих условий:

- ребро из вершины с аттестацией i из отсортрованного списка аттестаций E, может быть соединено только с аттестацией i+1 из E;
- ребро из вершины <i,j> с аттестацией i и окном j в вершину <i+1, q> может существовать, если выполняются условия формул (2.33), (2.31) и (2.32) для S(i+1, q) при n=i+1.

Тогда решением будет являться связный граф состоящий из k вершин - пар аттестаций и временных окон, отведённых для этих аттестаций.

Основная идея обхода в глубину – когда возможные пути по ребрам, выходящим из вершин, разветвляются, нужно сначала полностью исследовать одну ветку

и только потом переходить к другим веткам. Сложность классического обхода в глубину с матричным заданием графа равна $O(m^2)$, где $m = n * k$ - количество вершин. Но так как известно, что доступность рёбер для каждой из вершин с аттестацией i огрничена n вершинами с аттестацией i+1, сложность снизится до $O(n^2 * k)$. Но так как для проверки остальных условий необходимо проверять i предыдущих вершин графа, сложность останется $O(n^2 * k * \log(k))$.

В процессе решения занчениями функции S(i,j) будет заполняться булева матрица. Важно помнить, что значение 1 в ячейке может быть проставлено, только при соблюдении ограничений системы (2.33) , и стоит заметить, что выполнение ограничения, что в одной аудитории в одно время может проводиться только одна аттестация, влечёт за собой наличие в одном столбце матрицы не более одного значения 1, из-за чего матрицу можно заменить на целочисленный массив, где индексы соответствуют индексу возможного окна, а значения - индексу события, которое будет там проведено.

Будем считать, что идеальное решение найдено, если в каждой строке матрицы есть значение 1 или каждое число от 0 до k встречается в массиве решения, что означает, что каждую аттестацию можно сопоставить какому-то окну w_j с учётом перечисленных выше ограничений. Такое решение добавляется в список всех идеальных решений. Даллее представлен псевдокод процедуры FindSolutions 2.1. , которая реализует алгоритм обхода графа в глубину. Заметим, что в списке доступных окон W все элементы отсортированы по дням, кабинетам и часам, чтобы легко можно было оценивать занятость одного кабинета несколько часов подряд.

В процедуре FindNextStartTime 2.4 проводится проверка условий из формулы (2.33), чтобы подобрать следующее подходящее ребро, исходящее из указанной вершины. В случае, если для вершины не существует исходящего ребра, которое удовлетворяет всем условиям, данная процедура вернёт значение -1, в противном случае вернёт следующее ребро. В данной функции также происходит проверка условий из формул (2.31) и (2.32), чтобы обеспечить поиск идеального решения, удовлетворяющего пожланиям всех преподавателей.

Algorithm FindSolutions

Input: $\mathbb{E}$, $\mathbb{W}$, $k = |\mathbb{E}|$, $n = |\mathbb{W}|$

Output: $\mathbb{S}olutions$

```
1.  Solutions ← {}, S ← [], i ← 0
2.  while i < k do
3.      time = NextTime(i)
4.      for ∀u ∈ 0,...,k do
5.          if S[u] = i then
6.              S[u] ← −1
7.      nextTime = FindNextStartTime(i,time)
8.      if nextTime = −1 then
9.          if i = 0 then
10.             return Solutions
11.         else
12.             i ← i − 1
13.     else
14.         duration = ed_i
15.         for ∀b ∈ {nextTime,...,nextTime + duration} do
16.             S[b] ← i
17.         i ← i + 1
18.     if i = k then
19.         Solutions ← Solutions ⋃ S
20.         i ← i − 1
21. return Solutions
```

Рис.2.1. Псевдокод алгоритма DFS для составления расписания

Function FindNextStartTime

Input: $event, time, \mathbb{E}, \mathbb{W}, \mathbb{S}, k = |\mathbb{E}|, n = |\mathbb{W}|$

Output: t

1. **if** $time < 0$ **then**
2. **return**-1
3. **for** $i \in \{time,...,k\}$ **do**
4. $w \leftarrow W[i]$
5. **if** $S[i] = -1 \wedge rt_w = er_{event} \wedge i + ad_{event} < n \wedge s_{eg_{event}} <= er_w \wedge er_w = er_{W[i+ad_{event}]} \wedge ed_w = ed_{W[i+ad_{event}]} \wedge wd \in Td_{event} \wedge wt \in Tt_{event} \wedge wt + ed_w \in Tt_{event}$ **then**
6. $teacherTime \leftarrow 0, countPerDay \leftarrow 0, j \leftarrow 0$
7. **while** $i < k$ **do**
8. **if** $S[j] = -1$ **then**
9. $j \leftarrow j + 1$
10. $continue$
11. **if** $w = S[j] \wedge wt \in et_{solution[j]}$ **then**
12. $break$
13. **if** $et_{S[j]} = et_{event}$ **then**
14. $teacherTime = teacherTime + 1$
15. **if** $teacherTime > maxTimePerDay - ed_{event}$ **then**
16. $break$
17. **if** $eg_{S[j]} = eg_{event}$ **then**
18. **if** $(ea_{S[j]=ea_{event}}) \wedge (wd = ed_{S[j]})$ **then**
19. $countPerDay \leftarrow countPerDay + 1$
20. **if** $(ac_{et_{event}} < countPerDay) \vee (wt - pb_{ea_{event}} <= wd_{S[j]} \wedge wt + pa_{ea_{event}} >= wd_{S[j]})$ **then**
21. $break$
22. $k \leftarrow k + 1$
23. **if** $j = n$ **then**
24. **return** i
25. **return**-1

Рис.2.2. Проверка условий формул (2.33), (2.31) и (2.32)

2.3.2. Алгоритм обхода графа в глубину с учётом приоритетов преподавателей

На практике возникают ситуации, когда недостаток аудиторий и накладки в личных расписаниях преподавателей не позволяют найти такое расписание, которое удовлетворит всем пожеланиям. По этой причине приходится учитывать приоритеты ограничений и минимизировать потери по формуле (2.34).

Существует ряд ограничений, поступиться с которыми нельзя. В их число входят правила проведения аттестаций и физические условия, связанные с проведением экзаменов в аудиториях, опписанные в формуле (2.33). Но есть и менее жёсткие факторы - это пожелания преподавателей к датам и времени экзаменов.

Каждый преподаватель указывает дни и время, когда ему было бы удобно присутствовать на аттестации. В некоторых случаях удобство эквивалентно тому, что в остальные дни преподаватель в принципе не может принимать экзамены. Приоритет таких ограничений должен быть выше, чем у случаев, когда преподавателю просто комфортнее было бы присутствовать на экзаменах в определённые дни.

Для этого было введено множество приоритетов Pr, где больший приоритет соответствует большему влиянию учёта пожеланий преподавателя на потери функции (2.34). Система приоритетов в случае появления коллизий при составлении расписания позволит в первую очередь пытаться нарушить только пожелания с наименьшим приоритетом и только, если это необходимо, с более высокими.

Модернизируем алгоритм поиска «идеального» решения:

Для этого нужно хранить массив ignoreWishes, в котором будут фиксироваться те аттестации, для которых приходится искать решения без учёта предпочтений преподавателей.

Также, в алгоритме 2.3 на строке 2 необходимо отсортировать массив аттестаций по приоритетам преподавателей, чтобы искать решения без учёта высокоприоритетных преподавателей лишь в последнюю очередь.

Метод поиска решений FindSolutions 2.3 теперь содержит условный оператор, который в случае определения того, что идеального решения найти не удаётся, переходит в режим игнорирования пожеланий преподавателя для конкретного события. Для этого в массиве ignoreWishes выставляется true по индексу, соотвествующему этому событию.

Algorithm FindSolutions

Input: $\mathbb{P}, \mathbb{E}, \mathbb{W}, k = |\mathbb{E}|, n = |\mathbb{W}|$

Output: $\mathbb{S}$

```
1.  ignoreWishes ← [], S ← [], i ← 0
2.  Sort(E)
3.  while i < k do
4.      time = NextTime(i)
5.      for ∀u ∈ 0,...,k do
6.          if S[u] = i then
7.              S[u] ← −1
8.      nextTime = FindNextStartTime(i,time)
9.      if ignoreWishes[i] = False ∧ nextTime = −1 then
10.         ignoreWishes[i] = True
11.         nextTime = FindNextStartTime(i,0)
12.     if nextTime = −1 then
13.         if i = 0 then
14.             return null
15.         else
16.             ignoreWishes[i] = False
17.             i ← i − 1
18.     else
19.         duration = ed_i
20.         for ∀b ∈ {nextTime,...,nextTime + duration} do
21.             S[b] ← i
22.         i ← i + 1
23. return S
```

Рис.2.3. Псевдокод алгоритма DFS для составления расписания с учётм приоритетов преподавателей

Метод поиска следующего подходящего времени и аудитории теперь может работать в двух режимах. В режими игнорирования рассматриваются только те окна, которые хотя бы в какой-то мере не удовлетворяют пожеланиям преподавателя. Для этого в алгоритме 2.4 в строке 5 используется логический оператор XOR между результатом проверки соблюдения пожеланий преподавателя и значением в массиве игнорирования.

Function FindNextStartTime

Input: $event, time, \mathbb{E}, \mathbb{W}, \mathbb{S}, k = |\mathbb{E}|, n = |\mathbb{W}|, ignoreWishes$

Output: t

1. **if** $time < 0$ **then**
2. **return**-1
3. **for** $i \in \{time,...,k\}$ **do**
4. $w \leftarrow W[i]$
5. $teacherWish \leftarrow (wd \in Td_{event} \wedge wt \in Tt_{event} \wedge wt + ed_w \in Tt_{event}) \nrightarrow ignoreWishes[i]$
6. **if** $S[i] = -1 \wedge rt_w = er_{event} \wedge i + ad_{event} < n \wedge s_{eg_{event}} <= er_w \wedge er_w = er_{W[i+ad_{event}]} \wedge ed_w = ed_{W[i+ad_{event}]} \wedge teacherWish$ **then**
7. ...
8. **if** $j = n$ **then**
9. **return** i
10. **return**-1

Рис.2.4. Проверка условий формулы (2.33) с учётом режима игнорирования

Асимптотическая сложность этой интерпретации алгоритма тоже равна $O(n^2 * k)$, но требует дополнительную память на хранение бинарного массива ignoreWishes.

2.3.3. Генетический алгоритм

Генетический алгоритм имитирует естественный отбор и состоит из нескольких этапов:

- Создание популяции;
- Размножение путём обмена генами у двух особей;

– Мутации путём проведения определённых изменений генов некоторых особей;
– Расчёт метрики приспособленности для особей и отбор лучших для перехода в новое поколение.

Определим основные понятия генетического алгоритма для простого случая составления расписания. Ген будет представлять собой объект с полями: Группа, Преподаватель, Предмет, Тип аттестации, Дата, Время, Аудитория. Популяция - все возможные комбинации расположения аттестаций по аудиториям во времени. Особь - некоторый набор генов (пример расписания). Такая задача сводится к нахождению наилучшей особи - расписания.

Фитнес-функция для этого алгоритма, в которой задаётся способ расчёта метрики качества расписания, должна учитывать ограничения (2.33), чтобы с каждым новым поколениям расписания имели всё меньше и меньше накладок.

Из преимуществ генетического алгоритма можно выделить:

– Адаптивность времени выполнения;
– Возможность ограничить количество поколений алгоритма, по истечении которых алгоритм оставит наиболее подходящее расписание;
– Адаптивность качества;
– Если задать условием остановки алгоритма - достижение некоторого счёта в фитнес-функции, можно завершить работу с приемлемыми потерями, полученными, например, от пренебрежения пожеланиями преподавателями.

Недостатками генетического алгоритма для задачи составления расписания будут:

– Массивные входные данные, так как для формирования популяции необходимо брать декартово произведение нескольких датасетов, а после манипулировать большой начальной популяцией;
– Нелинейная зависимость качества от длительности выполнения, так как часто в эволюционных алгоритмах последующее поколение бывает менее приспособлено, чем предыдущее, а значит, заранее предугадать количество итераций при заданном минимальном пороге качества нельзя;
– Сложность подбора параметров фитнес-функции;
– Длительная отладка.

2.3.4. Жадный алгоритм

Жадный алгоритм - это алгоритм, который на каждом шаге выбирает локально оптимальное решение, ожидая, что итоговое решение тоже будет оптимальным. В какой-то мере жадным алгоритмом пользуются диспетчеры при составлнии расписания сессии: сначала выставляются даты для аттестаций проводимых высокоприоритетными преподавателями-совместителями, а далее в свободные окна расставляются менее приоритетные.

Но программная реализация жадного алгоритма при составлении расписания слишком часто не будет сходиться к решению, которое удовлетворяет всем ограничениям. Во многих задачах допустимо искать не глобальный оптимум решения, чтобы уменьшить время выполнения, но при составлении расписания сессии нельзя пренебрегать рябом ограничений, а значит если на какой-то итерации алгоритма некоторую аттестацию остаётся расположить в занятой аудитории, применение такокой оптимизации совершенно теряет смысл. Можно повторять жадный поиск решения несколько, если сразу не удаётся найти подходящее расписание, но так процесс подбора решения потеряет свою "жадность"и выродится к полному перебору.

2.4. Методы оптимизации поиска лучших решений

2.4.1. Метрики оценки качества расписания

Для оценки предложенных расписаний подсчитаем несколько метрик:

Длительность сессии для каждого преподавателя с учётом его приоритета:

$$DurationT = \sum_{\forall t \in T} (pr_t * (\max_{i \in \{0,\dots,n\}, et_{S[i]}=t} (wd_i) - \min_{i \in \{0,\dots,n\}, et_{S[i]}=t} (wd_i))) \qquad (2.35)$$

Считаем, что расписание преподавателя должно быть максимально компактным по датам, а значит разницу между первым и последним днём проведения экзаменов нужно минимизировать.

Суммарная длительность минимального отдыха между экзаменами по группам:

$$Pause = \sum_{\forall gr \in Gr} (\min_{i,j \in \{0,\dots,n\}, eg_{S[i]}=gr, wd_i \neq wd_j, i>j} (wd_i - wd_j)) \qquad (2.36)$$

Считаем, что студентам желательно не иметь маленьких перерывов между экзаменами, поэтому стремимся максимизировать Pause для расписания.

Суммарная длительность сессии по группам:

$$DurationG = \sum_{\forall gr \in Gr} \max_{i \in \{0,\ldots,n\}, eg_{S[i]}=gr} (wd_i) \tag{2.37}$$

Чем раньше закончится сессия, тем раньше у иногородних студентов появится возможность уехать домой, а значит порядковый номер последнего дня сессии для группы нужно пытаться минимизировать.

Суммарное кол-во рабочих дней для преподавателей с учётом их приоритетов:

$$Work_t = \{wd_i, \forall i \; in 1,\ldots,n, et_{S[i]} = t\}; \tag{2.38}$$

$$WDays = \sum_{\forall t \in T} (pr_t * \|Work_t\|) \tag{2.39}$$

Считаем, что преподавателю удобно иметь максимальное количество дней, свободных от проведения экзаменов, а значит метрику WDays стараемся минимизировать. На практике это означает, что лучше провести несколько зачётов в один день, чем заставлять человека несколько раз в неделю приезжать в университет.

Все эти метрики имеют разные шкалы, какие-то мы пытаемся максимизировать, какие-то минимизировать, одни измеряются несколькими неделями, а другие несколькими днями. Поэтому каждую метрику для всех рассматриваемых решений необходимо нормализовать. Это приведёт данные метрики к единой шкале и улучшит их интерпретируемость.

Ко всем метрикам качества для выбранных для сравнения расписаний применим минимаксную нормализацию [14], которая расчитывается по формуле:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2.40}$$

Далее сделаем так, чтобы большему значению каждой метрики соответствовало более предпочтительное расписание. Для этого те метрики, для которых лучшее решение имеет меньшее значение - вычтем из 1. Тем самым соотношение всех значений метрики среди расписаний останется таким же, но качество метрики будет расти с качеством решения.

2.4.2. Метод ветвей и границ

Метод ветвей и границ принадлежит к группе алгоритмов целочисленного линейного программирования и применяется для оптимизации задач полного

перебора [3]. Он как нельзя кстати придётся для модернизации алгоритма полного обхода графа в глубину. Суть этого метода состоит в том, чтобы в процессе обхода графа, отбрасывать заведомо менее оптимальные решения, чем у же найденные.

Такой подход потребует выделить память под хранение метрик качества. В памяти необходимо держать качество текущего расчитываемого расписания и предыдущего лучшего расписания. В качестве метрики качества возьмём разброс между первым и последним днём проведения сессии для каждого преподавателя. Чем он меньше, тем лучше считается расписание. Естественно, можно выбрать и другие критерии качества расписания, но алгоритм при этом поменяется незначительно. Выбор такой метрики вынуждает хранить массив с количеством дней разброса для каждого преподавателя, поэтому выделим память под масиивы metrics[] и bestMetrics[] размером p (число преподавателей).

Модернизируем функцию поиска решений методом обхода графа в глубину, добавив в качестве условия перехода к следующей аттестации проверку на то, является ли расписание на данном этапе, худшим, чем предыдущее полностью составленное расписание. Эта проверка добавляется в строке 18 алгоритма 2.5.

Algorithm FindSolutions

Input: $\mathbb{E}$, $\mathbb{W}$, $k = |\mathbb{E}|, n = |\mathbb{W}|$

Output: $\mathbb{S}olutions$

1. $metrics \leftarrow [], bestMetrics \leftarrow [], Solutions \leftarrow \{\}, S \leftarrow [], i \leftarrow 0$
2. **while** $i < k$ **do**
3. $\quad time = NextTime(i)$
4. $\quad$ **for** $\forall u \in 0,...,k$ **do**
5. $\quad\quad$ **if** $S[u] = i$ **then**
6. $\quad\quad\quad S[u] \leftarrow -1$
7. $\quad nextTime = FindNextStartTime(i,time)$
8. $\quad$ **if** $nextTime = -1$ **then**
9. $\quad\quad$ **if** $i = 0$ **then**
10. $\quad\quad\quad$ **return** $Solutions$
11. $\quad\quad$ **else**
12. $\quad\quad\quad i \leftarrow i - 1$
13. $\quad$ **else**
14. $\quad\quad duration = ed_i$
15. $\quad\quad$ **for** $\forall b \in \{nextTime,...,nextTime + duration\}$ **do**
16. $\quad\quad\quad S[b] \leftarrow i$
17. $\quad\quad i \leftarrow i + 1$
18. $\quad\quad$ **if** $i = 0 \vee \neg isItWorse(i)$ **then**
19. $\quad\quad\quad i \leftarrow i + 1$
20. $\quad\quad$ **else**
21. $\quad\quad\quad$ **return** $Solutions$
22. $\quad$ **if** $i = k$ **then**
23. $\quad\quad Solutions \leftarrow Solutions \bigcup S$
24. $\quad\quad bestMetrics \leftarrow metrics$
25. $\quad\quad i \leftarrow i - 1$
26. **return** $Solutions$

Рис.2.5. Псевдокод алгоритма DFS с использованием метода ветвей и границ

Функция isItWorse 2.6 пересчитывает для рассматриваемого преподавателя длительность его сессии и проверяет, не является ли оно худшим, чем у того же преподавателя в предыдущем полном расписании. Для этого в массиве текущего решения находится максимальный и минимальный день, для рассматриваемого преподавателя и возвращается их разница.

Function isItWorse

Input: $event, time, \mathbb{E}, \mathbb{W}, \mathbb{S}, k = |\mathbb{E}|, n = |\mathbb{W}|, ignoreWishes$

Output: t

1. max,$min \leftarrow wd_{n-1}, i \leftarrow event - 1$
2. **while** $i >= 0 \wedge et_i = et_{event}$ **do**
3. $\quad j \leftarrow \min_{j \in 0,k-1, S[j]=i}(j)$
4. $\quad$ **if** $wd_j > max$ **then**
5. $\quad\quad max = wd_j$
6. $\quad$ **else if** $wd_j < min$ **then**
7. $\quad\quad min = wd_j$
8. $\quad i \leftarrow i - 1$
9. $metrics[event] \leftarrow max - min$
10. **return** $bestMetrics[0] \neq -1 \wedge bestMetrics[et_{event}] < metrics[et_{event}]$

Рис.2.6. Функция сравнения метрики качества для преподавателя с предыдущим лучшим решением

2.4.3. Алгоритм иммитации отжига

Для выбора лучшего из расписаний и расчёта всех метрик необходимо будет дополнительно обойти x расписаний, состоящих из n возможных временных окон. А потом просуммировать их по p преподавателям и g учебным группам. Если входные данные были такими, что было найдено 5-10 возможных расписаний, то не составит труда просчитать их все, так как относительно n - числа временных окон, оно вносит минимальный вклад в сложность вычислений.

Но рассмотрим случай, когда методом полного перебора было найдено большое количество возможных расписаний, и число x достаточно велико и даже сравнимо с n. В такой ситуации было бы полезно находить более оптимальные решения без расчёта метрик для каждого расписания, жертвуя некоторой погрешностью.

Для этого можно применить оптимизационный алгоритм имитации отжига [9]. Он основан на имитации физического процесса отжига металлов - при постепенно понижающейся температуре переход атома из одной ячейки кристаллической решётки в другую происходит с некоторой вероятностью, которая понижается с понижением температуры.

При помощи моделирования этого процесса ищется такая точка или множество точек, на котором достигается минимум некоторой функции F(S). Для задачи выбора лучшего расписания функция F(S) - функция, вычисляющая качество расписания, например, по метрике DurationG, где S - некоторое из полученных расписаний:

$$F(S) = \sum_{\forall gr \in Gr} \max_{i \in \{0,\ldots,x\}, eg_{S[i]}=gr} (wd_i) \tag{2.41}$$

Решение ищется последовательным вычислением точек $S_0,\ldots,S_f$ из множества Solutions(для задачи выбора оптимального расписания - возможные варианты расписаний), где f - количество итераций понижения температуры. Каждая последующая точка претендует на то, чтобы лучше предыдущих приближать решение. В качестве исходных данных возьмём точку S_0. На каждом шаге алгоритм вычисляет новую точку и понижает значение счётчика - температуры Q_i , и когда он достигает нуля, алгоритм останавливается в этой точке.

Температурой для данной задачи возьмём некоторую константу, достаточно большую, чтобы хватило итераций для нахождения примерного оптимума, но при этом меньшую, чем x. Пусть это будет константа $Q_0 = \frac{n}{10}$.

К точке x_i на каждом шаге применяется оператор A, который случайным образом модифицирует её, в результате чего получается новая точка x^*. Он возвращет следующее распиание для расчёта метрики качества на следующей итерации.

$$A(S_i) = S_{Y(0,x-i)}, \text{ Y - random number on a segment [0,x-i]} \tag{2.42}$$

Но перейдёт ли S^* в S_{i+1} произойдёт с вероятностью, которая вычисляется в соответствии с распределением Гиббса:

$$P(x^* \to S_{i+1} \mid S_i) = \begin{cases} 1, & F(S^*) - F(S_i) < 0 \\ \exp\left(-\dfrac{F(S^*) - F(S_i)}{Q_i}\right), & F(S^*) - F(S_i) \geqslant 0 \end{cases} \tag{2.43}$$

В псевдокоде 2.7 представлено создание цикла понижения температуры, который возвращает расписание, которое будет выбрано, когда температура станет нулевой.

Algorithm SimulatedAnnealing

Input: $\$olutions, \mathbb{E}, \mathbb{W}, x = |\$olutions|, k = |\mathbb{E}|, n = |\mathbb{W}|$

Output: $\$$

1. $i \leftarrow 0$
2. $metrics[i] = F(Solutions[i])$
3. **for** $q \in n/10,...,0$ **do**
4. $\quad next = A(Solutions[i])$
5. $\quad metrics[next] = F(Solutions[next])$
6. $\quad pr = P(S_{next}) \mid S_i)$
7. $\quad$ **if** $Y(0,1 < p)$ **then**
8. $\quad\quad i \leftarrow next$
9. **return** $Solutions[i]$

Рис.2.7. Псевдокод алгоритма имитации отжига для определения оптимального расписания из предложенных

Из преимуществ этого метода оптимизации можно выделить его скорость, потому что расчёт метрик качества расписания осуществляется только для установленного числа решений. Но при этом это снижает точность выбора наиболее удачного расписания, так как присутствует фактор случайности при выборе следующего решения на каждом шаге алгоритма.

2.5. Выводы

Система составления расписания сессии СПбПу реализует алгоритм обхода в глубину с учётом приоритетов преподавателей, одним из входных параметров которого будет максимальное количество расчитанных расписаний. Этот параметр необходим для ускорения работы алгоритма, если не стоит задачи выбирать самое лучшее из всех возможных расписаний. Реализация этого алгоритма представлена в приложении 1

Если выбрана метрика определения наиболее оптимального распиания, можно применить метод ветвей и границ, чтобы ещё ускорить вычисления и вернуть

меньшее каоличество наиболее качественных расписаний сессии. Метод имитации отжига плохо применим к оптимизации выбора самого удачного расписания на практике, так как он нацелен на относительно большое число входных вариантов расписания, что во-первых редкость, потому что зачастую предпочтения преподавателей и загруженность помещений не даёт большой выбор расписаний, а во-вторых требует длительного времени на расчёт большого числа расписаний, среди которых будет выбираться лучшее.

ГЛАВА 3. ВЫБОР СРЕДСТВ РЕАЛИЗАЦИИ

В данной главе речь идёт о ПО, рассматривавшемся в качестве средств разработки ситемы составления расписания сессии для СПбПу. В параграфе 3.1 приводится обоснования выбора языка программирования, в параграфе 3.2 - фреймворка для работы с веб-составляющей системы, а в параграфе 3.4 - рассказано о способе хранения и взаимодействия с данными.

3.1. Выбор языка программирования

Так как система составления расписания сессии должна иметь веб-модуль, необходимо подобрать подхлдящий для этой цели язык программирования. Среди высокоуровневых языков, которые потенциально могли бы сделать процесс разработки более быстрым, а результат качественным рассматривались следующие претенденты:

- Java;
- JavaScript;
- C#;
- C++.

3.1.1. Java

Java - один из самых пополуярных объектно-ориентированных языков программирования, что всягда является преимуществом, так как количество и качество документации, множество фреймворков и библиотек под любые нужды делает процесс разработки быстрее и эффективнее. Технология Garbage Collection, реали-

зованная в Java оптимизирует управление памятью программ, что очень важно для работы с большим количеством входных данных, как в задаче составления расписания сессии. Также плюсом данного языка является его кроссплатформенность, а значит код на нём может быть запущен везде, где установлена Java Virtual Machine.

3.1.2. JavaScript

JavaScript - язык программирования, который изначально использовался для разработки фронтенда и с помощью скриптов встраивал в HTML страницы исполняемый код. Сейчас JavaScript можно использовать и в качестве основного языка для написания бэкенда, так как с появлением среды Node.js программы на JavaScript можно запускать и на сервере. Преимуществом JavaScript всё ещё является его ориентированность на работу с веб страницами, а также его простота. Скрипты на этом языке поддерживаются всеми современными браузерами и не требуют установки дополнительного программноо обеспечения. В качестве языка для бэкенда недостатком служит отсутствие явной типизации, а так же разработку усложняет тот факт, что нет возможности узнать об ошибке на этапе компиляции, а значит о том, что где-то в коде появилось сложение строки с счислом, разработчик узнает только тогда, когда программа выполнится до этого места и не раньше.

3.1.3. C#

C#, как и Java, является кроссплатформенным языком программирования, который разработан для создания приложений среды NET Framework. С точки зрения разработки C# удобен обилием синтаксического сахара, за счёт которого упраздняются громоздкие конструкции, делающие написание кода более бастрым, а также повышающие его читаемость. Это в какой-то мере сказывается на производительности вычислений, но окупается удобством разработки. Помимо этого для C# тоже существует множество фреймворков, что также предоставляет возможность переиспользовать уже готовые технологии.

3.1.4. C++

C++ - объектно-ориентрованный язык программирования с возможностью низкоуровневой работы с данными, что даёт возможность писать на нём даже микроконтроллеры. Но сложность синтаксиса и небогатая стандартная библиотека затрудняет высокоуровневую разработку. C++ достаточно популярен и имеет

большое количество документации, но даже с ней не всегда легко правильно самостоятельно обеспечить грамотную работу с памятью. Ещё одним недостатком этого языка является зависимость от платформы и этот фактор перевешивает даже отличную производительность этого языка.

3.1.5. Определение языка программирования

Для реализации сервиса составления предварительного расписания сессии к языку программирования предъявлялись следующие требования:

- Наличие подробной докуменации;
- Наличие фреймворков для упрощения работы с базой данных и веб-сервисом программы;
- Кроссплатформенность;
- Удобство и скорость разработки.

Язык Java удовлетворяет всем перечисленным критериям благодаря своей популяности среди программистов. Это существенно упрощает процесс разработки, так как для Java существует множество пособий, а способы устранения возникающих ошибок уже в большинстве собой описаны на различных интернет-ресурсах. Далее выбор фреймворков будет рассматриваться именно для Java, благо для данного языка их существует достаточно. Но стоит заметить, что язык JavaScript, хоть и не очень хорошо подходит для основоного языка в этом проекте, но будет использоваться для создания веб-интерфейса, так как Java не очень хорошо приспособлена для этой задачи.

3.2. Выбор фреймворка для построения веб-приложения

3.2.1. Spring

Spring - Java-фреймворк, состоящий из множества компонентов таких как Inversion of Control для управления объектами, MVC для расширения Servlet API и прозрачного разделения между слоями модель, представление, контроллер и даже компонента доступа к базам данных [8]. Все эти и другие компоненты могут добавляться в проект независимо, что делает разработку более гибкой. Наличие этих модулей избавляет разработчика от низкоуровневой работы с запросами, налаживания взаимодействия с базами данных и прочих моментов имющих чисто технический характер и не имеющих связи с бизнес-задачей программы. Ещё

одним преимуществом Spring для разработчика является его популярность и качественная документация, чего часто не хватает молодым и мало известным проектам.

3.2.2. Dropwizard

Dropwizard - это фреймворк для создания веб-сервисов RESTful [4]. Он по умолчанию поставляется с такими библиотеками, как Jetty server, Jackson, Metrics, Hibernate Validator и Guava. Dropwizard не имеет русскоязычной документации, что является недостатком. Так же у Dropwizard не так много модулей, как у Spring, но всё ещё просто расширяется с ипользованием файлов конфигураций.

3.2.3. Vert.x

Vert.x - это асинхронный, событийно ориентированный фреймворк, создатели которого во многом вдохновлялись фреймворком node.js [10]. Он позволяет оптимизировать параллельную отправку и получение запросов, обрабатывая их большее количество с меньшими ресурсами, нежели фреймворки, основанные на блокирующем асинхронном вводе-выводе. Vert.x позволяет работать с действительно нагруженными системами, но не иммет такого же количества модулей для разных целей, как Dropwizard и тем более Spring.

3.3. Определение фреймворка для построения веб-приложения

Для написания системы составления расписнаия сессии СПбПу из рассомтренных фреймворков был выбран именно Spring, потому что обладает всеми необходимыми для данной задачи модулями. Огромное количество библиотек, включённых в Spring может помочь при дальнейшем расширении сервиса без переписывания его на другой фреймворк. Также при поддержании работы системы другими разработчиками не возникнет проблемы с изучением Spring, потому что для него существует множество справочных материалов в том числе на русском языке.

3.4. Хранение данных

Spring Framework имеет поддержку ORM Hibernate [5]. ORM или Object–Relational Mapping - это технология, позволяющия сопоставлять объекты объекно–ориентрованной модели, на которой зиждится Java, с сущностями базы данных. Таким образом, не нужно вручную создавать таблицы с полями, соответствующими структуре Java - объекта, ведь можно просто добавить к нему аннотацию, а Hibernate сделает всё автоматически, а самое главное автоматически преобразует полученные результаты select-запроса в объект, что не всегда просто делать самостоятельно. Также, Hibernate имеет предопределённые запросы к базе данных, из-за чего можно не прописывать монотонные sql-запросы на поиск и удаления элемента по id или добавления его в таблицу.

Hibernate поддерживает диалекты и MySQL, и Oracle, и Microsoft SQL Server, и PostgreSQL, поэтому в качестве СУБД была выбрана PostgreSQL, так как она полностью бесплатна и проста в установке на любой платформе.

3.5. Выводы

Таким образом, в качестве основного языка программирования был выбран Java, который во фронтенд части дополняется JavaScript, более приспособленным для реализации интерфейсов. Среди фреймворков для работы с веб-приложениями для Java выбор пал на Spring Framework, имеющий множество разным модулей, в том числе Hibernate, с помощью которого осуществляется взаимодействие с базой данной PostgreSQL.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 1С: ХроноГраф Расписание. — URL: https://1c.ru/rus/products/1c/predpr/compat/catalog/solution.jsp?SolutionID=15710 (дата обращения: 11.03.2021).
2. Apereo UniTime. — URL: https://www.unitime.org/ (дата обращения: 11.03.2021).
3. *Doig L. A. H.* An automatic method of solving discrete programming problems. — 1960.
4. Dropwizard Docs. — URL: https://www.dropwizard.io/en/latest/getting-started.html (дата обращения: 17.03.2021).
5. Hibernate Docs. — URL: https://hibernate.org/orm/documentation/5.4/ (дата обращения: 02.04.2021).
6. *Knuth D. E.* The Art of Computer Programming Vol 1. 3rd ed. — 1997. — URL: https://www-cs-faculty.stanford.edu/~knuth/taocp.html (дата обращения: 17.02.2021).
7. Lantiv Scheduling Studio. — URL: https://scheduling-studio.lantiv.com/ (дата обращения: 11.03.2021).
8. Spring Framework documentation. — URL: https://spring.io/projects/spring-framework (дата обращения: 17.03.2021).
9. *Strobl M.A.R.; Barker D.* On simulated annealing phase transitions in phylogeny reconstruction. — 2016. — URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4912009/ (дата обращения: 19.02.2021).
10. Vert.x Docs. — URL: https://vertx.io/docs/ (дата обращения: 17.03.2021).
11. АВТОРасписание. — URL: http://avtor.bravosoft.org/ (дата обращения: 11.03.2021).
12. Галактика Расписание учебных занятий. — URL: https://galaktika-it.ru (дата обращения: 11.03.2021).
13. Расписание СПбПУ. — URL: http://ruz.spbstu.ru (дата обращения: 11.03.2021).
14. *Freedman D. Pisani R. P. R.* Statistics: Fourth International Student Edition. — 2007.

Приложение 1

Реализация алгоритма поиска расписаний с учётом приоритетов преподавателей на Java

```
import java.util.*;

  public class SchedulePriorAlgorithm {
    List<Event> events;
    List<DateTimeClass> dtc;
    int[] solution;
    int eventSize;
    int dtcSize;
    int maxTimePerDay;
    boolean[] ignoreWishes; //приходится ли игнорировать пожел
       ания преподавателя для этого события

    public SchedulePriorAlgorithm(List<Event> events, List<
       DateTimeClass> dtc, int maxTimePerDay) {
      this.events = events;
      // отсортируем события так, чтобы аттестации, которые п
         роводятся более приоритетным преподавателями, были ра
         ньше
      Collections.sort(this.events);
      this.dtc = dtc;
      this.eventSize = events.size();
      this.dtcSize = dtc.size();
      this.maxTimePerDay = maxTimePerDay;

      this.solution = new int[dtcSize];
      for (int i = 0; i < dtcSize; i++) {
        solution[i] = -1;
      }
      this.ignoreWishes = new boolean[eventSize];
    }

    boolean findSolution() {
      int event = 0;
      while (event < eventSize) {
        int time = this.getTime(event);
        int nextTime;
        this.cleanEvent(event);
        nextTime = this.findNextStartTime(event, time);
```

```
        //переходим в режим игнорирования пожеланий преподавателя, если не смогли найти идеального решения
        if(!ignoreWishes[event] && nextTime == -1){
          ignoreWishes[event] = true;
          nextTime = this.findNextStartTime(event, 0);
        }

        if (nextTime == -1) { //если не удалось найти другого подходящего DateTimeClass для этого события
          if (event == 0) {
            return false; //если речь о первом событии, то уже были перебраны все остальные варинты, и решения нет
          } else {
            ignoreWishes[event] = false;
            event--; // иначе возвращаемся к предыдущему событию и пробуем изменить для него DateTimeClass
          }
        } else {
          this.submitDateTimeClass(event, nextTime); //бронируем за событием время и аудиторию
          event++; // переходим к следующему событию
        }
      }
      return true;
    }

    private int findNextStartTime(int event, int time) {
      if (time < 0) return -1;

      int duration = events.get(event).type.duration;
      for (int i = time; i < dtcSize; i++) {
        DateTimeClass location = dtc.get(i);
        Event ev = events.get(event);
        Group group = ev.group;
        Set<Teacher> teachers = ev.teacher;
        boolean success = false;

        boolean teacherWishes = teachers.stream().mapToInt(teacher -> {
          if (teacher.date.stream().anyMatch(dt -> dt.equals(location.date))
          && teacher.time.stream().anyMatch(t -> t == location.time)
```

```
          && teacher.time.stream().anyMatch(t -> t == location
             .time + duration)) return 0;
          else return 1;
        }).sum() == 0;
        if (ignoreWishes[event]) {
          teacherWishes = !teacherWishes;
        }
        if (solution[i] == -1 // аудитория в это время свободн
           а
        && ev.wishedClassroomType <= location.classroom.type
           //есть ли в аудитории нужное оборудование
        && group.size <= location.classroom.size //влезает ли
           группа в аудиторию
        // не слишком ли сейчас поздно для начала проведения д
           лительного события
        && i + duration < dtcSize
        && location.classroom.num == dtc.get(i + duration).
           classroom.num
        && location.date.equals(dtc.get(i + duration).date)
        // подходит ли дата и время преподавателю
        && teacherWishes
        ) {
          Map<String, Integer> teacherTime = new HashMap<>();
          int countPerDay = 0;
          for (int k = 0; k < dtcSize; k++) {
            // если время-место свободны, они не повлияют на о
               граничения
            if (solution[k] == -1) continue;

            Event currentEvent = events.get(solution[k]);
            DateTimeClass currentLocation = dtc.get(k);

            // если проверяемая аудитория в это время занята
            if (location.date.compareTo(currentLocation.date)
               == 0) {
              break;
            }
            // если у преподавателя в этот день уже были занят
               ия
            if (currentLocation.date == location.date) {
              for (Teacher curT : currentEvent.teacher) {
                for (Teacher teacher : teachers) {
                  if (curT.name.equals(teacher.name)) {
                    if (teacherTime.containsKey(teacher.name))
                       {
```

```
                    teacherTime.put(teacher.name, 1);
                } else {
                    teacherTime.compute(teacher.name, (key,
                        v) -> v + 1);
                }
                // если преподаватель уже достаточно отраб
                    отал в этот день.
                if (teacherTime.get(teacher.name) >
                    maxTimePerDay - duration) {
                  break;
                }
              }
            }
          }
        }
        if (currentEvent.group.group.equals(group.group))
            {
          //если в этот день уже были занятия у этой групп
              ы
          if (currentEvent.type.type.equals(ev.type.type)
              && location.date == currentLocation.date) {
            countPerDay++;
            if (ev.type.countPerDay < countPerDay) {
              break;
            }
          }
          //если до или после события есть другое событие,
              до которого меньше "отдыха", чем нужно
          if (location.date.minusDays(ev.type.pauseBefore)
              .compareTo(currentLocation.date) <= 0
          && location.date.plusDays(ev.type.pauseAfter).
              compareTo(currentLocation.date) >= 0) {
            break;
          }

        }
        if (k == dtcSize - 1) success = true;
      }
    }
    if (success) return i;
  }
  return -1;
}

private void cleanEvent(int event) {
```

```
      for (int i = 0; i < dtcSize; i++) {
        if (solution[i] == event) solution[i] = -1;
      }
    }

    // бронинирование за событием помещения и времени
    private void submitDateTimeClass(int event, int nextTime)
       {
      int duration = events.get(event).type.duration;
      for (int i = nextTime; i < nextTime + duration; i++) {
        solution[i] = event;
      }
    }

    // возвращает следующий свободный индекс DateTimeClass для
        указанного события
    private int getTime(int event) {
      int min = -1;
      boolean contains = false;
      for (int i = 0; i < dtcSize - 1; i++) {
        // если аудитория в это время ещё никем не забронирова
           на
        if (solution[i] == -1) {
          if (contains) {
            return i;
          } else if (min == -1) {
            min = i;
          }
        }
        // если это событие уже бронировало какое-то время и м
           есто
        if (solution[i] == event) {
          if (contains) return i;
          contains = true;
        }
      }
      // если последняя ячейка была занята текущим событием, т
         о следующей для него нет
      if (contains) return -1;
      return min;
    }
  }
```

Приложение 2

Классы для хрранения входных данных

```
public class ClassConstraint {
  public Integer room;
  public List<Integer> dof;
  public List<Integer> time;

  public ClassConstraint(Integer room, List<Integer> dof, List
     <Integer> time) {
    this.room = room;
    this.dof = dof;
    this.time = time;
  }
}
public class AttestationType {
  public String type;
  public int pauseBefore;
  public int pauseAfter;
  public int duration;
  public int countPerDay;

  public AttestationType(String type, int pauseBefore, int
     pauseAfter, int duration, int countPerDay) {
    this.type = type;
    this.pauseBefore = pauseBefore;
    this.pauseAfter = pauseAfter;
    this.duration = duration;
    this.countPerDay = countPerDay;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    AttestationType that = (AttestationType) o;
    return Objects.equals(type, that.type);
  }

  @Override
  public int hashCode() {
    return Objects.hash(type);
  }
```

```
}

public class Classroom {
  public int num;
  public int type;
  public int size;

  public Classroom(int num, int type, int size) {
    this.num = num;
    this.type = type;
    this.size = size;
  }
}

public class DateTime {
  LocalDate date;
  Integer time;

  public DateTime(LocalDate date, Integer time) {
    this.date = date;
    this.time = time;
  }
}

public class DateTimeClass {
  public LocalDate date;
  public int time;
  public Classroom classroom;

  public DateTimeClass(LocalDate date, int time, Classroom
     classroom) {
    this.date = date;
    this.time = time;
    this.classroom = classroom;
  }

  @Override
  public String toString() {
    return "DateTimeClass{" +
      "date=" + date.toString() +
      ", time=" + time +
      ", classroom=" + classroom.num +
      '}';
  }
```

```
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    DateTimeClass that = (DateTimeClass) o;
    return time == that.time && Objects.equals(date, that.date
       ) && Objects.equals(classroom, that.classroom);
  }

  @Override
  public int hashCode() {
    return Objects.hash(date, time, classroom);
  }
}


public class Event implements Comparable {
  public Group group;
  public Set<Teacher> teacher;
  public AttestationType type;
  public String course;
  public int wishedClassroomType;

  public Event(Group group, Set<Teacher> teacher,
     AttestationType type, String course, int
     wishedClassroomType) {
    this.group = group;
    this.teacher = teacher;
    this.type = type;
    this.course = course;
    this.wishedClassroomType = wishedClassroomType;
  }

  // чтобы при сортировке сначала был больший приоритет
  @Override
  public int compareTo(Object o) {
    Event event = (Event) o;
    return -Integer.compare(Teacher.maxPrior(this.teacher),
       Teacher.maxPrior(event.teacher));
  }

  @Override
  public String toString() {
    return "Event{" +
      "group=" + group.group +
```

```
        ", teacher=" + teacher.toString() +
        ", type=" + type +
        ", course='" + course + '\'' +
        ", wishedClassroomType=" + wishedClassroomType +
        '}';
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Event event = (Event) o;
    return Objects.equals(group.group, event.group.group) &&
       Objects.equals(teacher, event.teacher) && Objects.
       equals(type.type, event.type.type) && Objects.equals(
       course, event.course);
  }

  @Override
  public int hashCode() {
    return Objects.hash(group, teacher, type, course);
  }
}

public class Group {
  public String group;
  public int size;

  public Group(String group, int size) {
    this.group = group;
    this.size = size;
  }
}

public class Solution {
  public Event event;
  public DateTimeClass dtc;

  public Solution(Event event, DateTimeClass dtc) {
    this.event = event;
    this.dtc = dtc;
  }

  @Override
  public String toString() {
```

```
    return event.group.group + "," + event.teacher.toString()
       +","+event.course+","+event.type.type+","+
    dtc.date+ ","+ dtc.time+","+ dtc.classroom.num;
  }
}


public class Teacher {
  public String name;
  public List<LocalDate> date;
  public List<Integer> time;
  public int prior;

  public Teacher(String name, List<LocalDate> date, List<
     Integer> time, int prior) {
    this.name = name;
    this.date = date;
    this.time = time;
    this.prior = prior;
  }

  public static int maxPrior(Set<Teacher> set){
    return set.stream().mapToInt(x -> x.prior).max().orElse(0)
       ;
  }

  @Override
  public String toString() {
    return name;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Teacher teacher = (Teacher) o;
    return Objects.equals(name, teacher.name);
  }

  @Override
  public int hashCode() {
    return Objects.hash(name);
  }

  public int compareTo(Teacher teacher) {
```

```
    return Integer.compare(this.prior, teacher.prior);
  }
}

public class TeacherDate {
  public String name;
  public LocalDate date;

  public TeacherDate(String name, LocalDate date) {
    this.name = name;
    this.date = date;
  }
}


public class TeacherPrior {
  public String name;
  public Integer prior;

  public TeacherPrior(String name, Integer prior) {
    this.name = name;
    this.prior = prior;
  }
}


public class TeacherTime {
  public String name;
  public Integer time;

  public TeacherTime(String name, Integer time) {
    this.name = name;
    this.time = time;
  }
}
```

Приложение 3

Модуль взаимодействия с API ruz.spbstu.ru

```
public class ApiRuz {
  public Set<Rasp> rasp;
  public Set<Rasp> raspPI;
  public Set<Rasp> raspMO;
  public List<Group> pi;
  public List<Group> mo;

      public ApiRuz() {
        raspPI = new HashSet<>();
        raspMO = new HashSet<>();
        rasp = new HashSet<>();
        pi = new ArrayList<>();
        mo = new ArrayList<>();
        RuzSpbStu.getGroupsbyFacultyId(95).forEach(x -> {
          if (x.getNameGroup().startsWith("3530903/") ) {
                pi.add(x);
              }
              if (x.getNameGroup().startsWith("3530203/")){
                    mo.add(x);
                  }
                });
                for (Group g : pi) {
                  int idGroup = g.getIdGroup();
                  String groupName = g.getNameGroup();
                  int level = g.getLevel();
                  for (int i = 0; i < 3; i++) {
                    java.util.ArrayList<Day> days = RuzSpbStu.
                       getScheduleByGroupIdAndDate(idGroup,
                       LocalDate.now().minusWeeks(i)).getDays
                       ();
                    for (Day d : days) {
                      List<Lesson> lessons = d.getLessons();
                      for (Lesson x : lessons) {
                        String sub = x.getSubject();
                        List<Teacher> t = x.getTeachers();
                        if (t != null) {
                          Set<String> set = new HashSet<>();
                          for (Teacher te : t) {
                            set.add(te.getFullName());
                          }
```

```
                        Rasp r = new Rasp(set, groupName,
                           sub, level);
                        if (raspPI.contains(r)) {
                          for (Rasp y : raspPI) {
                            if (y.equals(r)) {
                              Set<String> n = y.teacherName;
                              set.addAll(n);
                              raspPI.remove(y);
                              break;
                            }
                          }
                        }
                        Rasp ra = new Rasp(set, groupName,
                           sub, level);
                        raspPI.remove(ra);
                        raspPI.add(ra);
                        rasp.remove(ra);
                        rasp.add(ra);
                      }
                    }
                  }
                }

                for (Group g : mo) {
                  int idGroup = g.getIdGroup();
                  String groupName = g.getNameGroup();
                  int level = g.getLevel();
                  for (int i = 0; i < 3; i++) {
                    java.util.ArrayList<com.fleshka4.spbstu.
                       ruz.api.models.Day> days = RuzSpbStu.
                       getScheduleByGroupIdAndDate(idGroup,
                       LocalDate.now().minusWeeks(i))
                    .getDays();
                    for (Day d : days) {
                      List<Lesson> lessons = d.getLessons();
                      for (Lesson x : lessons) {
                        String sub = x.getSubject();
                        List<Teacher> t = x.getTeachers();
                        if (t != null) {
                          Set<String> set = new HashSet<>();
                          for (Teacher te : t) {
                            set.add(te.getFullName());
                          }
```

```
                        Rasp r = new Rasp(set, groupName,
                            sub, level);
                        if (raspMO.contains(r)) {
                          for (Rasp y : raspMO) {
                            if (y.equals(r)) {
                              Set<String> n = y.teacherName;
                              set.addAll(n);
                              break;
                            }
                          }
                        }
                        Rasp ra = new Rasp(set, groupName,
                            sub, level);
                        raspMO.remove(ra);
                        raspMO.add(ra);
                        rasp.remove(ra);
                        rasp.add(ra);
                      }
                    }
                  }
                }
              }

              raspMO.forEach(System.out::println);
              raspPI.forEach(System.out::println);
            }

            public Map<String, Integer> getTeachersNames() {
              Map<String, Integer> names = new HashMap<>();
              for (Rasp r : raspPI) {
                r.teacherName.forEach(t -> names.put(t,
                    names.getOrDefault(t, 0)));
              }
              for (Rasp r : raspMO) {
                r.teacherName.forEach(t -> names.put(t,
                    names.getOrDefault(t, 0)));
              }
              return names;
            }
          }
```